30th International Symposium on Theoretical Aspects of Computer Science

STACS'13, February 27th to March 2nd, 2013, Kiel, Germany

Edited by

Natacha Portier
Thomas Wilke

LIPIcs – Vol. 20 – STACS'13 www.dagstuhl.de/lipics

Editors

Natacha Portier	Thomas Wilke
École Normale Supérieure de Lyon	Christian-Albrechts-Universität zu Kiel
Lyon	Kiel
`natacha.portier@ens-lyon.fr`	`wilke@ti.informatik.uni-kiel.de`

ACM Classification 1998
F.1.1 Models of Computation, F.2.2 Nonnumerical Algorithms and Problems, F.4.1 Mathematical Logic, F.4.3 Formal Languages, G.2.1 Combinatorics, G.2.2 Graph Theory

ISBN 978-3-939897-50-7

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-939897-50-7.

Publication date
February, 2013

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/LIPIcs.STACS.2013.i

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

www.dagstuhl.de/lipics

Foreword

The Symposium on Theoretical Aspects of Computer Science (STACS) is an international forum for original and unpublished research on theoretical aspects of computer science. Typical areas are (cited from the call for papers for this year's conference):

- algorithms and data structures, including: parallel, distributed, approximation, and randomized algorithms, computational geometry, cryptography, algorithmic learning theory, analysis of algorithms;
- automata and formal languages, games;
- computational complexity, randomness in computation;
- logic in computer science, including: semantics, specification and verification, rewriting and deduction;
- current challenges, for example: natural computing, quantum computing, mobile and net computing.

STACS has taken place each year since 1984, alternately in Germany and France. The conference in Kiel from February 27 through March 2, 2013, is the 30th in this series: Paris (1984), Saarbrücken (1985), Orsay (1986), Passau (1987), Bordeaux (1988), Paderborn (1989), Rouen (1990), Hamburg (1991), Cachan (1992), Würzburg (1993), Caen (1994), München (1995), Grenoble (1996), Lübeck (1997), Paris (1998), Trier (1999), Lille (2000), Dresden (2001), Antibes (2002), Berlin (2003), Montpellier (2004), Stuttgart (2005), Marseille (2006), Aachen (2007), Bordeaux (2008), Freiburg (2009), Nancy (2010), Dortmund (2011), Paris (2012), and Kiel (2013).

The interest in STACS has remained at a high level over the past years; STACS 2013 received 254 submissions from 41 countries. (Authors were asked to submit an extended abstract of at most 12 pages; missing proofs had to be put into an appendix.) In the selection process, 54 submissions were selected for presentation and publication, which implies an acceptance rate of about 21%. The selection of the contributions was carried out in a two-phase process in autumn 2012: over a period of eight weeks, every paper was reviewed by three members of the program committee, who, at their discretion, involved external reviewers; over a period of four weeks, intensive discussions within the program committee, structured in five rounds, led to the selection of the papers published in this volume. The overall very high quality of the submissions made the selection a difficult task.

As co-chairs of the program committee, we would like to sincerely thank our colleagues for having submitted to STACS such a great number of excellent papers, our co-members of the program committee (see list on page vii) and the many external reviewers (see list on page ix) for their valuable work in assessing the merits of each individual submission. We would like to express our thanks to the three invited speakers, Kousha Etessami, Kurt Mehlhorn, and Stéphan Thomassé, and to Dániel Marx, the invited tutorial speaker. Special thanks go to Andrei Voronkov for providing EasyChair, the software used for processing and screening submissions to the conference.

We would like to warmly thank Henning Schnoor and Björn Kinscher for preparing these conference proceedings, and Michael Wagner and Marc Herbstritt from the Dagstuhl/LIPIcs team for assisting us in the publication process and the final production of the proceedings.

These proceedings contain extended abstracts of the accepted contributions and abstracts of the invited talks and the tutorial. The authors have retained their rights and make their work available under a Creative Commons license. The proceedings are published electronically by Schloss Dagstuhl – Leibniz-Center for Informatics within their LIPIcs series;

they are accessible through several portals, in particular, DROPS and HAL. Both, DROPS and HAL, guarantee perennial and easy electronic access as well as indexing in DBLP and Google Scholar.

Lyon and Kiel, February 2013 *Natacha Portier and Thomas Wilke*

Program Committee

Eric Allender	Rutgers University
Pablo Barceló	Universidad de Chile
Frédérique Bassino	Université Paris 13
Artur Czumaj	University of Warwick
Hervé Fournier	Université Paris Diderot
Edward A. Hirsch	Steklov Institute, St. Petersburg
Iordanis Kerenidis	Université Paris Diderot
Michal Koucký	Czech Academy of Sciences
Dieter Kratsch	Université de Lorraine
Andrei Krokhin	University of Durham
Antonín Kučera	Masaryk University
Markus Lohrey	University of Leipzig
Katarzyna Paluch	University of Wrocław
Natacha Portier	École Normale Supérieure de Lyon (co-chair)
Kirk Pruhs	University of Pittsburgh
Peter Rossmanith	RWTH Aachen
Günter Rote	Freie Universität Berlin
Thomas Sauerwald	MPI Saarbrücken
Sandeep Sen	IIT Delhi
Subhash Suri	UC Santa Barbara
Jacobo Toran	Universität Ulm
Jouko Väänänen	University of Helsinki and University of Amsterdam
Thomas Wilke	Christian-Albrechts-Universität zu Kiel (co-chair)
Carsten Witt	Technical University of Denmark
Gerhard J. Woeginger	TU Eindhoven
Marc Zeitoun	Université de Bordeaux I

External Reviewers

Anna Adamaszek
Isolde Adler
Pankaj Agarwal
Hee-Kap Ahn
Kook Jin Ahn
Alex Andoni
Spyridon Antonakopoulos
Antonios Antoniadis
Marcelo Arenas
Pablo Arrighi
Samalam Arun-Kumar
Dror Atariah
Peter Auer
David Auger
Per Austrin
Pranjal Awasthi
Jossi Azar
Haris Aziz
Martin Babka
Sang Won Bae
Nikhil Bansal
David Mix Barrington
Libor Barto
Surender Baswana
Cristina Bazgan
Marie-Pierre Béal
Chris Beck
Florent Becker
Jason Bell
Shai Ben-David
Petra Berenbrink
Thierry Berger
Christoph Berkholz
Christoper Berlind
Dietmar Berwanger
Olaf Beyersdorff
Peter Biró
Henrik Bjrklund
Ivan Bliznets
Achim Blumensath
Hans L. Bodlaender
Benedikt Bollig
Paul Bonsma
Andreas Brandstadt
Tomas Brazdil
Romain Brenguier
Karl Bringmann
Christopher Broadbent
Joshua Brody
Hajo Broersma
Nicolas Broutin
Yuriy Brun
Maike Buchin
Peter Buergisser
Jan Bulanek
Jarosław Byrka
Sergio Cabello
Jin-Yi Cai
Alan Cain
Olivier Carton
David Cash
Julien Cassaigne
Katarína Cechlárová
Eranda Cela
Amit Chakrabarti
Deeparnab Chakrabarty
Jérémie Chalopin
Shiri Chechik
Ho-Lin Chen
Hubie Chen
Yann Chevaleyre
Marek Chrobak
Lorenzo Clemente
Albert Cohen
Amin Coja-Oghlan
Thomas Colcombet
Daniel Cole
Seshadhri Comandur
Hubert Comon
Anne Condon
Joshua Cooper
Graham Cormode
Erzsébet Csuhaj-Varjú
Peter Damaschke
Carsten Damm
Stefan Dantchev
Samir Datta
Laure Daviaud
Anuj Dawar
Vladimir Deineko
Holger Dell
Evgeny Demenkov
Hans de Nivelle
Nicolas de Rugy-Altherre
Ronald de Wolf
Michael Dinitz
Emilie Diot
Benjamin Doerr
David Doty
Philippe Duchon
Arnaud Durand
Christoph Durr
Sebastian Eggert
Khaled Elbassioni
Michael Elberfeld
Edith Elkind
Robert Elsaesser
Matthias Englert
David Eppstein
Leah Epstein
Jeff Erickson
Kousha Etessami
Oriol Farras
John Fearnley
Tomas Feder
Sandor Fekete
Mike Fellows
Shiguang Feng
Henning Fernau
Esteban Feuerstein
Arnaud Fietzke
Diego Figueira
Jeremy Fineman
Johannes Fischer
Fedor Fomin
Vojtech Forejt
Anna Frid
Tom Friedetzky
Tobias Friedrich
Matteo Frigo
Eric Fusy
Sajith G.
Travis Gagie
Martin Gairing
Robert Ganian
Naveen Garg
Luisa Gargano
William Gasarch

Serge Gaspers
Paul Gastin
Pawel Gawrychowski
Antoine Genitrini
Shayan Gharan
Amelie Gheerbrant
George Giakkoupis
Panos Giannopoulos
Matt Gibson
Alexander Gilbers
Hugo Gimbert
Marc Glisse
Marc Goerigk
Leslie Ann Goldberg
Stefan Göller
Petr Golovach
Alexander Golovnev
Eric Goubault
Olga Goussevskaia
Navin Goyal
Erich Grädel
Fabrizio Grandoni
Gianluigi Greco
Guiseppe Greco
Bruno Grenet
Martin Grohe
Romain Grunert
Sudipto Guha
Jiong Guo
Venkatesan Guruswami
Carsten Gutwenger
Christoph Haase
Peter Habermehl
Guillaume Hanrot
Tero Harju
Prahladh Harsha
Meng He
Brent Heeringa
Pinar Heggernes
Lauri Hella
Matthias Henze
Frédéric Herbreteau
Volker Heun
Thomas Hildebrandt
Hiroshi Hirai
Petr Hlineny
Jack H. Lutz
Martin Hoefer
Frank Hoffmann
Michael Hoffmann
Hendrik Jan Hoogeboom
Florian Horn
Peter Høyer
Wen-Lian Hsu
Bert Huang
Chien-Chung Huang
Falk Hüffner
Yumei Huo
Sungjin Im
Shunsuke Inenaga
R. Inkulu
Dmitry Itsykson
Ragesh Jaiswal
Wojtek Jamroga
Petr Jancar
Svante Janson
Jesper Jansson
Matti Järvisalo
Rafel Jaume
Emmanuel Jeandel
Stacey Jeffery
Artur Jeż
Łukasz Jeż
Colin Jia Zheng
Matthew Johnson
Peter Jonsson
Allan Jørgensen
Hossein Jowhari
Eun Jung Kim
Tomasz Jurdzinski
Marcin Kaminski
Iyad Kanj
Sampath Kannan
Mamadou Moustapha Kante
Haim Kaplan
George Karakostas
Juhani Karhumäki
Lila Kari
Alexander Kartzow
Petteri Kaski
Michael Kaufmann
Viv Kendon
Balázs Keszegh
Emanuel Kieronski
Daniel Kirsten
Ralf Klasing
Marek Klonowski
Christian Knauer
Alexander Knop
Yasuaki Kobayashi
Pascal Koiran
Mikko Koivisto
Christian Komusiewicz
Ranganath Kondapally
Arnd Christian König
Christian Konrad
Guy Kortsarz
Adrian Kosowski
Arie Koster
Ilias Kotsireas
Timo Kötzing
Ioannis Koutis
Lukasz Kowalik
Dariusz Kowalski
Stefan Kratsch
Jan Kretinsky
Stephan Kreutzer
Klaus Kriegel
Adrian Kügel
Alexander Kulikov
Oliver Kullmann
Amit Kumar
Michal Kunc
Robin Künzler
Stuart Kurtz
Dietrich Kuske
Eduardo Laber
Jakub Łącki
Alexander Langer
Sophie Laplante
Kasper Green Larsen
Silvio Lattanzi
Van Bang Le
Per Kristian Lehre
Thierry Lecroq
Troy Lee
Virginie Lerays
Jérôme Leroux
Fei Li
Yingyu Liang
Mathieu Liedloff
Nutan Limaye
Andrzej Lingas
Simone Linz
Kamal Lodaya
Christof Löding
Daniel Lokshtanov
Sylvain Lombardy
Krzysztof Loryś
Kerkko Luosto

Frederic Magniez
Meena Mahajan
Konstantin Makarychev
Guillaume Malod
Florin Manea
Sebastian Maneth
David Manlove
Yishay Mansour
Jan Manuch
Jerzy Marcinkowski
Jakub Marecek
Nicolas Markey
Conrado Martínez
Dániel Marx
Maarten Marx
Jannik Matuschke
Alexander May
Elvira Mayordomo
Julian McAuley
Catherine McCartin
Guy McCusker
Andrew McGregor
Pierre McKenzie
Daniel Meister
Stefan Mengel
George Mertzios
Julian Mestre
Dimitrios Michail
Filippo Mignosi
Tillmann Miltzow
Pradipta Mitra
Rajat Mittal
Matthias Mnich
Tobias Moemke
Ankur Moitra
Debajyoti Mondal
Herve Moulin
Shay Mozes
Marcin Mucha
Markus Mueller
Mike Müller
Wolfgang Mulzer
Viswanath Nagarajan
Satyadev Nandakumar
Ashwin Nayak
Blaine Nelson
Alantha Newman
Cyril Nicaud
André Nichterlein
Rolf Niedermeier
Andre Nies
Harumichi Nishimura
Petr Novotný
Dirk Nowotka
Jan Obdrzalek
Alexander Okhotin
Vsevolod Oparin
Martin Otto
Joel Ouaknine
Sang-Il Oum
Ozgur Ozkan
David Pal
Konstantinos Panagiotou
Dana Pardubska
Francesco Pasquale
Boaz Patt-Shamir
Christophe Paul
Daniel Paulusma
Sylvain Perifel
Giovanni Pighizzini
Marcin Pilipczuk
Jean-Eric Pin
Marek Piotrów
Thomas Place
Wojciech Plandowski
James Plank
Libor Polak
Amaury Pouly
Ali Pourmiri
Sanjiva Prasad
Sebastian Preugschat
Pavel Pudlak
Gabriele Puppis
Karin Quaas
Harald Räcke
Prasad Raghavendra
Sanguthevar Rajasekaran
Rajeev Raman
Michael Rao
Igor Razgon
Vojtech Rehak
Felix Reidl
Klaus Reinhardt
Gwenaël Richomme
Inbal Rika
Oliver Riordan
Liam Roditty
Andrei Romashchenko
Adi Rosen
Dominique Rossin
Jörg Rothe
Sambuddha Roy
Sasha Rubin
Luis M. S. Russo
Bartosz Rybicki
Aleksi Saarela
Kalle Saari
Yogish Sabharwal
Mathieu Sablik
Ben Sach
Amit Sahai
Kai Salomaa
Rahul Santhanam
Palash Sarkar
Saket Saurabh
Dmytro Savchuk
Nicolas Schabanel
Ludmila Scharf
Ingo Schiermeyer
Lena Schlipf
Ildi Schlotter
Sylvain Schmitz
Henning Schnoor
Uwe Schöning
Luc Segoufin
Shinnosuke Seki
Géraud Sénizergues
Jiří Sgall
Jeffrey Shallit
Asaf Shapira
Alexander Shen
Yaoyun Shi
Somnath Sikdar
Jamie Sikora
D Sivakumar
Daniel Sleator
Michiel Smid
Christian Sohler
Dmitry Sokolov
Troels Bjerre Sørensen
Manuel Sorge
Michèle Soria
Frits Spieksma
Andrea Sportiello
Jiri Srba
Piyush Srivastava
Grzegorz Stachowiak
Tatiana Starikovskaya
Ulrike Stege
Howard Straubing

Ileana Streinu
Jan Strejcek
Yann Strozecki
Ondrej Suchy
Mario Szegedy
Tony Tan
Bangsheng Tang
Sébastien Tavenas
Jan Arne Telle
Sebastiaan Terwijn
Olivier Teytaud
Guillaume Theyssier
Thomas Thierauf
Dimitrios Thilikos
Ioan Todinca
Théophile Trunck
Madhur Tulsiani
Christos Tzamos
Salil Vadhan
Rene van Bevern
Jan Van Den Bussche
Rob van Stee
Anke van Zuylen
Kasturi Varadarajan
Moshe Vardi
Santosh Vempala
Stéphane Vialette
Antoine Vigneron
Fernando Sanchez Villaamil
Mikhail Volkov
Jan Vondrak
Pedro V. Silva
Mikhail Vyalyi
Kira Vyatkina
Magnus Wahlström
Pascal Weil
Oren Weimann
Matthias Westermann
Chris Whidden
Bryan T. Wilkinson
Anthony Widjaja Lin
Andreas Wiese
Ryan Williams
Philipp Woelfel
Christian Wulff-Nilsen
David Xiao
Tomoyuki Yamakami
Li Yan
Yitong Yin
Jia Yuan Yu
Henry Yuen
Filip Zagorski
Faried Abu Zaid
Hans Zantema
Thomas Zeugmann
Thomas Zeume
Lisa Zhang
Qin Zhang
Yair Zick
Rosalba Zizza

Table of Contents

30th Internatioanl Symposium on Theoretical Aspects of Computer Science (STACS'13).
Editors: Natacha Portier and Thomas Wilke
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The complexity of analyzing infinite-state Markov chains, Markov decision processes, and stochastic games

Kousha Etessami

School of Informatics, University of Edinburgh, UK
kousha@inf.ed.ac.uk

Abstract

In recent years, a considerable amount of research has been devoted to understanding the computational complexity of basic analysis problems, and model checking problems, for finitely-presented countable infinite-state probabilistic systems. In particular, we have studied *recursive Markov chains* (RMCs), *recursive Markov decision processes* (RMDPs) and *recursive stochastic games* (RSGs). These arise by adding a natural recursion feature to finite-state Markov chains, MDPs, and stochastic games. RMCs and RMDPs provide natural abstract models of probabilistic procedural programs with recursion, and they are expressively equivalent to probabilistic and MDP extensions of pushdown automata. Moreover, a number of well-studied stochastic processes, including multi-type branching processes, (discrete-time) quasi-birth-death processes, and stochastic context-free grammars, can be suitably captured by subclasses of RMCs.

A central computational problem for analyzing various classes of recursive probabilistic systems is the computation of their (optimal) *termination probabilities*. These form a key ingredient for many other analyses, including model checking. For RMCs, and for important subclasses of RMDPs and RSGs, computing their termination values is equivalent to computing the *least fixed point* (LFP) solution of a corresponding *monotone system of polynomial (min/max) equations*. The complexity of computing the LFP solution for such equation systems is a intriguing problem, with connections to several areas of research. The LFP solution may in general be irrational. So, one possible aim is to compute it to within a desired additive error $\epsilon > 0$. For general RMCs, approximating their termination probability within *any* non-trivial constant additive error $< 1/2$, is at least as hard as long-standing open problems in the complexity of numerical computation which are not even known to be in NP. For several key subclasses of RMCs and RMDPs, computing their termination values turns out to be much more tractable.

In this talk I will survey algorithms for, and discuss the computational complexity of, key analysis problems for classes of infinite-state recursive MCs, MDPs, and stochastic games. In particular, I will discuss recent joint work with Alistair Stewart and Mihalis Yannakakis (in papers that appeared at STOC'12 and ICALP'12), in which we have obtained polynomial time algorithms for computing, to within arbitrary desired precision, the LFP solution of probabilistic polynomial (min/max) systems of equations. Using this, we obtained the first P-time algorithms for computing (within desired precision) the extinction probabilities of multi-type branching processes, the probability that an *arbitrary* given stochastic context-free grammar generates a given string, and the optimum (maximum or minimum) extinction probabilities for branching MDPs and context-free MDPs. For branching MDPs, their corresponding equations amount to Bellman optimality equations for minimizing/maximizing their termination probabilities. Our algorithms combine variations and generalizations of Newton's method with other techniques, including linear programming. The algorithms are fairly easy to implement, but analyzing their worst-case running time is mathematically quite involved.

1998 ACM Subject Classification F.2.1 Theory of Computing, G.3 Probability and Statistics, G.1.5 Mathematics of Computing

Keywords and phrases recursive Markov chains, Markov decision processes, stochastic games, monotone systems of nonlinear equations, least fixed points, Newton's method, computational complexity, multi-type branching processes, stochastic context-free grammars

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.1

Graph coloring, communication complexity, and the stubborn problem

Nicolas Bousquet[1], **Aurélie Lagoutte**[2], **and Stéphan Thomassé**[3]

1 **AlGCo project-team, CNRS, LIRMM, Montpellier, France.**
`nicolas.bousquet@lirmm.fr`

2,3 **LIP, UMR 5668 ENS Lyon - CNRS - UCBL - INRIA, Université de Lyon, France.**
`{aurelie.lagoutte, stephan.thomasse}@ens-lyon.fr`

Abstract

We discuss three equivalent forms of the same problem arising in communication complexity, constraint satisfaction problems, and graph coloring. Some partial results are discussed.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases stubborn problem, graph coloring, Clique-Stable set separation, Alon-Saks-Seymour conjecture, bipartite packing

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.3

1 The equivalent forms

A classical result of Graham and Pollak [5] asserts that the edge set of the complete graph on n vertices cannot be partitioned into less than $n-1$ complete bipartite graphs. A natural question is then to ask for some properties of graphs G_ℓ which are edge-disjoint unions of ℓ complete bipartite graphs. An attempt in this direction was proposed by Alon, Saks and Seymour, asking if the chromatic number of G_ℓ is at most $\ell+1$. This wild generalization of Graham and Pollak's theorem was however disproved by Huang and Sudakov [6] who provided graphs with chromatic number $\Omega(\ell^{6/5})$. The $O(\ell^{\log \ell})$ upper-bound being routine to prove, this leaves as open question the *polynomial Alon-Saks-Seymour conjecture* asking if an $O(\ell^c)$ coloring exists for some fixed c.

A communication complexity problem introduced by Yannakakis[8] involves a graph G of size n and the usual suspects Alice and Bob. Alice plays on the stable sets of G and Bob plays on the cliques. Their goal is to exchange the minimum amount of information to decide if Alice's stable set S intersect Bob's clique K. In the nondeterministic version, one asks for the minimum size of a certificate one should give to Alice and Bob to decide whether S intersects K. If indeed S intersects K, the certificate consists in the vertex $x = S\cap K$, hence one just has to describe x, which costs $\log n$. The problem becomes much harder if one want to certify that $S \cap K = \emptyset$ and this is the core of this problem. A natural question is to ask for a $O(\log n)$ upper bound. Yannakakis observed that this would be equivalent to the following *polynomial clique-stable separation conjecture*: There exists a c such that for any graph G on n vertices, there exists $O(n^c)$ vertex bipartitions of G such that for every disjoint stable set S and clique K, one of the bipartitions separates S from K.

A variant of Feder and Vardi celebrated dichotomy conjecture for Constraint Satisfaction Problems, the List Matrix Partition (LMP) problem, see [3], asks whether all $(0,1,*)$ CSP instances are NP–complete or polytime solvable. The LMP was investigated for small matrices, and was completely solved in dimension 4, save for a unique case, known as the *stubborn problem* [1]: Given a complete graph G which edges are labelled by 1,2, or 3, the question is to partition the vertices into three classes V_1, V_2, V_3 so that V_i does not span an edge labelled i. An easy branching majority algorithm computes $O(n^{\log n})$ 2-list-assignments of the vertices such that every solution of the stubborn problem is covered by at least one of these 2-list-assignments. The stubborn problem hence reduces into $O(n^{\log n})$ 2-SAT instances, yielding a pseudo polynomial algorithm. A polynomial algorithm was recently discovered by Cygan et al.[2], but whether the original branching algorithm could be turned into a polynomial algorithm is still open. Precisely one can ask the *polynomial stubborn 2-list cover conjecture* asking if the set of solutions of any instance of the stubborn problem can be covered by $O(n^c)$ instances consisting of lists of size 2.

We show that these three problems are indeed equivalent. One direction was already proved by Alon and Haviv.

We further discuss the polynomial clique-stable separation conjecture by restricting ourselves to some classes of graphs. An open problem of Lovász, see for instance [7], asks for an extending formulation of the stable set polytope of perfect graphs. This does not exist for all graphs, as Fiorini [4] et al. have recently proved, but special classes enjoy this property, like for instance comparability graphs. Furthermore, the existence of extended formulations for perfect graphs would give a polynomial clique-stable set separation for this class, which is still open. It is unlikely that the polynomial clique-stable separation holds for the class of all graphs, hence we propose a milder version of it: for every graph H, the class of H-free graphs (in the induced sense) has the polynomial clique-stable set separation. This question is wide open for the cycle of length 5, this would imply the result for perfect graphs. However, we can show that if H is a split graph, i.e. the union of a clique and a stable set, then there exists a c depending on H such that the clique-stable set separation can be achieved with n^c cuts in the class of H-free graphs. Observe that H can be wild since the edges between the clique and the stable set are not specified, hence no structural description of H-free graphs can be used here. The key tools for finding these n^c cuts is VC-dimension.

References

1 K. Cameron, E. M. Eschen, C. T. Hoàng, and R. Sritharan. The complexity of the list partition problem for graphs. *SIAM J. Discrete Math.*, 21(4):900–929, 2007.

2 M. Cygan, M. Pilipczuk, M. Pilipczuk, and J. O. Wojtaszczyk. The stubborn problem is stubborn no more (a polynomial algorithm for 3-compatible colouring and the stubborn list partition problem). In Dana Randall, editor, *SODA*, pages 1666–1674. SIAM, 2011.

3 T. Feder, P. Hell, S. Klein, and R. Motwani. List partitions. *SIAM J. Discrete Math.*, 16(3):449–478, 2003.

4 S. Fiorini, S. Massar, S. Pokutta, H. R. Tiwary, and R. de Wolf. Linear vs. semidefinite extended formulations: exponential separation and strong lower bounds. In *STOC*, pages 95–106, 2012.

5 R. L. Graham and H. O. Pollak. On embedding graphs in squashed cubes. *Graph theory and applications*, 303:99–110, 1972.

6 H. Huang and B. Sudakov. A counterexample to the Alon-Saks-Seymour conjecture and related problems. *Combinatorica*, 32:205–219, 2012.

7 L. Lovász. Stable sets and polynomials. *Discrete Mathematics*, 124(1-3):137–153, 1994.

8 M. Yannakakis. Expressing combinatorial optimization problems by linear programs. *J. Comput. Syst. Sci.*, 43(3):441–466, 1991.

Physarum Computations

Kurt Mehlhorn[1]

1 Max Planck Institute for Informatics

Abstract

Physarum is a slime mold. It was observed over the past 10 years that the mold is able to solve shortest path problems and to construct good Steiner networks [9, 11, 8]. In a nutshell, the shortest path experiment is as follows: A maze is covered with mold and food is then provided at two positions s and t and the evolution of the slime is observed. Over time, the slime retracts to the shortest s-t-path. A video showing the wet-lab experiment can be found at `http://www.youtube.com/watch?v=tLO2n3YMcXw&t=4m43s`. We strongly recommend to watch this video.

A mathematical model of the slime's dynamic behavior was proposed in 2007 [10]. Extensive computer simulations of the mathematical model confirm the wet-lab findings. For the edges on the shortest path, the diameter converges to one, and for the edges off the shortest path, the diameter converges to zero.

We review the wet-lab and the computer experiments and provide a proof for these experimental findings. The proof was developed over a sequence of papers [6, 7, 4, 2, 1, 3]. We recommend the last two papers for first reading.

An interesting connection between Physarum and ant computations is made in [5].

1998 ACM Subject Classification G.2.2 Graph Theory (Path and circuit problems)

Keywords and phrases Biological computation, shortest path problems

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.5

References

1 Vincenzo Bonifaci. Physarum can compute shortest paths: A short proof. *Inf. Process. Lett.*, 113(1-2):4–7, 2013.

2 Vincenzo Bonifaci, Kurt Mehlhorn, and Girish Varma. Physarum can compute shortest paths. In *SODA*, pages 233–240, 2012. full version to appear in Journal of Theoretical Biology.

3 Michael Dirnberger and Kurt Mehlhorn. A convergence proof for nonuniform directed Physarum. January 2013.

4 Kentaro Ito, Anders Johansson, Toshiyuki Nakagaki, and Atsushi Tero. Convergence properties for the physarum solver. arXiv:1101.5249v1, January 2011.

5 Qi Ma, Anders Johansson, Atsushi Tero, Toshiyuki Nakagaki, and David J. T. Sumpter. Current reinforced random walks for biological problem solving. 2012.

6 T. Miyaji and Isamu Ohnishi. Mathematical analysis to an adaptive network of the plasmodium system. *Hokkaido Mathematical Journal*, 36:445–465, 2007.

7 T. Miyaji and Isamu Ohnishi. Physarum can solve the shortest path problem on riemannian surface mathematically rigourously. *International Journal of Pure and Applied Mathematics*, 47:353–369, 2008.

8 T. Nakagaki, M. Iima, T. Ueda, Y. Nishiura, T. Saigusa, A. Tero, R. Kobayashi, and K. Showalter. Minimum-risk path finding by an adaptive amoebal network. *Physical Review Letters (PRL)*, 99(068104):4, 2007.

9 T. Nakagaki, H. Yamada, and Á. Tóth. Maze-solving by an amoeboid organism. *Nature*, 407:470, 2000.
10 A. Tero, R. Kobayashi, and T. Nakagaki. A mathematical model for adaptive transport network in path finding by true slime mold. *Journal of Theoretical Biology*, pages 553–564, 2007.
11 A. Tero, S. Takagi, T. Saigusa, K. Ito, D. Bebber, M. Fricker, K. Yumiki, R. Kobayashi, and T. Nakagaki. Rules for biologically inspired adaptive network design. *Science*, 327:439–442, 2010.

Algorithmic Graph Structure Theory

Dániel Marx

Computer and Automation Research Institute,
Hungarian Academy of Sciences (MTA SZTAKI),
Budapest, Hungary dmarx@cs.bme.hu

Abstract

The Graph Minors project of Robertson and Seymour uncovered a very deep structural theory of graphs. This theory had several important consequences, among others, the proof of Wagner's Conjecture. While the whole theory, presented in a series of 23 very dense papers, is notoriously difficult to understand, it has to be emphasized that these papers introduced several elementary concepts and tools that had strong impact on algorithms, complexity, and combinatorics. Moreover, even some of the very deep results can be stated in a compact and useful way, and it is possible to build upon these results without a complete understanding of the underlying machinery.

In the first part of the lecture, I will introduce the concept of treewidth, which can be thought of as an elementary entry point to graph minors theory. I will overview its graph-theoretic and algorithmic properties that make it especially important in the design of parameterized algorithms and approximation schemes on planar graphs. Furthermore, I will briefly explain some of the connections of treewidth to complexity and automata theory.

In the next part of the lecture, we will turn our attention to the more advanced topic of graphs excluding a fixed minor: the structure of such graphs, finding minors, and the well-quasi-ordering of the minor relation. The primary goal here is to provide clear and useful statements of these results and to show how they generalize the concepts of treewidth and planar graphs. Finally, I will briefly overview some more recent results involving different kinds of excluded structures, such as graphs excluding odd minors and topological minors.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases graph theory, graph minors, structure theorems

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.7

Searching for better fill-in*

Fedor V. Fomin[1] **and Yngve Villanger**[1]

1 **Department of Informatics, University of Bergen, Bergen, Norway,**
{fomin,yngve.villanger}@ii.uib.no

Abstract

Minimum Fill-in is a fundamental and classical problem arising in sparse matrix computations. In terms of graphs it can be formulated as a problem of finding a triangulation of a given graph with the minimum number of edges. By the classical result of Rose, Tarjan, Lueker, and Ohtsuki from 1976, an inclusion *minimal* triangulation of a graph can be found in polynomial time but, as it was shown by Yannakakis in 1981, finding a triangulation with the *minimum* number of edges is NP-hard.

In this paper, we study the parameterized complexity of local search for the Minimum Fill-in problem in the following form: Given a triangulation H of a graph G, is there a better triangulation, i.e. triangulation with less edges than H, within a given distance from H? We prove that this problem is fixed-parameter tractable (FPT) being parameterized by the distance from the initial triangulation by providing an algorithm that in time $\mathcal{O}(f(k)|G|^{\mathcal{O}(1)})$ decides if a better triangulation of G can be obtained by swapping at most k edges of H.

Our result adds Minimum Fill-in to the list of very few problems for which local search is known to be FPT.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases Local Search, Parameterized Complexity, Fill-in, Triangulation, Chordal graph

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.8

1 Introduction

A graph is *chordal* (or triangulated) if every cycle of length at least four contains a chord, i.e. an edge between non-adjacent vertices of the cycle. The Minimum Fill-in problem (also known as Minimum Triangulation and Chordal Graph Completion) is to turn a given graph into a chordal by adding as few new edges as possible. The name fill-in is due to the fundamental problem arising in sparse matrix computations which was studied intensively in the past. During Gaussian eliminations of large sparse matrices new non-zero elements called *fills* can replace original zeros thus increasing storage requirements and running time needed to solve the system. The problem of finding an optimal elimination ordering minimizing the number of fill elements can be expressed as the Minimum Fill-in problem on graphs [44, 45]. See also [9, Chapter 7] for a more recent overview of related problems and techniques. Besides sparse matrix computations, applications of Minimum Fill-in can be found in database management [3], artificial intelligence, and the theory of Bayesian statistics [8, 22, 32, 50]. The survey of Heggernes [25] gives an overview of techniques and applications of minimum and minimal triangulations.

* Supported by the European Research Council (ERC) via grant Rigorous Theory of Preprocessing, reference 267959, and the Research Council of Norway.

Minimum Fill-in (under the name Chordal Graph Completion) was one of the 12 open problems presented at the end of the first edition of Garey and Johnson's book [19] and it was proved to be NP-complete by Yannakakis [51]. While different approximation and parameterized algorithms for Minimum Fill-in were studied in the literature [2, 5, 7, 8, 17, 27, 38], in practice, to reduce the fill-in different *heuristic* ordering methods are commonly used. We refer to the recent survey of Duff and Bora [13] on the history and recent developments of fill-in reducing heuristics.

In this paper we study the following local search variant of the problem: given a fill-in of a graph, is it possible to obtain a better fill-in by changing a small number of edges? An efficient local search algorithm could be used as a generic subroutine of almost every fill-in heuristic.

The idea of local search is to improve a solution by searching for a better solution in a neighborhood of the current solution, that is defined in a problem specific way. For example, for the classic Traveling Salesman problem, the neighborhood of a tour can be defined as the set of all tours that differ from it in at most k edges, the so-called *k-exchange neighborhood* [33, 42]. For inputs of size n, a naïve brute-force search of the k-exchange neighborhood requires $n^{\mathcal{O}(k)}$ time; this is infeasible in practical terms even for relatively small values of k. But is it possible to do better? Is it possible to solve local search problems in, say time $\tau(k) \cdot n^{\mathcal{O}(1)}$, for some function τ of k only? It has been generally assumed, perhaps because of the typical algorithmic structure of local search algorithms: "Look at *all* solutions in the neighborhood of the current solution ...", that finding an improved solution (if there is one) in a k-exchange neighborhood necessarily requires brute-force search of the neighborhood; therefore, verifying optimality in a k-exchange neighbourhood requires $\Omega(n^k)$ time (see, e.g. [1] p. 339 or [29] p. 680).

An appropriate tool to answer these questions is parameterized complexity. In the parameterized framework, for decision problems with input size n and a parameter k, the goal is to design algorithms with runtime $\tau(k) \cdot n^{\mathcal{O}(1)}$, where τ is a function of k alone. Problems having such algorithms are said to be *fixed-parameter tractable* (FPT). There is also a theory of hardness to identify parameterized problems that are probably not amenable to FPT algorithms, based on a complexity hypothesis similar to P$\neq$NP. For an introduction to the field and more recent developments, see the books [12, 15, 39].

By making use of developments from parameterized complexity, it appeared that the complexity of local search is much more interesting and involved than it was assumed to be for a long time. While many k-exchange neighbourhood search problems, like determining whether there is an improved solution in the k-exchange neighborhood for TSP, are $W[1]$-hard parameterized by k [35], it appears that for some problems FPT algorithms exist. For example, Khuller, Bhatia, and Pless [28] investigated the NP-hard problem of finding a feedback edge set that is incident to the minimum number of vertices. One of the results obtained in [28] is that checking whether it is possible to improve a solution by replacing at most k edges can be done in time $\mathcal{O}(n^2 + n\tau(k))$, i.e., it is FPT parameterized by k. Similar results were obtained for many problems on planar graphs [14] and for the feedback arc set problem in tournaments [16]. Complexity of k-exchange problems for Boolean CSP and SAT was studied in [31, 47]. The parameterized complexity of local search of different problems was investigated in [20, 24, 36, 37, 41]. However, most of these results exhibit the hardness of local search, and, as it was mentioned by Marx in [34], in most cases, the fixed-parameter tractability results are somewhat unexpected.

Our result. There are various neighborhoods considered in the literature for different problems. Since for the Minimum Fill-in problem the solution is determined by an edge subset, the following definition of the neighbourhood comes naturally. For a pair of graphs $G = (V, E)$ and $G' = (V, E')$ on the same vertex set V, let $H(G, G')$ be $|E \triangle E'|$, i.e. the Hamming distance between the edge sets of E and E'. We say that G is a *neighbor of G' with respect to k-exchange neighbourhood* k-**ExN** if $H(G, G') \leq k$. Let $\mathcal{N}_k^{en}(G)$ be the set of neighbours of G with respect to k-**ExN**. We define the following variant of local search.

k-Local Search Fill-in (k-LS-FI) **Parameter:** k
Input: A graph $G = (V, E)$, its triangulation $H = (V, E \cup F)$ and an integer $k > 0$.
Question: Decide whether there is a triangulation of $H' = (V, E \cup F')$ of G such that $H' \in \mathcal{N}_k^{en}(H)$ and $|F'| < |F|$.

The main result of the paper is the following theorem.

▶ **Theorem 1.** *k-LS-FI is FPT.*

The theorem is proved in several steps. Let a graph $G = (V, E)$ and its triangulation $H = (V, E \cup F)$ be an input of k-LS-FI. We refer to a graph $H' \in \mathcal{N}_k^{en}(H)$ with $|F'| < |F|$ as to a solution of k-LS-FI. We start from a simple criteria to identify edges of F that should be in every solution of k-LS-FI (Lemma 15). Based on this criteria, we can show that if a solution exist, i.e. G and H is a YES-instance of k-LS-FI, then there is a solution $H' = (V, E \cup F')$ such that the edges of $F \triangle F'$ "affect" at most $k(k + 1)$ maximal cliques of H. This is done in Lemma 17. The next step is to identify the cliques of H that can be affected by the solution. While the number of sets of at most $k(k + 1)$ maximal cliques in a chordal graph can be $n^{\mathcal{O}(k^2)}$, we design a procedure to generate at most $n2^{\mathcal{O}(k^5)}$ sets of maximal cliques of H, each set containing at most $k(k+1)$ cliques, and at least one of these sets is a set of cliques affected by the solution. The procedure generating sets of affected maximal cliques is given in Lemma 20, and this is the most technical part of our algorithm. What remains to show is that for a given set of maximal cliques, we can construct in FPT time a solution of k-LS-FI affecting only these cliques.

2 Preliminaries

We denote by $G = (V, E)$ a finite, undirected and simple graph with vertex set $V(G) = V$ and edge set $E(G) = E$. We also use n to denote the number of vertices in G. For a non-empty subset $W \subseteq V$, the subgraph of G induced by W is denoted by $G[W]$. We also use $G \setminus W$ to denote $G[V \setminus W]$. The *open neighborhood* of a vertex v is $N(v) = \{u \in V : uv \in E\}$ and the *closed neighborhood* is $N[v] = N(v) \cup \{v\}$. For a vertex set $W \subseteq V$, we put $N(W) = \bigcup_{v \in W} N(v) \setminus W$ and $N[W] = N(W) \cup W$. We say that an edge uv of graph G is *contained* in set $X \subseteq V$, if $u, v \in X$. We refer to Diestel's book [10] for basic definitions from graph theory.

A *walk* is a sequence of vertices $v_1 v_2 \ldots v_\ell$ where $v_i v_{i+1} \in E(G)$ for $1 \leq i < \ell$. The walk is called a *path* if the vertices are distinct, and the path is called a *cycle* if $v_1 v_\ell \in E$. The path is refereed to as *induced* if $G[\{v_1 v_2 \ldots v_\ell\}]$ only contains the edges of the walk, and the walk is an induced cycle if $v_1 v_\ell$ is the only non-walk edge. A *chord* of a cycle is an edge between two non-consecutive vertices of the cycle, thus induced cycles are chordless.

Chordal graphs and minimal triangulations. *Chordal* or *triangulated* graphs form the class of graphs containing no induced cycles of length more than three. In other words, every cycle of length at least four in a chordal graph contains a chord.

A *triangulation* of graph $G = (V, E)$ is a chordal supergraph $H = (V, E \cup F)$ of G. For a triangulation $H = (V, E \cup F)$, we refer to edge set F as the set of *fill* edges. A triangulation

H of graph G is called *minimal* if $H' = (V, E \cup F')$ is not chordal for any edge set $F' \subset F$ and H is a *minimum* triangulation if $H' = (V, E \cup F')$ is not chordal for every edge set F' such that $|F'| < |F|$. If H is a minimum triangulation of G, then $|F|$ is the minimum fill-in for G.

By the following result, for every non-minimal triangulation, there is a fill edge whose removal leaves a chordal graph. It also implies that a greedy approach—as far as there is an edge e which removal does not create an induced 4-cycle, delete e—can be used to obtain a minimal triangulation from a non-minimal triangulation.

▶ Proposition 2 ([46]). A triangulation $H = (V, E \cup F)$ of $G = (V, E)$ is minimal if and only if for every edge $uv \in F$, deleting uv from H results in a graph with a chordless cycle of length four.

If chordal graph $H = (V, E \cup F)$ is not a minimal triangulation of $G = (V, E)$, then by Proposition 2, we can always find an edge $uv \in F$ such that $H \setminus uv$ is chordal. It is possible to check in linear time if the input graph is chordal [48], and thus in time $\mathcal{O}(|F|(|V| + |E \cup F|))$ one can check if H is a minimal triangulation of G. Hence if the input graph H is not a minimal triangulation of G, we can solve k-LS-FI in time $\mathcal{O}(|F|(|V| + |E \cup F|))$. In the remaining part of the paper, we assume that H is a *minimal* triangulation of G.

Even though we can always argue that the input chordal graph H is a minimal triangulation of G, we can not ensure that every solution H' of the k-LS-FI problem is a minimal triangulation of G, see Fig. 1.

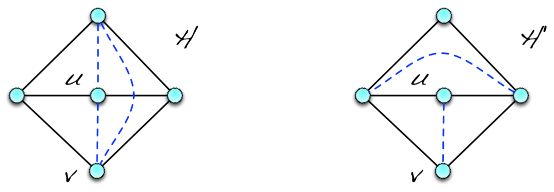


Figure 1 In the instance of k-LS-FI, $k = 3$, the original edges of $G = (V, E)$ are solid lines, and the fill edges F are dashed lines. Graph $H = (V, E \cup F)$ is one of two minimal triangulations of $G = (V, E)$ and H' on the right side is a solution of the provided 3-LS-FI instance. However, graph H' is not a minimal triangulation of G as $H' \setminus uv$ is chordal and to obtain a minimal triangulation $H' \setminus uv$ from H one has to swap four edges.

On the other hand, the following lemma ensures that we can seek a solution which is a minimal triangulation of some supergraph of G and a subgraph of H. Because of the following lemma, we will be able to use nice properties of minimal triangulations in search of a better solution.

▶ **Lemma 3.** *Let $H' = (V, E \cup F')$ be a solution of k-LS-FI with instance graphs $G = (V, E)$ and $H = (V, E \cup F)$. Then there is a solution $H'' = (V, E \cup F'')$ such that H'' is a minimal triangulation of $H_r = (V, E \cup (F \cap F'))$.*

Proof. Graph H' is chordal and is a supergraph of H_r, hence it is a triangulation of H_r. If H' was not a minimal triangulation of H_r, then removal of a non-empty subset of edges $S \subseteq F' \setminus (F \cap F')$ from H' results in a minimal triangulation $H'' = (V, E \cup F'')$ of H_r. Since $|F \triangle F''| < |F \triangle F'| \leq k$, we have that H'' is the required minimal triangulation. ◀

Vertex eliminations. A vertex of a graph is *simplicial*, if its neighbourhood is a clique. By the classical result of Fulkerson and Gross [18], a graph H is chordal if and only if it admits a *perfect elimination ordering*, i.e. vertex ordering $\pi\colon \{1, 2, \ldots, n\} \to V(G)$ such that for every $i \in \{1, 2, \ldots, n\}$, vertex $\pi(i)$ is simplicial in graph $H[\{\pi(i), \ldots, \pi(v)\}]$. Given a vertex ordering π of a graph G, we can construct a triangulation H of G such that π is a perfect elimination ordering for H. Triangulation H is obtained by the following *vertex elimination procedure* (also known as Elimination Game) [18, 44]. A vertex elimination procedure takes as an input a vertex ordering π of graph G and outputs a chordal graph $H = H_n$. We put $H_0 = G$ and define H_i to be the graph obtained from H_{i-1} by completing all neighbours v of $\pi(i)$ in H_{i-1} with $\pi^{-1}(v) > i$ into a clique. An elimination ordering π is called *minimal* if the corresponding vertex elimination procedure outputs a minimal triangulation of G.

▶ Proposition 4 ([40]). Graph H is a minimal triangulation of G if and only if there exists a minimal elimination ordering π of G such that the corresponding procedure outputs H.

We will also need the following description of the fill edges introduced by vertex eleminations.

▶ Proposition 5 ([46]). Let H be the chordal graph produced by vertex elimination of graph G according to ordering π. Then $uv \notin E(G)$ is a fill edge of H if and only if there exists a path $P = uw_1w_2 \ldots w_\ell v$ such that $\pi^{-1}(w_i) < \min(\pi^{-1}(u), \pi^{-1}(v))$ for each $1 \leq i \leq \ell$.

By the arguments used by Fulkerson and Gross [18] in combination with Ohtsuki et al. [40], we can reach the following conclusion.

▶ Proposition 6 (Folklore). Let H be a minimal triangulation of G and let $X \subseteq V$ be a clique of G. Then there exists a minimal elimination ordering π of G resulting in H such that vertices of X are the last vertices in π.

Minimal separators. Let u and v be two non-adjacent vertices of a graph G. A set of vertices $S \subseteq V$ is an *u, v-separator* if u and v are in different connected components of the graph $G[V \setminus S]$. We say that S is a *minimal u, v-separator* of G if no proper subset of S is an u, v-separator and that S is a *minimal separator* of G if there are two vertices u and v such that S is a minimal u, v-separator. Notice that a minimal separator can be contained in another one. If a minimal separator is a clique, we refer to it as to a *clique minimal separator*. In a chordal graph, each minimal separator is a clique minimal separator. Also a chordal graph on n vertices contains at most n maximal cliques and $n-1$ minimal separators [11].

A connected component C of $G \setminus S$ is a *full component* associated with S if $N(C) = S$. The following proposition is an exercise in [23].

▶ Proposition 7 (Folklore). A set S of vertices of G is a minimal a, b-separator if and only if a and b are in different full components associated with S. In particular, S is a minimal separator if and only if there are at least two distinct full components associated with S.

Two separators S and S' are *crossing* if S is a u, v-separator for a pair of vertices $u, v \in S'$, and S' is a u', v'-separator for some $u', v' \in S$.

▶ Proposition 8 ([43]). Graph H is a minimal triangulation of G if and only if H can be obtained from G by completing a maximal set of pairwise non-crossing minimal separators into cliques.

▶ Proposition 9 ([30, 43]). Let H be a minimal triangulation of G. Then every minimal separator in H is a minimal separator in G.

For a minimal triangulation $H = (V, E \cup F)$ of G, propositions 8 and 9 imply that for every edge $uv \in F$ there exists a minimal separator S of both G and H such that $u, v \in S$. We also use the following result.

▶ Proposition 10 ([30, 43]). Let H be a minimal triangulation of G. Then every full component C associated with a minimal separator S in H is also a full component associated with the minimal separator S in G.

The following proposition is folklore; see, e.g., [5].

▶ Proposition 11 ([5]). Let $H = (V, E \cup F)$ be a minimal triangulation of $G = (V, E)$ and let $v_1 v_2 \dots v_\ell$ be a chordless cycle in G. Then either $v_2 v_\ell \in F$, or $v_1 v_i \in F$ for some $2 < i < \ell$.

We also use the following result.

▶ Proposition 12 ([30]). Let S be a minimal separator of G, and let G_S be the graph obtained from G by completing S into a clique. Let $C_1, C_2, \dots, C_r$ be the connected components of $G \setminus S$. Then graph H obtained from G_S by adding a set of fill edges F is a minimal triangulation of G if and only if $F = \bigcup_{i=1}^{r} F_i$, where F_i is the set of fill edges in a minimal triangulation of $G_S[N[C_i]]$.

Clique trees and tree decompositions. A *tree decomposition* TD_G of a graph $G = (V, E)$ is a pair (T, χ) consisting of a set χ of vertex subsets of V and the elements of χ are mapped bijectively onto the nodes of T such that $V = \bigcup_{X \in \chi} X$; for every $uv \in E$, $u, v \in X$ for some $X \in \chi$;, and for every vertex $v \in V$ the set of elements of χ containing v induces a subtree of T. Tree decompositions are strongly related to chordal graphs due to the following proposition.

▶ Proposition 13 ([6, 21, 49]). Graph G is chordal if and only if there exists a tree decomposition (T, χ) of G such that every $X \in \chi$ is a maximal clique in G.

Such a tree decomposition is referred to as a *clique tree* of G. It is well known that a clique tree of a chordal graph on n vertices and m edges can be constructed in $O(n + m)$ time [4]. Vertices of the clique tree will be refereed to as *nodes* in order to distinguish them from the vertices of the graph. We also need the following result relating edges of a clique tree of a chordal graph and its minimal separators.

▶ Proposition 14 ([6, 26]). Let (T, χ) be a clique tree of a chordal graph G. Then S is a minimal separator of G if and only if $S = X_i \cap X_j$ for some edge $X_i X_j \in E(T)$.

For ease of notation we will often refer to the edge set of an edge $X_i X_j$ in the clique tree T as the vertex set $S = X_i \cap X_j$.

Parameterized complexity. A parameterized problem Π is a subset of $\Gamma^* \times \mathbb{N}$ for some finite alphabet Γ. An instance of a parameterized problem consists of (x, k), where k is called the parameter. A central notion in parameterized complexity is *fixed-parameter tractability (FPT)* which means, for a given instance (x, k), solvability in time $f(k) \cdot p(|x|)$, where f is an arbitrary function of k, and p is a polynomial in the input size. We refer to the book of Downey and Fellows [12] for further reading on parameterized complexity.

3 Local search

Immovable edges. Let $G = (V, E)$ be a graph and $H = (V, E \cup F)$ be a minimal triangulation of G. We say that an edge $e \in F$ is *immovable*, if for every triangulation $H' = (V, E \cup F') \in \mathcal{N}_k^{en}(H)$ we have $e \in F'$. In other words, each triangulation H' from the k-neighbourhood of H should contain e. We need a sequence of results providing conditions enforcing edges to be immovable. Due to space limitations the proof of the following lemma has been removed.

▶ **Lemma 15.** *Let S be a minimal separator of a minimal triangulation $H = (V, E \cup F)$ of an n-vertex graph $G = (V, E)$, let C be a full component associated with S in H, and let $u, v \in S$ such that $uv \in F$ and $|(N_H(u) \cap N_H(v)) \setminus (C \cup S)| > k$. Then uv is an immovable edge. Moreover, one can check in time $\mathcal{O}(n^3)$ if an edge $uv \in F$ satisfies the above conditions and thus is immovable.*

Lemma 15 yields the following lemma.

▶ **Lemma 16.** *Let $H = (V, E \cup F)$ be a minimal triangulation of graph $G = (V, E)$ and let X_1 and X_2 be maximal cliques of H such that $|X_2 \setminus X_1| > k$. Then every edge of F contained in $X_1 \cap X_2$ is immovable.*

Proof. Let T be a clique tree of H and remember that each node of T represent a maximal clique of H. Let X' be the neighbour of X_1 on the unique path from X_1 to X_2 in T. By Proposition 14, $S = X_1 \cap X'$ is a minimal separator in H. Let us remark, that $S \supseteq X_1 \cap X_2$. Let C be the full component of $H \setminus S$ associated with S containing $X_1 \setminus S$. For every edge $uv \in F$ such that $u, v \in X_1 \cap X_2$, we have that $u, v \in S$, and because X_2 is a clique, we have that every vertex from $X_2 \setminus (S \cup C)$ is adjacent to both u and v. Finally, $|(N_H(u) \cap N_H(v)) \setminus (C \cup S)| \geq |X_2 \setminus (S \cup C)| = |X_2 \setminus X_1| > k$. Now the proof of the lemma follows by Lemma 15. ◀

▶ **Lemma 17.** *Let $H = (V, E \cup F)$ and $H' = (V, E \cup F') \in \mathcal{N}_k^{en}(H)$ be minimal triangulations of G. Then H has at most $k(k+1)$ maximal cliques containing both endpoints of some edge from $F \setminus F'$.*

Proof. We start the proof with the following claim.
Claim: Every edge $uv \in F$ contained in more than $k+1$ maximal cliques of H is immovable.

Proof of the claim: In the clique tree T of H, the nodes corresponding to these maximal cliques containing uv induce a subtree T_{uv}. Let $X_1, X_2, \ldots, X_\ell$, $\ell \geq k+2$ be the maximal cliques corresponding to nodes of T_{uv} and let them be numbered such that X_1 is a leaf of T_{uv} and X_2 is the parent of X_1 in T_{uv}. Then $S = X_1 \cap X_2$ is a minimal separator containing u and v. Because X_1 is a maximal clique, there is $x_1 \in X_1 \setminus S$ such that the connected component of $H \setminus S$ containing x_1 is a full component C associated with S. Remove X_1 and repeat on the cliques $X_2, \ldots, X_\ell$, $\ell \geq k+2$ that still induces a sub-tree of T. Hence, there are at least $k+1$ vertices that are adjacent to both u and v and not contained in $C \cup S$. By Lemma 15, edge uv is immovable. This concludes the proof of the claim.

We proceed with the proof of the lemma. Because $H' \in \mathcal{N}_k^{en}(H)$, we have that none of the edges from $F \setminus F'$ is immovable. By the claim above, each such edge $e \in F \setminus F'$ is contained in at most $k+1$ maximal cliques of H. Since $|F \setminus F'| \leq k$, the lemma follows. ◀

Generating affected cliques. The following lemmata allow us to reduce the search space. As a result, we are able to generate at most $2^{\mathcal{O}(k^5)}$ sets of cliques, each set of size at most $k(k+1)$, such that if there is a solution to the problem, then there is also a solution that swaps edges only between vertices in one of the sets of maximal cliques. Due to space limitations the proof of the following lemma has been removed.

▶ **Lemma 18.** *Let $H = (V, E \cup F)$ be a minimal triangulation of G and let $H' = (V, E \cup F')$ be a solution of k-LS-FI. If H has a minimal separator S containing no edges of $F \setminus F'$, then there is a connected component C of $H \setminus S$ and a solution $H'' = (V, E \cup F'')$ of k-LS-FI such that every edge from $(F'' \setminus F) \cup (F \setminus F'')$ is contained in $N_H[C]$.*

By Lemma 18, we obtain the following lemma.

▶ **Lemma 19.** *Let $H = (V, E \cup F)$ be a minimal triangulation of G and let T be a clique tree of H. If there is a triangulation $H' = (V, E \cup F') \in \mathcal{N}_k^{en}(H)$ with $|F'| < |F|$, then there is a triangulation $H'' = (V, E \cup F'') \in \mathcal{N}_k^{en}(H)$ with $|F''| < |F|$ such that the maximal cliques of H containing edges from $F \setminus F''$ induce a subtree of T.*

Proof. As long as the maximal cliques of H containing edges from $F \setminus F'$ do not induce a subtree of the clique tree T of H, there exists a minimal separator S of H such that no edges of $F \setminus F'$ are contained in S and there exist endpoints of edges in $F \setminus F'$ that are separated by S. By Lemma 18, we can obtain a new solution $H'' = (V, E \cup F'')$ where $|F \setminus F''| < |F \setminus F'|$ and all endpoints of the edges in $F \setminus F''$ are contained in the same connected component of $H[V \setminus S]$. Repeat this until the maximal cliques of H containing edges from $F \setminus F''$ induce a subtree of the clique tree of H. ◀

By Lemma 19, if there is a solution of k-LS-FI, then there is also a solution where the maximal cliques of H containing edges deleted from H form a subtree of the clique tree of H. The next lemma gives an algorithm that in FPT time outputs at least one of such subtrees. Due to space limitations the proof of the following lemma has been removed.

▶ **Lemma 20.** *Let $H = (V, E \cup F)$ be a minimal triangulation of an n-vertex graph G. There is an algorithm that in time $\mathcal{O}(2^{\mathcal{O}(k^5)} n^2 + |F| \cdot n^3)$ outputs sets $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_t$, $t \leq n2^{\mathcal{O}(k^5)}$, of maximal cliques of H such that*

- *if there is a solution to k-LS-FI, then there exists a solution $H' = (V, E \cup F')$, $|F'| < |F|$, of k-LS-FI and a set $\mathcal{X} \in \{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_t\}$ such that the cliques of $\mathcal{X}$ induces a subtree of clique tree T of H and are exactly the cliques containing edges of $F \setminus F'$.*

Final step. By Lemma 20, we are able to compute at least one of the subtrees of the maximal clique tree of H that consists of maximal cliques containing edges of H that will be removed in a better triangulation. We are ready to prove the main result about k-LS-FI, Theorem 1.

Proof of Theorem 1. To prove the theorem, we show that given a minimal triangulation $H = (V, E \cup F)$ of an n-vertex graph $G = (V, E)$, searching for a better triangulation in the k-exchange neighbourhood of H can be performed in time $\mathcal{O}(2^{\mathcal{O}(k^5)} n^4 + |F| \cdot n^3)$.

Let T be a clique tree of H. We use Lemma 15 to mark some edges of F as immovable. We also mark minimal separators of H containing only immovable edges from F as immovable. We use the algorithm from Lemma 20 to output at most $n2^{\mathcal{O}(k^5)}$ sets $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_t$ of maximal cliques of $H = (V, E \cup F)$ such that

- If pair G and H is a YES-instance of k-LS-FI, then there is a triangulation of G, $H' = (V, E \cup F') \in \mathcal{N}_k^{en}(H)$ with $|F'| < |F|$ such that at least one $\mathcal{X}_i$ consists of all cliques containing both endpoints for some edge of $F \setminus F'$;
- Each set $\mathcal{X}_i$ contains at most $k(k + 1)$ maximal cliques of H;
- For every set $\mathcal{X}_i$, no two maximal cliques from $\mathcal{X}_i$ can be separated by an immovable separator.

For set $\mathcal{X}_i$, $1 \leq i \leq t$, we define H_i to be the induced subgraph of H induced by the vertices of cliques from $\mathcal{X}_i$. Let S be a minimal separator of H_i. By Lemma 16, for every intersecting maximal cliques $X_1, X_2 \in \mathcal{X}_i$, we have that $|X_1 \setminus X_2| < k$. Hence, graph H_i contains at most $|S| + k^2(k+1)$ vertices as the hole sub-tree can be reduced to two maximal cliques by recursively removing leaf cliques and each of them have at most $k - 1$ private vertices. We also define G_i to be the induced subgraph of G induced by the vertices of cliques from $\mathcal{X}_i$. Then G_i also has at most $|S| + k^2(k+1)$ vertices.

Let $\mathcal{C}$ be the set of all maximal cliques of H. By Lemmata 18 and 20, the search of a solution boils down to the search in the k-exchange neighbourhood of H for a better triangulation $H' = (V, E \cup F')$, which satisfies for some i, $1 \leq i \leq t$, the following additional condition: no maximal clique $C \in \mathcal{C} \setminus \mathcal{X}_i$ contains any edges from $F \setminus F'$ and no edge from $F' \setminus F$. The later is trivial as edges of $F' \setminus F$ are not present in H.

Let G'_i be the graph obtained from G_i by adding immovable edges of H_i and all edges of $F \cap E(H_i)$ which are contained in maximal cliques of $\mathcal{C} \setminus \mathcal{X}_i$. We show how to find a better triangulation of G'_i.

By Proposition 4, every minimal triangulation of G'_i corresponds to a minimal elimination ordering of G'_i. In graph G'_i, there are at most $k^2(k+1)$ vertices outside S. Thus in every elimination ordering, there are at most $k^2(k+1)$ vertices preceding the first vertex of S. We try all possible subsets of $V(G'_i) \setminus S$ and their permutations for a possible prefix in this ordering. Thus we try at most $2^{k^2(k+1)}(k^2(k+1))!$ ordered subsets. For every prefix π, we guess also the first vertex $v \in S$ which goes after π. So in total we try at most $n \cdot 2^{k^2(k+1)}(k^2(k+1))!$ ordered subsets. Let Y be the subset of vertices of S which are either adjacent to v or reachable from v through the vertices of the prefix. By Proposition 5, set Y is a clique in any triangulation obtained by an ordering extending π. Let $Z = S \setminus Y$. If $|Z| > k$, then we made a wrong guess on the prefix π because at least $k+1$ edges incident to v have to be deleted, and this prefix cannot produce a triangulation in a k-exchange neighbourhood of H_i.

Hence we assume that $|Z| \leq k$. By eliminating vertices of π and v first it follows by Proposition 6, that there exists a minimal elimination ordering producing the minimum fill such that the vertices of Y are the last vertices in this ordering. Thus there is a minimal elimination ordering producing the minimum fill of the form $\pi v Z Y$. As we already shown, there are at most $2^{k^2(k+1)}(k^2(k+1))!$ ways to select the ordered prefix π, and at most n ways to select $v \in S$. As far as π and v are fixed, there is a unique way to define Y and Z. There are at most $k!$ permutations of Z and any permutation of Y will do the job. Thus in total, there are at most $n \cdot 2^{k^2(k+1)}(k^2(k+1))!k! = 2^{\mathcal{O}(k^3 \log k)}n$ permutations. If $H'_i \in \mathcal{N}_k^{en}(H_i)$, then we output the minimal triangulation $H' = (V, E \cup (F \setminus (E(H_i)) \cup E(H'_i))$. If for every i, $1 \leq i \leq t$, the minimum triangulation $H'_i \notin \mathcal{N}_k^{en}(H_i)$, then we conclude that the pair G and H is a NO-instance of the problem, and thus there is no better triangulation of G in the k-exchange neighbourhood of H.

By Lemma 20, it takes time $\mathcal{O}(2^{\mathcal{O}(k^5)}n^2 + |F| \cdot n^3)$ to generate all subsets of set $\mathcal{X}$ and there are $2^{\mathcal{O}(k^5)}n$ such subsets. For each of the subsets consisting of at most $k(k+1)$ maximal cliques, a separator S can be found in $\mathcal{O}(n^2)$ time. For each set, we try $2^{\mathcal{O}(k^3 \log k)}n$ permutations, resulting in $2^{\mathcal{O}(k^5)}n \cdot 2^{\mathcal{O}(k^3 \log k)}n = 2^{\mathcal{O}(k^5)}n^2$ different elimination orderings. Finally, for each ordering, the corresponding triangulation can be computed in $\mathcal{O}(n^2)$ time. Thus, the total running time is $\mathcal{O}(2^{\mathcal{O}(k^5)}n^4 + |F| \cdot n^3)$. ◀

4 Conclusion and open problems

We have shown fixed-parameter tractability of the variant of search of the k-exchange neighbourhood for Minimum Fill-in. Since only a very few search problems known to be FPT, we find it very interesting to explore what general properties of problems and exchange neighbourhoods are responsible for such phenomena. Another natural question is about the running time of the algorithm. The worst case upper bound on the running time of our algorithm makes the result of the paper mainly of theoretical importance. However, the common story about improvements of FPT algorithms is that with more work and new ideas, these algorithm can be made practical.[1] Very recently, it was shown that the parameterized version of MINIMUM FILL-IN is solvable in subexponential $2^{o(k)}n^{\mathcal{O}(1)}$ time. Can it be that k-LS-FI is solvable in time $\mathcal{O}(2^{o(k)}n^c)$ for some small constant c? Combined with other fill-in reducing heuristics, such an algorithm would be of real practical importance.

References

1 Emile H. L. Aarts and Jan Karel Lenstra. *Local Search in Combinatorial Optimization.* Princeton University Press, 1997.

2 Ajit Agrawal, Philip N. Klein, and R. Ravi. Cutting down on fill using nested dissection: provably good elimination orderings. *Graph Theory and Sparse Matrix Computation*, 56:31–55, 1993.

3 Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513, 1983.

4 J. R. S. Blair and B. W. Peyton. An introduction to chordal graphs and clique trees. In *Graph Theory and Sparse Matrix Computations*, pages 1–30. Springer, 1993. IMA Volumes in Mathematics and its Applications, Vol. 56.

5 Hans Bodlaender, Pinar Heggernes, and Yngve Villanger. Faster parameterized algorithms for minimum fill-in. *Algorithmica*, 61:817–838, 2011.

6 P. Buneman. A characterization of rigid circuit graphs. *Discrete Math.*, 9:205–212, 1974.

7 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Inf. Process. Lett.*, 58(4):171–176, 1996.

8 Fan R. K. Chung and David Mumford. Chordal completions of planar graphs. *J. Comb. Theory, Ser. B*, 62(1):96–106, 1994.

9 Timothy A. Davis. *Direct methods for sparse linear systems*, volume 2 of *Fundamentals of Algorithms.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2006.

10 Reinhard Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics.* Springer-Verlag, Berlin, third edition, 2005.

11 G. A. Dirac. On rigid circuit graphs. *Abh. Math. Sem. Univ. Hamburg*, 25:71–76, 1961.

12 R. G. Downey and M. R. Fellows. *Parameterized complexity.* Springer-Verlag, New York, 1999.

13 Iain S. Duff and Bora Ucar. Combinatorial problems in solving linear systems. In *Combinatorial Scientific Computing*, number 09061 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2009. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

14 Michael R. Fellows, Fedor V. Fomin, Daniel Lokshtanov, Frances A. Rosamond, Saket Saurabh, and Yngve Villanger. Local search: Is brute-force avoidable? *J. Comput. Syst. Sci.*, 78(3):707–719, 2012.

15 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory.* Springer-Verlag, Berlin, 2006.

16 Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Fast local search algorithm for weighted feedback arc set in tournaments. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010)*, pages 65–70. AAAI Press, 2010.

[1] Parameterized complexity community wiki contains different examples of running time improvements at http://fpt.wikidot.com/fpt-races

17 Fedor V. Fomin and Yngve Vilanger. Subexponential parameterized algorithm for minimum fill-in. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 1737–1746. SIAM, 2012.

18 D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pacific J. Math.*, 15:835–855, 1965.

19 Michael R. Garey and David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

20 Serge Gaspers, Eun Jung Kim, Sebastian Ordyniak, Saket Saurabh, and Stefan Szeider. Don't be strict in local search! In *Proceedings of the 26th AAAI Conference (AAAI-12)*, page to appear. AAAI Press, 2012.

21 F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *J. Combin. Theory Ser. B*, 16:47–56, 1974.

22 D. Geman. Random fields and inverse problems in imaging. In *École d'été de Probabilités de Saint-Flour XVIII—1988*, volume 1427 of *Lecture Notes in Math.*, pages 113–193. Springer, Berlin, 1990.

23 M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, New York, 1980.

24 Jiong Guo, Sepp Hartung, Rolf Niedermeier, and Ondrej Suchý. The parameterized complexity of local search for TSP, more refined. In *Proceedings of the 22nd International Symposium on Algorithms and Computation (ISAAC 2011)*, volume 7074 of *LNCS*, pages 614–623. Springer, 2011.

25 Pinar Heggernes. Minimal triangulations of graphs: A survey. *Discrete Mathematics*, 306(3):297–317, 2006.

26 Chin-Wen Ho and R.C.T. Lee. Counting clique trees and computing perfect elimination schemes in parallel. *Information Processing Letters*, 31(2):61 – 68, 1989.

27 Haim Kaplan, Ron Shamir, and Robert E. Tarjan. Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM J. Comput.*, 28:1906–1922, May 1999.

28 Samir Khuller, Randeep Bhatia, and Robert Pless. On local search and placement of meters in networks. *SIAM J. Comput.*, 32(2):470–487, 2003.

29 Jon Kleinberg and Eva Tardos. *Algorithm Design.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

30 Ton Kloks, Dieter Kratsch, and Jeremy Spinrad. On treewidth and minimum fill-in of asteroidal triple-free graphs. *Theor. Comput. Sci.*, 175(2):309–335, 1997.

31 Andrei Krokhin and Dániel Marx. On the hardness of losing weight. *ACM Trans. Algorithms*, 8(2):19:1–19:18, 2012.

32 S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *J. Royal Statist. Soc. B*, 50(2):157–224, 1988.

33 S. Lin and B. W. Kernighan. An effective heuristic algorithm for traveling-salesman problem. *Operations Research*, 21:498–516, 1973.

34 Dániel Marx. Local search. *Parameterized Complexity Newsletter*, 3:7–8, 2008.

35 Dániel Marx. Searching the k-change neighborhood for TSP is W[1]-hard. *Oper. Res. Lett.*, 36(1):31–36, 2008.

36 Dániel Marx and Ildikó Schlotter. Parameterized complexity and local search approaches for the stable marriage problem with ties. *Algorithmica*, 58(1):170–187, 2010.

37 Dániel Marx and Ildikó Schlotter. Stable assignment with couples: Parameterized complexity and local search. *Discrete Optimization*, 8(1):25–40, 2011.

38 Assaf Natanzon, Ron Shamir, and Roded Sharan. A polynomial approximation algorithm for the minimum fill-in problem. *SIAM J. Comput.*, 30:1067–1079, October 2000.

39 Rolf Niedermeier. *Invitation to fixed-parameter algorithms.* Oxford University Press, 2006.

40 T. Ohtsuki, L. K. Cheung, and T. Fujisawa. Minimal triangulation of a graph and optimal pivoting ordering in a sparse matrix. *J. Math. Anal. Appl.*, 54:622–633, 1976.

41 Sebastian Ordyniak and Stefan Szeider. Algorithms and complexity results for exact bayesian structure learning. In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI 2010)*, pages 401–408. AUAI Press, 2010.

42 Christos H. Papadimitriou and Kenneth Steiglitz. On the complexity of local search for the traveling salesman problem. *SIAM J. Comput.*, 6(1):76–83, 1977.

43 Andreas Parra and Petra Scheffler. Characterizations and algorithmic applications of chordal graph embeddings. *Discrete Applied Mathematics*, 79(1-3):171–188, 1997.

44 S. Parter. The use of linear graphs in Gauss elimination. *SIAM Review*, 3:119–130, 1961.

45 D. J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In R. C. Read, editor, *Graph Theory and Computing*, pages 183–217. Academic Press, New York, 1972.

46 D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5:266–283, 1976.

47 Stefan Szeider. The parameterized complexity of k-flip local search for SAT and MAX SAT. *Discrete Optimization*, 8(1):139–145, 2011.

48 Robert Endre Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3):566–579, 1984.

49 J. R. Walter. *Representations of rigid cycle graphs.* PhD thesis, Wayne State University, 1972.

50 S. Wong, D. Wu, and C. Butz. Triangulation of Bayesian networks: A relational database perspective. In *Rough Sets and Current Trends in Computing*, volume 2475 of *LNCS*, pages 950–950. Springer, 2002.

51 M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Meth.*, 2:77–79, 1981.

Probably Optimal Graph Motifs*

Andreas Björklund[1], Petteri Kaski[2], and Łukasz Kowalik[3]

1 **Department of Computer Science, Lund University, Sweden**
andreas.bjorklund@yahoo.se

2 **Helsinki Institute for Information Technology HIIT & Department of Information and Computer Science, Aalto University, Finland**
petteri.kaski@aalto.fi

3 **Institute of Informatics, University of Warsaw, Poland**
kowalik@mimuw.edu.pl

Abstract

We show an $O^*(2^k)$-time polynomial space algorithm for the k-sized Graph Motif problem. We also introduce a new optimization variant of the problem, called Closest Graph Motif and solve it within the same time bound. The Closest Graph Motif problem encompasses several previously studied optimization variants, like Maximum Graph Motif, Min-Substitute, and Min-Add.

Moreover, we provide a piece of evidence that our result might be essentially tight: the existence of an $O^*((2-\epsilon)^k)$-time algorithm for the Graph Motif problem implies an $O((2-\epsilon')^n)$-time algorithm for Set Cover.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases graph motif, FPT algorithm

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.20

1 Introduction

The Graph Motif problem is defined as follows. We are given an undirected connected graph $G = (V, E)$, a vertex coloring $c : V \to C$, and a multiset M consisting of colors in the set C. The goal is to find a subset $S \subseteq V$ such that the induced subgraph $G[S]$ is connected, and the *multiset* $c(S)$ of colors of the vertices of S is equal to M. To avoid confusion, let us stress that the input function c is arbitrary and it does not need to be a proper vertex coloring. Let $k = |S|$ denote the size of the solution (which is $|M|$ in the case of Graph Motif but may differ from $|M|$ in variants of the problem also considered in this paper).

Graph Motif was introduced by Lacroix et al. [17] and motivated by applications in bioinformatics, specifically in metabolic network analysis. It is known to be NP-hard even when the given graph is a tree of maximum degree 3 and the motif is a set [11]. However, in practice the size of M is expected to be small, what motivates the research on so-called FPT algorithms parameterized by k, that is, algorithms with running times bounded from above by a function $f(k)$ times a function polynomial in the input size, which is commonly abbreviated by $O^*(f(k))$. Indeed, Fellows et al. [10] discovered that such an algorithm exists, which was followed by a rapid series of improvements to $f(k)$ (see Table 1).

* This research was supported in part by the Academy of Finland, Grants 252083 and 256287 (P.K.), and by the National Science Centre of Poland, Grant N206 567140 (Ł.K.).

Table 1 The progress on FPT algorithms for the GRAPH MOTIF problem

Paper	Running time	Approach
Fellows et al. [10]	$O^*(87^k)$, implicit	Color-coding
Betzler et al. [1]	$O^*(4.32^k)$	Color-coding
Guillemot and Sikora [12]	$O^*(4^k)$	Multilinear detection
Koutis [15]	$O^*(2.54^k)$	Constrained multilinear detection
this work	$O^*(2^k)$	Constrained multilinear detection

The two most recent results, namely the $O^*(4^k)$ algorithm of Guillemot and Sikora [12] and the $O^*(2.54^k)$-time algorithm of Koutis [15] apply the so-called multilinear detection technique. This technique was introduced by Koutis [13] and further developed by Williams [20] and Koutis and Williams [16] to solve subgraph containment problems. Inspired by the GRAPH MOTIF application and the reduction to multilinear detection [12], Koutis [15] introduced a tailored variant of multilinear detection called constrained multilinear detection (abbreviated k-CMLD) to make further progress on the GRAPH MOTIF problem. In the k-CMLD problem, we are given a multivariate polynomial represented as an arithmetic circuit, and are asked determine whether the polynomial has a multilinear monomial of degree k with an odd coefficient, with the further constraint that the polynomial indeterminates have colors associated to them, and the monomial must not exceed a prescribed number of occurrences of each color. At a Dagstuhl seminar in 2010, Koutis [14] posed the open problem of devising an $O^*(2^k)$-time algorithm for k-CMLD. His recent paper [15] provides an $O^*(2.54^k)$-time algorithm for the problem where the worst case bound results from a budget of at most three occurrences of every color.

In this paper we indirectly show an $O^*(2^k)$-time polynomial space algorithm for k-CMLD, thereby answering Koutis's open problem in the affirmative. Since the main application of k-CMLD is the GRAPH MOTIF problem and its variants, we here present the result directly in terms of graph motifs. In the full version of this paper we will give a self-contained proof for the general k-CMLD (see also [4]). Our approach is inspired by Koutis's beautiful idea of assigning random subspaces of dimension equal to the prescribed multiplicities of the colors. Koutis used group algebras $\mathbb{F}_2[\mathbb{Z}_2^k]$ for his construction, whereas ours appear to require an extension to $\mathbb{F}_{2^\beta}[\mathbb{Z}_2^k]$ for $\beta = \Omega(\log k)$. Rather than proving the result in terms of a group algebra as Koutis suggests, we provide a construction using inclusion–exclusion over labelled indeterminates. As in [3], a paper using the technique co- authored by a subset of the present authors, we find it more convenient to work in this alternative setting.

A further contribution of the present work is to develop a generalization of GRAPH MOTIF called CLOSEST GRAPH MOTIF. In particular, we introduce a notion of edit distance between two multisets, and the objective is to find the subset $S \subseteq V$ such that $G[S]$ is connected and the edit distance from M to $c(S)$ is minimized (see Section 3 for a precise definition). In the literature there are some more optimization variants of GRAPH MOTIF and our CLOSEST GRAPH MOTIF generalizes three of them: MAXIMUM GRAPH MOTIF (see Section 2), MIN-ADD, and MIN-SUBSTITUTE (see Section 3). The previous fastest algorithms for these problems are due to Koutis [15]; he shows $O^*(2.54^k)$-time algorithms for MAXIMUM GRAPH MOTIF and MIN-ADD, and an $O^*(5.08^k)$-time algorithm for MIN-SUBSTITUTE. We present an $O^*(2^k)$-time algorithm for the general CLOSEST GRAPH MOTIF.

Similarly to the three previous FPT improvements on GRAPH MOTIF relative to the parameter k, our algorithm is Monte Carlo with one-sided error (with probability at most $1/2$ the algorithm asserts the absence of a solution when in fact there is one; one can get arbitrarily small error probability p by repeating the algorithm $\log_2(1/p)$ times).

In addition to our algorithmic results, we give a piece of evidence that further improvement on the running time is substantially harder. Namely, we show that for any $\epsilon > 0$ the existence of an $O^*((2-\epsilon)^k)$-time algorithm for the Graph Motif problem implies an $O((2-\epsilon')^n)$-time algorithm for the Set Cover problem, for some $\epsilon' > 0$. Thus, instead of trying to improve our algorithm one should rather attack the more generic Set Cover. After decades of research on the problem, an $O((2-\epsilon)^n)$-time algorithm for Set Cover would be a major breakthrough. Indeed, the nonexistence of such an algorithm has already been used as an assumption for proving hardness results [5]. Furthermore, it is conjectured [5] that an $O((2-\epsilon)^n)$-time algorithm for Set Cover contradicts the SETH (Strong Exponential Time Hypothesis, which states that if k-CNF SAT can be solved in $O^*(c_k^n)$ time, then $\lim_{k\to\infty} c_k = 2$). This conjecture is supported by the fact that the number of solutions to Set Cover cannot be computed in $O((2-\epsilon)^n)$ time for any $\epsilon > 0$ unless SETH fails [5]. Another consequence of such a counting algorithm would be the existence of an $O((2-\epsilon')^n)$-time algorithm to compute the permanent of an $n \times n$ integer matrix, see [2].

This paper is organized as follows. In Section 2 we describe our $O^*(2^k)$-time algorithm for Maximum Graph Motif, while in Section 3 we describe how our algorithm works for the more general Closest Graph Motif and how it encompasses earlier named variants. In Section 4 we show the reduction from Set Cover.

2 An $O^*(2^k)$-time algorithm for Maximum Graph Motif

2.1 The general approach

In this section we will study a slight generalization of Graph Motif, namely the Maximum Graph Motif problem parameterized by the solution size k. That is, for a given instance (G, c, M), the task is to decide whether there exists a subset $S \subseteq V$ such that (i) the induced subgraph $G[S]$ is connected, (ii) $|S| = k$, and (iii) the multiset inclusion $c(S) \subseteq M$ holds.

It is immediate that once we have an algorithm for the decision problem that runs in $T(n,k)$ time, we can *find* a solution in $O(nT(n,k))$ time as follows. For every vertex $v \in V$, check whether a solution exists if we remove v frmo G; if not, then put v back to G; in both cases proceed to the next vertex. After iterating over all vertices, we are left with a desired induced subgraph $G[S]$.

To solve the decision problem we use the following approach. We define a multivariate polynomial P that has two key properties: (1) P is not identically zero if and only if there is a solution S, and (2) P can be evaluated fast (that is, in $O^*(2^k)$ time) at any given point. Then an $O^*(2^k)$-time algorithm follows via the DeMillo–Lipton–Schwartz–Zippel Lemma (see Section 2.6). We note that alternatively, instead of the polynomial approach one can use the random weights approach and the Isolation Lemma (see e.g. [6]).

The main difference between the present approach and earlier works that deploy a similar polynomial sieve is that we employ the same "labels" (the universe of k elements whose subsets we use in sieving) to simultaneously accomplish two different tasks: (i) we sieve out all homomorphisms that are not injective, and (ii) we sieve out all multisets that use too many colors when compared with M.

The rest of the section is organized as follows. First we give some preliminaries and notation on branching walks and labellings in Sections 2.2 and 2.3. In Section 2.4 we define the polynomial P and we prove the property (1) as Lemma 1. In Section 2.5 we show property (2) and finally in Section 2.6 we describe the complete algorithm and analyze its failure probability using the DeMillo–Lipton–Schwartz–Zippel Lemma.

2.2 Preliminaries on branching walks

The concept of branching walks was first introduced by Nederlof [18] to sieve for Steiner trees. Here we provide a slightly modified definition of branching walks. Let G be a graph with vertex set $V = V(G)$ and edge set $E = E(G)$. A mapping $h : V(T) \to V(G)$ is a *homomorphism* from a graph T to a graph G if for all $\{u, v\} \in E(T)$ it holds that $\{h(u), h(v)\} \in E(G)$. We adopt the convention of calling the elements of $V(T)$ *nodes* and the elements of $V(G)$ *vertices*.

A *branching walk* G is a pair $W = (T, h)$ where T is an ordered rooted tree with node set $V(T) = \{1, 2, \ldots, |V(T)|\}$ such that every node $v \in V(T)$ coincides with its rank in the preorder traversal of T, and $h : V(T) \to V(G)$ is a homomorphism from T to G. For a vertex $r \in V$, we say that W *starts* from r if $h(1) = r$.

Let $W = (T, h)$ be a branching walk in G. We define $h(T)$ to be the subgraph of G induced by the set of edges $\{\{h(u), h(v)\} : \{u, v\} \in E(T)\}$. We observe that $h(T)$ is not necessarily a tree because h need not be injective. We say that W is *simple* if $h(T)$ is injective.

It will be convenient to assume that $V(G)$ is totally ordered. Towards this end, let us assume that $V = V(G) = \{1, 2, \ldots, n\}$. We say that a branching walk $W = (T, h)$ in G is *properly ordered* if any two sibling nodes $u < v$ in T satisfy $h(u) < h(v)$.

2.3 Labelling and shading

Let (G, c, M) be the input instance of the Maximum Graph Motif problem and let $m : C \to \mathbb{N}$ be the multiplicity function for M. For each color $q \in C$ and $i = 1, 2, \ldots, m(q)$, let us call the formal pair (q, i) the *i-th shade of color q*. In particular, the number of shades for each color matches the multiplicity of the color in M. Let us write $D(q) = \{(q, i) : i = 1, 2, \ldots, m(q)\}$ for the set of all shades of color $q \in C$, and $D = \cup_{q \in C} D(q)$ for the set of all shades of all colors.

A branching walk (T, h) in G may be *labelled* with a function $\ell : V(T) \to \{1, 2, \ldots, k\}$. The three-tuple (T, h, ℓ) is called a *labelled branching walk*, and the function ℓ is a *labelling* of the branching walk.

A branching walk (T, h) in G may also be *shaded* with a function $s : V(T) \to D$. A shading may also be *partial*, that is, of the form $s : U \to D$ for a subset $U \subseteq V(T)$. We say that a (partial) shading $s : U \to D$ of a branching walk (T, h) is *consistent* with the input coloring $c : V(G) \to C$ if for every node $v \in U$ we have $s(v) \in D(c(h(v)))$. For a branching walk $W = (T, h)$ and a subset $U \subseteq V(T)$, denote by $\mathcal{S}_U(W)$ the set of all consistent (partial) shadings $s : U \to D$. Let us abbreviate $\mathcal{S}(W) = \mathcal{S}_{V(T)}(W)$.

2.4 The polynomial P

We use three different types of indeterminates in our polynomials. First, for each edge $\{a, b\} \in E(G)$, introduce two indeterminates $x_{a,b}$ and $x_{b,a}$. Second, for each vertex $a \in V(G)$ and each shade $t \in D(c(a))$, introduce an indeterminate $y_{a,t}$. Third, for each shade $t \in D$ and each label $j \in \{1, 2, \ldots, k\}$, introduce an indeterminate $z_{t,j}$. Let us write $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ for the sequences of all the $x_{a,b}$-type, $y_{a,t}$-type, and $z_{t,j}$-type variables, respectively.

Let $W = (T, h)$ be a branching walk in G, let $s : V(T) \to D$ be a consistent shading of W, and let $\ell : V(T) \to \{1, 2, \ldots, k\}$ be a labelling of W. Associate with the consistently shaded and labelled branching walk (W, s, ℓ) the monomial

$$\mathrm{mon}(W, s, \ell) = \prod_{\substack{\{u,v\} \in E(T) \\ u<v}} x_{h(u),h(v)} \prod_{v \in V(T)} y_{h(v),s(v)} z_{s(v),\ell(v)} \,.$$

Denote by $\mathcal{W}$ the set of all branching walks of $W = (T, h)$ in G that are properly ordered and satisfy $|V(T)| = k$. Let $\beta = \lceil \log k \rceil + 3$ and denote by $\mathbb{F}_{2^\beta}$ the finite field of order 2^β. Define the multivariate polynomial P with coefficients in $\mathbb{F}_{2^\beta}$ by setting

$$P(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \sum_{W=(T,h) \in \mathcal{W}} \sum_{s \in \mathcal{S}(W)} \sum_{\substack{\ell : V(T) \to \{1,2,\ldots,k\} \\ \ell \text{ bijective}}} \mathrm{mon}(W, s, \ell) \,.$$

▶ **Lemma 1.** *We have $P \not\equiv 0$ if and only if there exists a solution $S \subseteq V(G)$.*

Proof. ($\Leftarrow$) Let T_S be a spanning tree of $G[S]$. Transform T_S into a rooted tree, and make T_S an ordered tree so that the children of every vertex listed in tree order form an increasing sequence. If we replace every vertex in T_S with its rank in a preorder traversal, we obtain a properly ordered simple branching walk $W = (T, h)$. Define the shading $s : V(T) \to D$ for each node $v \in V(T)$ by setting

$$s(v) = \big(c(h(v)), |\{w \in S : c(h(w)) = c(h(v)) \text{ and } w \leq v\}|\big) \,.$$

Note that s is well-defined and consistent because S is a solution. Furthermore, observe that s is injective. Finally, choose an arbitrary bijection $\ell : V(T) \to \{1, 2, \ldots, k\}$. We must now have $P \not\equiv 0$ because we can uniquely reconstruct any three-tuple (W, s, ℓ) with a simple $W = (T, h) \in \mathcal{W}$, an injective $s \in \mathcal{S}(W)$, and a bijective $\ell : V(T) \to \{1, 2, \ldots, k\}$ from its monomial representation $\mathrm{mon}(W, s, \ell)$ – indeed, first recover $W = (T, h)$ from the $x_{a,b}$-type indeterminates using the fact W is simple, properly ordered, and starts from the unique vertex r such that $\mathrm{mon}(W, s, \ell)$ contains variables of the form $x_{r,v}$ but does not contain variables of the form $x_{v,r}$; then recover s from the $y_{a,t}$-type indeterminates using the fact that h is injective; finally recover ℓ from the $z_{s,j}$-type indeterminates using the fact that s is injective.

($\Rightarrow$) Since $P \not\equiv 0$ there is a branching walk $W = (T, h) \in \mathcal{W}$, a shading $s \in \mathcal{S}(W)$ and a bijective labelling $\ell : V(T) \to \{1, 2, \ldots, k\}$ such that the monomial $\mathrm{mon}(W, s, \ell)$ in P has a nonzero coefficient. We must derive a solution S from (W, s, ℓ).

Let us first show that $\mathrm{mon}(W, s, \ell)$ has zero coefficient in P unless h is injective. Suppose that h is not injective; that is, $h(u_0) = h(v_0)$ for some distinct nodes $u_0, v_0 \in V(T)$. If there are many such pairs, take the minimum pair in the lexicographic ordering of 2-subsets of $V(T)$. Define $\ell' : V(T) \to \{1, 2, \ldots, k\}$ for all $v \in V(T)$ by

$$\ell'(v) = \begin{cases} \ell(v_0) & \text{if } v = u_0, \\ \ell(u_0) & \text{if } v = v_0, \\ \ell(v) & \text{otherwise.} \end{cases}$$

Similarly, define $s' : V(T) \to D$ for all $v \in V(T)$ by

$$s'(v) = \begin{cases} s(v_0) & \text{if } v = u_0, \\ s(u_0) & \text{if } v = v_0, \\ s(v) & \text{otherwise.} \end{cases}$$

We observe that ℓ' is bijective and that $\ell' \neq \ell$ because ℓ is bijective. Moreover, s' is consistent because $c(h(u_0)) = c(h(v_0))$ and s is consistent. In this way, to the triple (W, s, ℓ) we associated a different triple (W, s', ℓ') such that $\mathrm{mon}(W, s, \ell) = \mathrm{mon}(W, s', \ell')$. Since $\mathbb{F}_{2^\beta}$ has characteristic 2, these two monomials cancel out in P. Conversely, if we begin from (W, s', ℓ') and follow the same association rule, we get (W, s, ℓ). Hence, the set of all triples $((T, h), s, \ell)$ in which the homomorphism h is not injective is partitioned into pairs, and the two monomials corresponding to each pair cancel out. It follows that the homomorphism h is injective in every triple $((T, h), s, \ell)$ in the preimage of a monomial with a nonzero coefficient in P.

Next suppose that h is injective but s is not injective. Then there is pair of distinct nodes $u_0, v_0 \in V(T)$ that have the same shade $s(u_0) = s(v_0)$. Again, if there are many such pairs we take the lexicographic minimum pair. Define $\ell' : V(T) \to \{1, 2, \ldots, k\}$ as before. Again, to the triple (W, s, ℓ) we assigned a different triple (W, s, ℓ') and again one can verify that $\mathrm{mon}(W, s, \ell) = \mathrm{mon}(W, s, \ell')$. By a similar argument as above, we see that the two monomials corresponding to these two triples cancel out. It follows that the shading s is injective in every triple $((T, h), s, \ell)$ in the preimage of a monomial with a nonzero coefficient in P.

So we must have that both h and s are injective in a three-tuple $((T, h), s, \ell)$ where the monomial $\mathrm{mon}(W, s, \ell)$ in P has a nonzero coefficient. Let $S = V(h(T))$. Because h is injective, $|S| = k$. Because T is connected and h is a homomorphism, we have that $h(T)$ is connected and hence so is $G[S]$. Since s is consistent and injective, S is a solution. ◀

2.5 Evaluating the polynomial in time $O^*(2^k)$

In this section we show that the polynomial P can be evaluated in a given point $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ in time $O^*(2^k)$. To this end, let us rewrite P as a sum of 2^k polynomials such that each of them can be evaluated in time polynomial in the input size. For each $X \subseteq \{1, 2, \ldots, k\}$, let

$$P_X(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \sum_{W=(T,h)\in\mathcal{W}} \sum_{s\in\mathcal{S}(W)} \sum_{\ell:V(T)\to X} \mathrm{mon}(W, s, \ell)\,,$$

Note that we do not assume that the labellings in the third summation are bijective.

▶ **Lemma 2.** $P(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \sum_{X\subseteq\{1,2,\ldots,k\}} P_X(\mathbf{x}, \mathbf{y}, \mathbf{z})$.

Proof. Let us fix a branching walk $W = (T, h) \in \mathcal{W}$ such that $|V(T)| = k$ and a shading $s \in \mathcal{S}(W)$. Because $|V(T)| = k$, a function $\ell : V(T) \to \{1, 2, \ldots, k\}$ is bijective if and only if it is surjective, so

$$\sum_{\substack{\ell:V(T)\to\{1,2,\ldots,k\}\\ \ell \text{ bijective}}} \mathrm{mon}(W, s, \ell) = \sum_{\substack{\ell:V(T)\to\{1,2,\ldots,k\}\\ \ell \text{ surjective}}} \mathrm{mon}(W, s, \ell). \tag{1}$$

Observing again that $\mathbb{F}_{2^\beta}$ has characteristic 2, and hence $-1 = 1$, we have, by the Principle of Inclusion and Exclusion,

$$\sum_{\substack{\ell:V(T)\to\{1,2,\ldots,k\}\\ \ell \text{ surjective}}} \mathrm{mon}(W, s, \ell) = \sum_{X\subseteq\{1,2,\ldots,k\}} \sum_{\ell:V(T)\to X} \mathrm{mon}(W, s, \ell)\,. \tag{2}$$

From (1) and (2) we immediately obtain

$$P(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \sum_{W=(T,h)\in\mathcal{W}} \sum_{s\in\mathcal{S}(W)} \sum_{X\subseteq\{1,2,\ldots,k\}} \sum_{\ell:V(T)\to X} \mathrm{mon}(W, s, \ell)\,. \tag{3}$$

The claim follows by changing the order of summation. ◀

Now we are left with a tedious job of evaluating $P_X(\mathbf{x},\mathbf{y},\mathbf{z})$ in polynomial time. This is slightly technical because we consider properly ordered branching walks. To simplify notation in the running time bounds, let us write e for the number of edges in G, and $\mu = O(\beta\log\beta\log\log\beta)$ for the time needed to multiply or add two elements of $\mathbb{F}_{2^\beta}$.

▶ **Lemma 3.** *Given a nonempty subset $X \subseteq \{1,2,\ldots,k\}$ and three vectors $\mathbf{x},\mathbf{y},\mathbf{z}$ of values in $\mathbb{F}_{2^\beta}$ as input, the value of $P_X(\mathbf{x},\mathbf{y},\mathbf{z})$ can be computed by dynamic programming in time $O(k^2e\mu)$ and space $O(ke\beta)$.*

Proof. Recall that we assume that $V(G)=\{1,2,\ldots,n\}$. For a vertex $a\in V(G)$, denote the ordered sequence of neighbors of a in G by $a_1 < a_2 < \cdots < a_{\deg_G(a)}$. For each $a \in V(G)$, $1 \le i \le \deg_G(a)+1$, and $0 \le l \le k$, denote by $\mathcal{W}(a,i,l)$ the set of properly ordered branching walks $W=(T,h)$ such that (i) W starts from a, (ii) for any child node u of 1 in T it holds that $h(u)=a_j$ implies $j \ge i$, and (iii) $|V(T)|=l$.

Our objective is to compute a three-dimensional array A_X whose entries are defined by

$$A_X[a,i,l] = \sum_{W=(T,h)\in\mathcal{W}(a,i,l)} \sum_{s\in\mathcal{S}_{V(T)\setminus\{1\}}(W)} \sum_{\ell:V(T)\setminus\{1\}\to X} \prod_{\{u,v\}\in E(T)} x_{h(u),h(v)} \prod_{v\in V(T)\setminus\{1\}} y_{h(v),s(v)} z_{s(v),\ell(v)}\,.$$

The entries of A_X admit the following recurrence. For $i=\deg_G(a)+1$ or $l=1$, we have

$$A_X[a,i,l] = \begin{cases} 1 & \text{if } l=1,\\ 0 & \text{otherwise.}\end{cases}$$

For $1\le i\le \deg_G(a)$ and $2 \le l \le k$, we have

$$\begin{aligned} A_X[a,i,l] \;&=\; A_X[a,i+1,l] + \\ &\quad x_{a,a_i}\cdot\left(\sum_{t\in D(c(a_i))}\sum_{j\in X} y_{a_i,t}z_{t,j}\right)\cdot \sum_{\substack{l_1+l_2=l\\ l_1,l_2\ge 1}} A_X[a,i+1,l_1]\cdot A_X[a_i,1,l_2]. \end{aligned} \tag{4}$$

To see that the recurrence is correct, observe that the two lines above correspond to properly ordered branching walks in $\mathcal{W}(a,i,l)$ where either (a) there is no child node u of 1 in T such that $h(u)=a_i$ or (b) there is a unique such child. (At most one such child may exist because the branching walk is properly ordered.)

To recover the value of the polynomial $P_X(\mathbf{x},\mathbf{y},\mathbf{z})$, we observe that

$$P_X(\mathbf{x},\mathbf{y},\mathbf{z}) = \sum_{r\in V}\sum_{t\in D(c(r))}\sum_{j\in X} y_{r,t}z_{t,j}\cdot A_X[r,1,k]\,. \tag{5}$$

The time bound follows by noting that the values of $\sum_{t\in D(c(a))}\sum_{j\in X} y_{a,t}z_{t,j}$ can be precomputed and tabulated in $O(k^2n\mu)$ time to accelerate the computations for the individual entries of A_X. ◀

2.6 The decision algorithm

▶ **Lemma 4** (DeMillo and Lipton [7], Schwartz [19], Zippel [21]). *Let $P(x_1, x_2, \ldots, x_m)$ be a nonzero polynomial of degree at most d over a field $\mathbb{F}$ and let S be a finite subset of $\mathbb{F}$. Then, the probability that P evaluates to zero on a random element $(a_1, a_2, \ldots, a_m) \in S^m$ is bounded by $d/|S|$.*

▶ **Theorem 5.** *The* Maximum Graph Motif *problem admits a Monte Carlo algorithm that runs in $O(2^k k^2 e\mu)$ time and in polynomial space, with the following guarantees: (i) the algorithm always returns NO when given a NO-instance as input, (ii) the algorithm returns YES with probability at least $1/2$ when given a YES-instance as input.*

Proof. The algorithm is as follows. Select values for the variables in $\mathbf{x}, \mathbf{y}, \mathbf{z}$ independently and uniformly at random from $\mathbb{F}_{2^\beta}$. Then iterate over all $X \subseteq \{1, 2, \ldots, k\}$ and use the algorithm in Lemma 3 to evaluate the value $P_X(\mathbf{x}, \mathbf{y}, \mathbf{z})$. Accumulate the sum of the values $P_X(\mathbf{x}, \mathbf{y}, \mathbf{z})$ to obtain $P(\mathbf{x}, \mathbf{y}, \mathbf{z})$ by Lemma 2. If $P(\mathbf{x}, \mathbf{y}, \mathbf{z}) \neq 0$, answer YES and otherwise answer NO.

When the input is a NO instance, the polynomial P is the zero polynomial by Lemma 1, so the algorithm returns NO. When the input is a YES instance, by Lemma 1 the polynomial P is nonzero. The degree of P is exactly $3k - 1$, while the size of $\mathbb{F}_{2^\beta}$ is $2^{\lceil \log k \rceil + 3} \geq 8k$. Thus, by Lemma 4 the probability that P evaluated to 0 is bounded by $\frac{3k-1}{8k} < \frac{1}{2}$. ◀

3 Variants of the Graph Motif Problem

In Section 2 we described an algorithm for Maximum Graph Motif. It is easy to see that the algorithm can also be used to solve classical Graph Motif by setting $k = |M|$. Another variant of the problem studied in the literature is the list version of the problem, where every vertex $a \in V(G)$ is assigned *a set* of colors $C(a) \subseteq C$, not just one color $c(a)$, and for every vertex in a solution we can choose any of its colors to match the multiset M. It is straightforward to modify our algorithm for Maximum Graph Motif to solve the list version: in the dynamic programming of Lemma 3, in (4) instead of summing over $t \in D(c(a_i))$ we sum over $t \in \bigcup_{q \in C(a_i)} D(q)$, and similarly in (5).

Although Maximum Graph Motif (introduced in [8]) is a natural optimization version of the problem, it is not the only one. Two more optimization variants were introduced in [9]; we describe their decision versions below.

Min-Add
Input: Graph $G = (V, E)$, a coloring $c : V \to C$, a multiset of colors M, and $d \in \mathbb{N}$.
Question: Is there a subset $S \subseteq V$ such that $G[S]$ is connected, $M \subseteq c(S)$ and $|c(S) \setminus M| \leq d$?

Min-Substitute
Input: Graph $G = (V, E)$, a coloring $c : V \to C$, a multiset of colors M, and $d \in \mathbb{N}$.
Question: Is there a subset $S \subseteq V$ such that $G[S]$ is connected and $c(S)$ can be obtained from M with at most d substitutions?

In this paper we introduce a new variant, which is a generalization of Maximum Graph Motif, Min-Add and Min-Substitute. We believe that it might be useful in bioinformatics applications.

Consider the following three operations on a multiset M over a set of colors C:

1. insertion (I): adds a copy of $c \in C$ to M,

2. deletion (D): removes a copy of $c \in M$ from M,
3. substitution (S): removes a copy of $c_1 \in M$ from M and adds a copy of $c_2 \in C$ to M.

Associate with each of the three operations a nonnegative integer cost $\kappa_I, \kappa_D, \kappa_S$. Consider a sequence σ of the three operations applied to a multiset M. Let $m_{\mathrm{I}}, m_{\mathrm{D}}, m_{\mathrm{S}}$ be the numbers of insertions, deletions, and substitutions in σ. Then the *cost* of σ is defined as $m_{\mathrm{I}}\kappa_{\mathrm{I}} + m_{\mathrm{D}}\kappa_{\mathrm{D}} + m_{\mathrm{S}}\kappa_{\mathrm{S}}$. Moreover, for two multisets M and M', the *weighted edit distance* is defined as the minimum cost $\kappa(M, M')$ of a sequence of operations that turns M into M'.

CLOSEST GRAPH MOTIF
Input: Graph $G = (V, E)$, a coloring $c : V \to C$, a multiset of colors M, and numbers $d, \kappa_{\mathrm{I}}, \kappa_{\mathrm{D}}, \kappa_{\mathrm{S}} \in \mathbb{N}$.
Question: Is there a subset $S \subseteq V$ such that $G[S]$ is connected and $\kappa(M, c(S)) \leq d$?

Note that MIN ADD reduces to CLOSEST GRAPH MOTIF by settting $\kappa_{\mathrm{I}} = 1$, $\kappa_{\mathrm{D}} = \kappa_{\mathrm{S}} = d + 1$. Similarly, for MIN SUBSTITUTE set $\kappa_{\mathrm{S}} = 1$, $\kappa_{\mathrm{I}} = \kappa_{\mathrm{D}} = d + 1$.

We next describe an algorithm for CLOSEST GRAPH MOTIF subject to the parameterization that we are given an additional integer $k \in \mathbb{N}$ as input and the subset S must satisfy $|S| = k$. We will present an $O^*(2^k)$-time polynomial space algorithm, assuming that d is bounded by a polynomial function in n. Note that when parameterized by the edit distance (which also seems natural) the problem is unlikely to admit an FPT algorithm since GRAPH MOTIF is NP-hard. Since the algorithm is a rather straightforward extension of the algorithm for MAXIMUM GRAPH MOTIF presented in Section 2, we only sketch it by describing the modifications needed to handle the more general problem.

We proceed to define an analog of the polynomial P from Section 2. We use the same indeterminates as before, with additional indeterminates for tracking substitutions and the edit distance. Towards this end, for each $a \in V(G)$, introduce the indeterminate w_a. Denote by $\mathbf{w}$ the sequence of all such indeterminates. Introduce one further indeterminate η for tracking the edit distance.

Recall that in the polynomial P, every monomial corresponds to a consistently shaded and bijectively labelled branching walk (W, s, ℓ). In the new polynomial Q, every monomial corresponds to a quadruple (W, f, s, ℓ), where $f : V(T) \to \{0, 1\}$ is an indicator function for substitutions. That is, $f(v) = 1$ and $s(v) = (q, i)$ for a node v means that a copy of color q (the copy corresponding to the ith shade of q) is substituted by the color $c(h(v))$. We need to modify the set of shades in order to make it accept insertions. For a color $q \in C$ let $D'(q) = \{(q, i) \;:\; i = 1, 2, \ldots, m(q) + \lfloor d/\kappa_{\mathrm{I}} \rfloor\}$ and let $D' = \bigcup_{q \in C} D'(q)$. In Q the shading function s maps $V(T)$ to D'. The meaning of this modification is that the shade (q, i) for $i > m(q)$ corresponds to an inserted shade of color q. Accordingly, the notion of consistency of a shading requires a modification in order to accept substitutions. We say that a (partial) shading $s : U \to D'$ of a branching walk $W = (T, h)$ and a substitution indicator $f : V(T) \to \{0, 1\}$ is *consistent* if for every node $v \in U \subseteq V(T)$ one of the following conditions holds: (i) if $f(v) = 0$ then $s(v) = (c(h(v)), i)$ for some i, or (ii) if $f(v) = 1$ then $s(v) \in D$. For a given branching walk $W = (T, h)$, a substitution indicator $f : V(T) \to \{0, 1\}$, and a set $U \subseteq V(T)$, denote by $\mathcal{S}_U(W, f)$ the set of all (partial) shadings $s : U \to D'$ of W that are consistent. Let us abbreviate $\mathcal{S}(W, f) = \mathcal{S}_{V(T)}(W, f)$.

The following lemma will be useful in the construction of Q.

▶ **Lemma 6.** *Consider a sequence σ of m_I insertions, m_S substitutions and a number of deletions which transforms a multiset M into a multiset M' of size k. Then, the cost of σ is equal to $m_I(\kappa_I + \kappa_D) + m_S\kappa_S + (|M| - k)\kappa_D$.*

Proof. Observe that σ contains $|M| + m_{\mathrm{I}} - k$ deletions. ◀

By the above lemma, given a quadruple (W, f, s, ℓ) the cost of the sequence of operations corresponding to this quadruple is

$$\kappa(f,s) = \left(\sum_{q\in C}\sum_{i=1}^{\lfloor d/\kappa_{\mathrm{I}}\rfloor} |s^{-1}((q, m(q)+i))|\right)(\kappa_{\mathrm{I}}+\kappa_{\mathrm{D}}) + |f^{-1}(1)|\kappa_{\mathrm{S}} + (|M|-k)\kappa_{\mathrm{D}}\,.$$

For a substitution indicator function $f : V(T) \to \{0,1\}$ and a homomorphism $h : V(T) \to V(G)$, let us write $\mathbf{w}_h^f = \prod_{u\in V(T)} w_{h(u)}^{f(u)}$ for the indicator monomial of f given h.

Now we are ready to define the polynomial

$$Q(\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{w},\eta) = \sum_{W=(T,h)\in\mathcal{W}}\ \sum_{f:V(T)\to\{0,1\}}\ \sum_{s\in\mathcal{S}(W,f)}\ \sum_{\substack{\ell:V(T)\to\{1,2,\dots,k\}\\ \ell\text{ bijective}}} \operatorname{mon}(W,s,\ell)\mathbf{w}_h^f\eta^{\kappa(f,s)}\,.$$

▶ **Lemma 7.** *Let* $Q(\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{w},\eta) = \sum_{i\ge 0} Q_i(\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{w})\eta^i$*. Then, we have* $Q_i \not\equiv 0$ *for an* $i \le d$ *if and only if there exists a solution* $S \subseteq V(G)$ *with* $\kappa(M, c(S)) \le d$*.*

Proof. Analogous to the proof of Lemma 1, with the following modifications to handle the substitution indicator f.

($\Leftarrow$) Recover h as before, use the editing sequence for $i = \kappa(M, c(S)) \le d$ to construct a substitution indicator f, then define s and ℓ. To conclude that $Q_i \not\equiv 0$, reconstruct a quadruple (W, s, f, ℓ) from its monomial representation as before but with the additional observation that when h is injective we can recover f from $\mathbf{w}_h^f$.

($\Rightarrow$) When h is not injective, define f' from f by transposing the images of u_0 and v_0 under f. When h is injective but s is not injective, proceed as before. Construct S as before. From f and s we can read off a sequence of edits to transform M to $c(S)$ with cost $i \le d$. ◀

It is immediate that once we can evaluate polynomial Q in a given vector $(\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{w},\eta)$ we can test whether the polynomials Q_i are nonzero using the DeMillo–Lipton–Schwartz–Zippel Lemma and Lagrange interpolation. By Lemma 2, the task of evaluating Q in time $O^*(2^k)$ boils down to evaluating Q_X in polynomial time (where Q_X is defined analogously as P_X). The evaluation proceeds as in Lemma 3, with minor changes to the dynamic programming recurrence. In particular, in (4) we change the expression $\sum_{s\in D(c(a_i))}\sum_{j\in X} y_{a_i,s}z_{s,j}$ into

$$\sum_{s\in D(c(a_i))}\sum_{j\in X} y_{a_i,s}z_{s,j} + \sum_{p=m(c(a_i))+1}^{m(c(a_i))+\lfloor d/\kappa_{\mathrm{I}}\rfloor}\sum_{j\in X} y_{a_i,(c(a_i),p)}z_{(c(a_i),p),j}\eta^{\kappa_{\mathrm{I}}+\kappa_{\mathrm{D}}} + \sum_{s\in D}\sum_{j\in X} y_{a_i,s}z_{s,j}w_{a_i}\eta^{\kappa_{\mathrm{S}}}\,.$$

The three summands correspond to the three possibilities: (i) a_i just gets a color from M, (ii) a_i gets a new copy of the color $c(a_i)$ that is inserted into M, (iii) one copy of a color from M is substituted by one copy of the color $c(a_i)$. A similar change is required for the expression (5), including also multiplication of the whole expression by $\eta^{(|M|-k)\kappa_{\mathrm{D}}}$. (Alternatively, one may offset the final edit distance by $(|M|-k)\kappa_{\mathrm{D}}$.) Precomputing the above expression takes $O(nk(k+d+|M|)\mu)$ time so single evaluation of Q_X is performed in time $O(k^2e\mu + nk(k+d+|M|)\mu) = O((ke+nd+n|M|)k\mu)$.

We conclude with the following theorem which follows from considerations in this section. The additional factor of $(k+|M|)d$ in the running time is caused by the use of Lagrange interpolation, which requires $O((k+|M|)d)$ evaluations of Q. Note that the degree of η in Q is bounded by $(k+|M|)(\kappa_{\mathrm{I}}+\kappa_{\mathrm{D}}+\kappa_{\mathrm{S}}) = O((k+|M|)d)$ because we can assume $\kappa_{\mathrm{I}},\kappa_{\mathrm{D}},\kappa_{\mathrm{S}} \le d+1$. We also note that for every $i \ge 0$, we have $\deg(Q_i) \le (3k-1)k$, so to get the bound on the error probability as before, the size of the finite field $\mathbb{F}_{2^\beta}$ should be at least $2(3k-1)k$. On the other hand, Lagrange interpolation requires $\mathbb{F}_{2^\beta}$ to have size at least $(k+|M|)(\kappa_{\mathrm{I}}+\kappa_{\mathrm{D}}+\kappa_{\mathrm{S}})$. It suffices to put $\beta = \max\{\lceil 2\log_2 k\rceil, \lceil\log_2(k+|M|)d\rceil\} + 3$.

▶ **Theorem 8.** *The* Closest Graph Motif *problem admits a Monte Carlo algorithm that runs in* $O(2^k(ke+nd+n|M|)(k+|M|)dk\mu)$ *time and in polynomial space.*

4 A reduction from Set Cover

In the Set Cover problem we are given an integer t and a family of sets $\mathcal{S}=\{S_1,S_2,\ldots,S_m\}$ over *the universe* $U=\bigcup_{j=1}^m S_j$ with $n=|U|$. The task is to determine whether there is a subfamily of t sets $S_{i_1},S_{i_2},\ldots,S_{i_t}$ such that $U=\bigcup_{j=1}^t S_{i_j}$.

Cygan *et al.* proved the following result (see Theorem 4.4 in [5][1]).

▶ **Theorem 9** (Cygan et al. [5])**.** *If* Set Cover *can be solved in* $O((2-\epsilon)^{n+t})$ *time for some* $\epsilon>0$ *then it can also be solved in* $O((2-\epsilon')^n)$ *time, for some* $\epsilon'>0$.

We use Theorem 9 to show the following.

▶ **Theorem 10.** *If* Graph Motif *can be solved in* $O((2-\epsilon)^k)$ *time for some* $\epsilon>0$ *then* Set Cover *can be solved in* $O((2-\epsilon')^n)$ *time, for some* $\epsilon'>0$*. Moreover, this holds even for* Graph Motif *restricted to one of the following two extreme cases:*

(i) M *is a set,*
(ii) M *has only two distinct colors.*

Proof. Let $(\mathcal{S},t)$ be an instance of Set Cover. We are going to show a polynomial-time reduction to Graph Motif so that in the resulting instance (G,c,M) the multiset M has cardinality $n+t+1$. Clearly, combined with Theorem 9, this will prove our claim.

Graph $G=(V,E)$ is defined as follows. The vertex set consists of U, t copies of the family $\mathcal{S}$ and a special vertex r, that is, $V=U\cup\{s_i^j : i=1,2,\ldots,m,\ j=1,2,\ldots,t\}\cup\{r\}$. Moreover, $E=\{es_i^j : e\in S_i\}\cup\{rs_i^j : i=1,2,\ldots,m,\ j=1,2,\ldots,t\}$.

To establish case (i), let $M=\{1,2,\ldots,n+t+1\}$. Moreover we put $c(s_i^j)=j$ for every $i=1,2,\ldots,m,\ j=1,2,\ldots,t$. Further, $c(r)=t+1$. The n colors $t+2,t+3,\ldots,n+t+1$ are assigned bijectively to the vertices from U. Now we show that $(\mathcal{S},t)$ is a YES-instance of Set Cover iff (G,c,M) is a YES-instance of Graph Motif. Assume $S_{i_1},S_{i_2},\ldots,S_{i_t}$ is a solution to Set Cover. Then let $S=\{r\}\cup U\cup\{s_{i_j}^j : j=1,2,\ldots,t\}$. It is clear that the multiset of colors on S matches M. Obviously, $G[\{r\}\cup\{s_{i_j}^j : j=1,2,\ldots,t\}]$ is connected. Since for every $e\in U$ there is $j=1,2,\ldots,t$ such that $e\in S_{i_j}$, so $es_{i_j}^j\in E(G[S])$. It follows that $G[S]$ is connected, and hence S is a solution for Graph Motif. Conversely, if S is a solution for Graph Motif in (G,c,M) then for every $j=1,2,\ldots,t$ there is exactly one $i_j\in\{1,2,\ldots,m\}$ such that $s_{i_j}^j\in S$, since the colors of S match M. Moreover, since $G[S]$ is connected we infer that for every $e\in U$ there is $j=1,2,\ldots,t$ such that $es_{i_j}^j\in E(G[S])$. However, then $e\in S_{i_j}$ and it follows that $S_{i_1},2,\ldots,S_{i_t}$ is a solution for Set Cover.

To establish case (ii), let M consist of $n+1$ copies of color 1 and t copies of color 2. We put $c(r)=1$ and $c(e)=1$ for every $e\in U$. All the remaining vertices are colored with 2. The equivalence can be shown very similarly to the case (i), we skip the details. ◀

Acknowledgments

We thank reviewers for helpful comments. The third author thanks Sylwia Antoniuk, Marek Cygan, Michal Debski and Matthias Mnich for helpful discussions on related topics.

[1] Actually Theorem 4.4 is stated in a slightly different way, taking into account the maximum size of sets S_i, but Theorem 9 follows immediately from their proof.

References

1 N. Betzler, M. R. Fellows, C. Komusiewicz, and R. Niedermeier. Parameterized algorithms and hardness results for some graph motif problems. In *Proc. CPM'08*, volume 5029 of *LNCS*, pages 31–43, 2008.

2 A. Björklund. Counting perfect matchings as fast as Ryser. In *Proc. SODA'12*, pages 914–921, 2012.

3 A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Narrow sieves for parameterized paths and packings. *CoRR*, abs/1007.1161, 2010.

4 A. Björklund, P. Kaski, and Ł. Kowalik. Probably optimal graph motifs. *CoRR*, abs/1209.1082, 2012.

5 M. Cygan, H. Dell, D. Lokshtanov, D. Marx, J. Nederlof, Y. Okamoto, R. Paturi, S. Saurabh, and M. Wahlström. On problems as hard as CNF-SAT. In *IEEE Conference on Computational Complexity*, pages 74–84, 2012.

6 M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. M. M. van Rooij, and J. O. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proc. FOCS'11*, pages 150–159, 2011.

7 R. A. DeMillo and R. J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7:193–195, 1978.

8 R. Dondi, G. Fertin, and S. Vialette. Maximum motif problem in vertex-colored graphs. In *Proc. CPM'09*, volume 5577 of *LNCS*, pages 221–235, 2009.

9 R. Dondi, G. Fertin, and S. Vialette. Finding approximate and constrained motifs in graphs. In *Proc. CPM'11*, volume 6661 of *LNCS*, pages 388–401, 2011.

10 M. R. Fellows, G. Fertin, D. Hermelin, and S. Vialette. Sharp tractability borderlines for finding connected motifs in vertex-colored graphs. In *Proc. ICALP'07*, volume 4596 of *LNCS*, pages 340–351, 2007.

11 M. R. Fellows, G. Fertin, D. Hermelin, and S. Vialette. Upper and lower bounds for finding connected motifs in vertex-colored graphs. *J. Comput. Syst. Sci.*, 77(4):799–811, 2011.

12 S. Guillemot and F. Sikora. Finding and counting vertex-colored subtrees. In *Proc. MFCS'10*, volume 6281 of *LNCS*, pages 405–416, 2010.

13 I. Koutis. Faster algebraic algorithms for path and packing problems. In *Proc. ICALP'08*, volume 5125 of *LNCS*, pages 575–586, 2008.

14 I. Koutis. The power of group algebras for constrained multilinear monomial detection. *Dagstuhl meeting 10441*, 2010.

15 I. Koutis. Constrained multilinear detection for faster functional motif discovery. *Information Processing Letters*, 112(22):889 – 892, 2012.

16 I. Koutis and R. Williams. Limits and applications of group algebras for parameterized problems. In *ICALP (1)*, volume 5555 of *LNCS*, pages 653–664, 2009.

17 V. Lacroix, C. G. Fernandes, and M.-F. Sagot. Motif search in graphs: Application to metabolic networks. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 3(4):360–368, 2006.

18 J. Nederlof. Fast polynomial-space algorithms using Möbius inversion: Improving on Steiner tree and related problems. In *Proc. ICALP'09*, volume 5555 of *LNCS*, pages 713–725, 2009.

19 J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

20 R. Williams. Finding paths of length k in $O^*(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, 2009.

21 R. Zippel. Probabilistic algorithms for sparse polynomials. In *Proc. International Symposium on Symbolic and Algebraic Computation*, volume 72 of *LNCS*, pages 216–226, 1979.

Tight bounds for Parameterized Complexity of Cluster Editing*

Fedor V. Fomin[1], Stefan Kratsch[2], Marcin Pilipczuk[3], Michał Pilipczuk[1], and Yngve Villanger[1]

1 Department of Informatics, University of Bergen, Bergen, Norway, `{fomin,michal.pilipczuk,yngve.villanger}@ii.uib.no`

2 MPI Informatics, Saarbrücken, Germany, `skratsch@mpi-inf.mpg.de`

3 Institute of Informatics, University of Warsaw, Poland, `malcin@mimuw.edu.pl`

Abstract

In the Correlation Clustering problem, also known as Cluster Editing, we are given an undirected graph G and a positive integer k; the task is to decide whether G can be transformed into a cluster graph, i.e., a disjoint union of cliques, by changing at most k adjacencies, that is, by adding or deleting at most k edges. The motivation of the problem stems from various tasks in computational biology (Ben-Dor et al., Journal of Computational Biology 1999) and machine learning (Bansal et al., Machine Learning 2004). Although in general Correlation Clustering is APX-hard (Charikar et al., FOCS 2003), the version of the problem where the number of cliques may not exceed a prescribed constant p admits a PTAS (Giotis and Guruswami, SODA 2006).

We study the parameterized complexity of Correlation Clustering with this restriction on the number of cliques to be created. We give an algorithm that

- in time $\mathcal{O}(2^{\mathcal{O}(\sqrt{pk})} + n + m)$ decides whether a graph G on n vertices and m edges can be transformed into a cluster graph with exactly p cliques by changing at most k adjacencies.

We complement these algorithmic findings by the following, surprisingly tight lower bound on the asymptotic behavior of our algorithm. We show that unless the Exponential Time Hypothesis (ETH) fails

- for any constant $0 \leq \sigma \leq 1$, there is $p = \Theta(k^\sigma)$ such that there is no algorithm deciding in time $2^{o(\sqrt{pk})} \cdot n^{\mathcal{O}(1)}$ whether an n-vertex graph G can be transformed into a cluster graph with at most p cliques by changing at most k adjacencies.

Thus, our upper and lower bounds provide an asymptotically tight analysis of the multivariate parameterized complexity of the problem for the whole range of values of p from constant to a linear function of k.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases parameterized complexity, cluster editing, correlation clustering, subexponential algorithms, tight bounds

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.32

* The authors from University of Bergen are supported by the European Research Council (ERC) via grant Rigorous Theory of Preprocessing, reference 267959 and by the Research Council of Norway. The third author is supported by the National Science Centre grant N206 567140 and Foundation for Polish Science.

1 Introduction

Correlation clustering, also known as *clustering with qualitative information* or *cluster editing*, is the problem to cluster objects based only on the qualitative information concerning similarity between pairs of them. For every pair of objects we have a binary indication whether they are similar or not. The task is to find a partition of the objects into clusters minimizing the number of similarities between different clusters and non-similarities inside of clusters. The problem was introduced by Ben-Dor, Shamir, and Yakhini [6] motivated by problems from computational biology, and, independently, by Bansal, Blum, and Chawla [5], motivated by machine learning problems concerning document clustering according to similarities. The correlation version of clustering was studied intensively, including [1, 3, 4, 13, 14, 24, 34].

The graph-theoretic formulation of the problem is the following. A graph K is a *cluster graph* if every connected component of K is a complete graph. Let $G = (V, E)$ be a graph; then $F \subseteq V \times V$ is called a *cluster editing set* for G if $G \triangle F = (V, E \triangle F)$ is a cluster graph. Here $E \triangle F$ is the symmetric difference between E and F. In the optimization version of the problem the task is to find a cluster editing set of minimum size. Constant factor approximation algorithms for this problem were obtained in [1, 5, 13]. On the negative side, the problem is known to be NP-complete [34] and, as was shown by Charikar, Guruswami, and Wirth [13], also APX-hard.

Giotis and Guruswami [24] initiated the study of clustering when the maximum number of clusters that we are allowed to use is stipulated to be a fixed constant p. As observed by them, this type of clustering is well-motivated in settings where the number of clusters might be an external constraint that has to be met. It appeared that p-clustering variants posed new and non-trivial challenges. In particular, in spite of the APX-hardness of the general case, Giotis and Guruswami [24] gave a PTAS for this version of the problem.

A cluster graph G is called a *p-cluster graph* if it has exactly p connected components or, equivalently, if it is a disjoint union of exactly p cliques. Similarly, a set F is a *p-cluster editing set* of G, if $G \triangle F$ is a p-cluster graph. In parameterized complexity, correlation clustering and its restriction to bounded number of clusters were studied under the names Cluster Editing and p-Cluster Editing, respectively.

Cluster Editing *Parameter:* k.
Input: A graph $G = (V, E)$ and a non-negative integer k.
Question: Is there a cluster editing set for G of size at most k?

p-Cluster Editing *Parameters:* p, k.
Input: A graph $G = (V, E)$ and non-negative integers p and k.
Question: Is there a p-cluster editing set for G of size at most k?

The parameterized version of Cluster Editing, and variants of it, were studied intensively [7, 8, 9, 10, 11, 16, 20, 25, 27, 28, 31, 33]. The problem is solvable in time $\mathcal{O}(1.62^k + n + m)$ [7] and it has a kernel with $2k$ vertices [12, 15] (see Section 2 for the definition of a kernel). Shamir et al. [34] showed that p-Cluster Editing is NP-complete for every fixed $p \geq 2$. A kernel with $(p + 2)k + p$ vertices was given by Guo [26].

Our results

We study the impact of the interaction between p and k on the parameterized complexity of p-Cluster Editing. Our main algorithmic result is the following.

▶ **Theorem 1.** *p-Cluster Editing is solvable in time $\mathcal{O}(2^{\mathcal{O}(\sqrt{pk})} + m + n)$.*

It is straightforward to modify our algorithm to work also in the following variants of the problem, where each edge and non-edge is assigned some edition cost: either *(i)* all costs are at least one and k is the bound on the maximum total cost of the solution, or *(ii)* we ask for a set of at most k edits of minimum cost. Let us also remark that, by Theorem 1, if $p = o(k)$ then p-Cluster Editing can be solved in $2^{o(k)}n^{\mathcal{O}(1)}$ time, and thus it belongs to complexity class SUBEPT defined by Flum and Grohe [21, Chapter 16]. Until very recently, the only problems known to be in the class SUBEPT were the problems with additional constraints on the input, like being a planar, H-minor-free, or tournament graph [2, 17]. However, recent algorithmic developments indicate that the structure of the class SUBEPT is much more interesting than expected. It appears that some parameterized problems related to chordal graphs, like Minimum Fill-in or Chordal Graph Sandwich, are also in SUBEPT [23].

We would like to remark that p-Cluster Editing can be also solved in worse time complexity $\mathcal{O}((pk)^{\mathcal{O}(\sqrt{pk})} + m + n)$ using simple guessing arguments. One such algorithm is based on the following observation: Suppose that, for some integer r, we know at least $2r + 1$ vertices from each cluster. Then, if an unassigned vertex has at most r incident modifications, we know precisely to which cluster it belongs: it is adjacent to at least $r + 1$ vertices already assigned to its cluster and at most r assigned to any other cluster. On the other hand, there are at most $2k/r$ vertices with more than r incident modifications. Thus (i) guessing $2r + 1$ vertices from each cluster (or all of them, if there are less than $2r + 1$), and (ii) guessing all vertices with more than r incident modifications, together with their alignment to clusters, results in at most $n^{(2r+1)p}n^{2k/r}p^{2k/r}$ subcases. By pipelining it with the kernelization of Guo [26] and with simple reduction rules that ensure $p \le 6k$ (see Section 3.1 for details), we obtain the claimed time complexity for $r \sim \sqrt{k/p}$.

An approach via *chromatic coding*, introduced by Alon et al. [2], also leads to an algorithm with running time $\mathcal{O}(2^{\mathcal{O}(p\sqrt{k}\log p)} + n + m)$. However, one needs to develop new concepts to construct an algorithm for p-Cluster Editing with complexity bound as promised in Theorem 1, and thus obtain a subexponential complexity for every sublinear p.

The crucial observation is that a p-cluster graph, for $p = \mathcal{O}(k)$, has $2^{\mathcal{O}(\sqrt{pk})}$ edge cuts of size at most k (henceforth called *k-cuts*). As in a YES-instance to the p-Cluster Editing problem each k-cut is a $2k$-cut of a p-cluster graph, we infer a similar bound on the number of cuts if we are dealing with a YES-instance. This allows us to use dynamic programming over the set of k-cuts. Pipelining this approach with a kernelization algorithm for p-Cluster Editing proves Theorem 1.

A new and active direction in parameterized complexity is the pursuit of asymptotically tight bounds on the complexity of problems. In several cases, it is possible to obtain a complete analysis by providing matching lower (complexity) and upper (algorithmic) bounds. We refer to the recent survey of Marx [32], where recent developments in the area are discussed, and the “optimality program" is announced among the main future research directions in parameterized complexity. The most widely used complexity assumption for such tight lower bounds is the *Exponential Time Hypothesis (ETH)*, which posits that no subexponential-time algorithms for k-CNF-SAT or CNF-SAT exist [29].

Following this direction, we complement Theorem 1 with two lower bounds. Our first, main lower bound is based on the following technical Theorem 2, which shows that the exponential time dependence of our algorithm is asymptotically tight for any choice of parameters p and k, where $p = \mathcal{O}(k)$. As one can provide polynomial-time reduction rules that ensure that $p \le 6k$ (see Section 3.1 for details), this provides a full and tight picture of the multivariate parameterized complexity of p-Cluster Editing: we have asymptotically matching upper and lower bounds on the whole interval between p being a constant and linear in k. To the best of our knowledge, this is the first fully multivariate and tight complexity analysis of a parameterized problem.

▶ **Theorem 2.** *For any $\varepsilon > 0$ there is $\delta > 0$ and a polynomial-time algorithm that, given positive integers p and k and a 3-CNF-SAT formula Φ with n variables and m clauses, such that $k, n \geq \varepsilon p$ and $n, m \leq \sqrt{pk}/\varepsilon$, computes a graph G and integer k', such that $k' \leq \delta k$, $|V(G)| \leq \delta\sqrt{pk}$, and*

- *if Φ is satisfiable then there is a $6p$-cluster graph G_0 with $V(G) = V(G_0)$ and such that $|E(G)\triangle E(G_0)| \leq k'$;*
- *if there exists a p'-cluster graph G_0 with $p' \leq 6p$, $V(G) = V(G_0)$ and $|E(G)\triangle E(G_0)| \leq k'$, then Φ is satisfiable.*

As the statement of Theorem 2 may look technical, we gather its two main consequences in Corollaries 3 and 4. We state both corollaries in terms of an easier $p_{\leq}$-Cluster Editing problem, where the number of clusters has to be at most p instead of precisely equal to p. Clearly, this version can be solved by an algorithm for p-Cluster Editing with an additional p overhead in time complexity by trying all possible $p' \leq p$, so the lower bound holds also for harder p-Cluster Editing; however, we are not aware of any reduction in the opposite direction. In both corollaries we use the fact that existence of a subexponential, in both the number of variables and clauses, algorithm for verifying satisfiability of 3-CNF-SAT formulas would violate ETH [29].

▶ **Corollary 3** (♠[1]). *Unless ETH fails, for every $0 \leq \sigma \leq 1$, there is $p = \Theta(k^\sigma)$ such that $p_{\leq}$-Cluster Editing is not solvable in time $2^{o(\sqrt{pk})}|V(G)|^{\mathcal{O}(1)}$.*

▶ **Corollary 4** (♠). *Unless ETH fails, for every constant $p \geq 6$, there is no algorithm solving $p_{\leq}$-Cluster Editing in time $2^{o(\sqrt{k})}|V(G)|^{\mathcal{O}(1)}$ or $2^{o(|V(G)|)}$.*

Note that Theorem 2 and Corollary 3 do not rule out possibility that the general Cluster Editing is solvable in subexponential time. Our second, complementary lower bound shows that when the number of clusters is not constrained, then the problem cannot be solved in subexponential time unless ETH fails. This disproves the conjecture of Cao and Chen [12]. We note that Theorem 5 was independently obtained by Komusiewicz in his PhD thesis [30].

▶ **Theorem 5** (♠). *Unless ETH fails,* Cluster Editing *cannot be solved in time $2^{o(k)}n^{\mathcal{O}(1)}$.*

Clearly, by Theorem 1, the reduction of Theorem 5 must produce an instance where the number of clusters in any solution, if there exists any, is $\Omega(k)$. Therefore, intuitively the hard instances of Cluster Editing are those where every cluster needs just a constant number of adjacent editions to be extracted.

2 Preliminaries

We use n to denote the number of vertices and m the number of edges in the input graph G. For graphs G, H with $V(G) = V(H)$, by $\mathcal{H}(G, H)$ we denote the number of edge modifications needed to obtain H from G, i.e., $\mathcal{H}(G, H) = |E(G)\triangle E(H)|$. By $E(X, Y)$ we denote the set of edges having one endpoint in X and second in Y.

[1] Due to space constraints, the proofs of all statements marked with ♠ are omitted. The full version of this paper is available at http://arxiv.org/abs/1112.4419.

A parameterized problem Π is a subset of $\Gamma^* \times \mathbb{N}$ for some finite alphabet Γ. An instance of a parameterized problem consists of (x, k), where k is called the parameter. A central notion in parameterized complexity is *fixed-parameter tractability (FPT)* which means, for a given instance (x, k), solvability in time $f(k) \cdot p(|x|)$, where f is an arbitrary computable function of k and p is a polynomial in the input size. We refer to the book of Downey and Fellows [19] for further reading on parameterized complexity.

A *kernelization algorithm* for a parameterized problem $\Pi \subseteq \Gamma^* \times \mathbb{N}$ is an algorithm that given $(x, k) \in \Gamma^* \times \mathbb{N}$ outputs in time polynomial in $|x| + k$ a pair $(x', k') \in \Gamma^* \times \mathbb{N}$, called a *kernel* such that $(x, k) \in \Pi$ if and only if $(x', k') \in \Pi$, $|x'| \leq g(k)$, and $k' \leq k$, where g is some computable function.

We also need the following result of Guo [26].

▶ **Proposition 6** ([26]). p-Cluster Editing admits a kernel with $(p + 2)k + p$ vertices. The running time of the kernelization algorithm is $\mathcal{O}(n + m)$, where n is the number of vertices and m the number of edges in the input graph G.

3 A subexponential algorithm for p-Cluster Editing

In this section we prove Theorem 1, that is, we show a $\mathcal{O}(2^{\mathcal{O}(\sqrt{pk})} + n + m)$-time algorithm for p-Cluster Editing.

3.1 Reduction for large p

The first step of our algorithm is an application of the kernelization algorithm by Guo [26] (Proposition 6) followed by some additional preprocessing rules that ensure that $p \leq 6k$. These additional rules are encapsulated in the following technical lemma.

▶ **Lemma 7** (♠). *There exists a polynomial time algorithm that, given an instance (G, p, k) of p-Cluster Editing, outputs an equivalent instance (G', p', k), where G' is an induced subgraph of G and $p' \leq 6k$.*

The key idea behind Lemma 7 is the observation that if $p > 2k$, then at least $p - 2k$ clusters in the final cluster graph cannot be touched by the solution, hence they must have been present as isolated cliques already in the beginning. Hence, if $p > 6k$ then we have to already see $p - 2k > 4k$ isolated cliques; otherwise, we may safely provide a negative answer. Although these cliques may be still merged (to decrease the number of clusters) or split (to increase the number of clusters), we can apply greedy arguments to identify a clique that may be safely assumed to be untouched by the solution. Hence we can remove it from the graph and decrement p by one. Although the greedy arguments seem very intuitive, their formal proofs turn out to be somewhat technical.

3.2 Small cuts

We now proceed to the algorithm itself. Let us introduce the key notion.

▶ **Definition 8.** Let $G = (V, E)$ be an undirected graph. A partition (V_1, V_2) of V is called a *k-cut of G* if $|E(V_1, V_2)| \leq k$.

▶ **Lemma 9.** *k-cuts of a graph G can be enumerated with polynomial time delay.*

Proof. We follow the standard branching. We order the vertices arbitrarily, start with empty V_1, V_2 and for each consecutive vertex v we branch into two subcases: we put v either into V_1

or into V_2. Once the alignment of all vertices is decided, we output the partition. However, each time we put a vertex in one of the sets, we run a polynomial-time max-flow algorithm to check whether the minimum edge cut between V_1 and V_2 constructed so far is at most k. If not, then we terminate this branch as it certainly cannot result in any solutions found. Thus, we always pursue a branch that results in at least one feasible solution, and finding the next solution occurs within a polynomial number of steps. ◀

Intuitively, k-cuts of the graph G form the search space of the algorithm. Therefore, we would like to bound their number. We do this by firstly bounding the number of cuts of a cluster graph, and then using the fact that a YES-instance is not very far from some cluster graph. We begin with the following bound on binomial coefficients.

▶ **Lemma 10** (♠). *If a, b are nonnegative integers, then $\binom{a+b}{a} \le 2^{2\sqrt{ab}}$.*

▶ **Lemma 11.** *Let K be a cluster graph containing at most p clusters, where $p \le 6k$. Then the number of k-cuts of K is at most $2^{8\sqrt{pk}}$.*

Proof. By slightly abusing the notation, assume that K has exactly p clusters, some of which may be empty. Let $C_1, C_2, \ldots, C_p$ be these clusters and $c_1, c_2, \ldots, c_p$ be their sizes, respectively. We firstly establish a bound on the number of partitions (V_1, V_2) such that the cluster C_i contains x_i vertices from V_1 and y_i from V_2. Then we discuss how to bound the number of ways of selecting pairs x_i, y_i summing up to c_i for which the number of k-cuts is positive. Multiplying the obtained two bounds gives us the claim.

Having fixed the numbers x_i, y_i, the number of ways in which the cluster C_i can be partitioned is equal to $\binom{x_i+y_i}{x_i}$. Note that $\binom{x_i+y_i}{x_i} \le 2^{2\sqrt{x_i y_i}}$ by Lemma 10. Observe that there are $x_i y_i$ edges between V_1 and V_2 inside the cluster C_i, so if (V_1, V_2) is a k-cut, then $\sum_{i=1}^{p} x_i y_i \le k$. By applying the Cauchy-Schwarz inequality we infer that $\sum_{i=1}^{p} \sqrt{x_i y_i} \le \sqrt{p} \cdot \sqrt{\sum_{i=1}^{p} x_i y_i} \le \sqrt{pk}$. Therefore, the number of considered cuts is bounded by

$$\prod_{i=1}^{p} \binom{x_i + y_i}{x_i} \le 2^{2\sum_{i=1}^{p} \sqrt{x_i y_i}} \le 2^{2\sqrt{pk}}.$$

Moreover, observe that $\min(x_i, y_i) \le \sqrt{x_i y_i}$; hence, $\sum_{i=1}^{p} \min(x_i, y_i) \le \sqrt{pk}$. Thus, the choice of x_i, y_i can be modeled by first choosing for each i, whether $\min(x_i, y_i)$ is equal to x_i or to y_i, and then expressing $\lfloor \sqrt{pk} \rfloor$ as the sum of $p+1$ nonnegative numbers: $\min(x_i, y_i)$ for $1 \le i \le p$ and the rest, $\lfloor \sqrt{pk} \rfloor - \sum_{i=1}^{p} \min(x_i, y_i)$. The number of choices in the first step is equal to $2^p \le 2^{\sqrt{6pk}}$, and in the second is equal to $\binom{\lfloor \sqrt{pk} \rfloor + p}{p} \le 2^{\sqrt{pk} + \sqrt{6pk}}$. Therefore, the number of possible choices of x_i, y_i is bounded by $2^{(1+2\sqrt{6})\sqrt{pk}} \le 2^{6\sqrt{pk}}$. Hence, the total number of k-cuts is bounded by $2^{6\sqrt{pk}} \cdot 2^{2\sqrt{pk}} = 2^{8\sqrt{pk}}$, as claimed. ◀

▶ **Lemma 12.** *If (G, p, k) is a YES-instance of p-Cluster Editing with $p \le 6k$, then the number of k-cuts of G is bounded by $2^{8\sqrt{2pk}}$.*

Proof. Let K be a cluster graph with at most p clusters such that $\mathcal{H}(G, K) \le k$. Observe that every k-cut of G is also a $2k$-cut of K, as K differs from G by at most k edge modifications. The claim follows from Lemma 11. ◀

3.3 The algorithm

Proof of Theorem 1. Let $(G = (V, E), p, k)$ be the given p-Cluster Editing instance. By making use of Proposition 6, we can assume that G has at most $(p+2)k + p$ vertices, thus all the factors polynomial in the size of G can be henceforth hidden within the $2^{\mathcal{O}(\sqrt{pk})}$ factor. Application of Proposition 6 gives the additional $\mathcal{O}(n+m)$ summand to the complexity. By further usage of Lemma 7 we can also assume that $p \leq 6k$. Note that application of Lemma 7 can spoil the bound $|V(G)| \leq (p+2)k+p$ as p can decrease; however the number of vertices of the graph is still bounded in terms of initial p and k.

We now enumerate k-cuts of G with polynomial time delay. If we exceed the bound $2^{8\sqrt{2pk}}$ given by Lemma 12, we know that we can safely answer NO, so we immediately terminate the computation and give a negative answer. Therefore, we can assume that we have computed the set $\mathcal{N}$ of all k-cuts of G and $|\mathcal{N}| \leq 2^{8\sqrt{2pk}}$.

Assume that (G,p,k) is a YES-instance and let K be a cluster graph with at most p clusters such that $\mathcal{H}(G,K) \leq k$. Again, let $C_1, C_2, \ldots, C_p$ be the clusters of K. Observe that for every $j \in \{0,1,2,\ldots,p\}$, the partition $\left(\bigcup_{i=1}^{j} V(C_i), \bigcup_{i=j+1}^{p} V(C_i)\right)$ has to be the k-cut with respect to G, as otherwise there would be more than k edges that need to be deleted from G in order to obtain K. This observation enables us to use a dynamic programming approach on the set of cuts.

We construct a directed graph D, whose vertex set is equal to $\mathcal{N} \times \{0,1,2,\ldots,p\} \times \{0,1,2,\ldots,k\}$; note that $|V(D)| = 2^{\mathcal{O}(\sqrt{pk})}$. We create arcs going from $((V_1,V_2),j,\ell)$ to $((V_1',V_2'),j+1,\ell')$, where $V_1 \subsetneq V_1'$ (hence $V_2 \supsetneq V_2'$), $j \in \{0,1,2,\ldots,p-1\}$ and $\ell' = \ell + |E(V_1, V_1' \setminus V_1)| + |\overline{E}(V_1' \setminus V_1, V_1' \setminus V_1)|$ ($(V,\overline{E})$ is the complement of the graph G). The arcs can be constructed in $2^{\mathcal{O}(\sqrt{pk})}$ time by checking for all the pairs of vertices whether they should be connected. We claim that the answer to the instance (G,p,k) is equivalent to reachability of any of the vertices of form $((V,\emptyset),p,\ell)$ from the vertex $((\emptyset,V),0,0)$.

In one direction, if there is a path from $((\emptyset,V),0,0)$ to $((V,\emptyset),p,\ell)$ for some $\ell \leq k$, then the consecutive sets $V_1' \setminus V_1$ along the path form clusters C_i of a cluster graph K, whose editing distance to G is accumulated on the last coordinate, thus bounded by k. In the second direction, if there is a cluster graph K with clusters $C_1, C_2, \ldots, C_p$ within editing distance at most k from G, then vertices $\left(\left(\bigcup_{i=1}^{j} V(C_i), \bigcup_{i=j+1}^{p} V(C_i)\right), j, \mathcal{H}\left(G\left[\bigcup_{i=1}^{j} V(C_i)\right], K\left[\bigcup_{i=1}^{j} V(C_i)\right]\right)\right)$ form a path from $((\emptyset,V),0,0)$ to $((V,\emptyset),p,\mathcal{H}(G,K))$. Note that all these triples are indeed vertices of the graph D, as $\left(\bigcup_{i=1}^{j} V(C_i), \bigcup_{i=j+1}^{p} V(C_i)\right)$ are k-cuts of G.

Reachability in a directed graph can be tested in linear time with respect to the number of vertices and arcs. We can now apply this algorithm to the graph D and conclude solving the p-Cluster Editing instance in $\mathcal{O}(2^{\mathcal{O}(\sqrt{pk})} + n + m)$ time. ◀

4 Multivariate lower bound

This section is devoted to sketching the proof of Theorem 2. As the provided reduction is very technical, in this extended abstract we only provide the construction of the graph G, explaining also all the necessary intuition, and sketch the completeness implication, i.e., how to translate a satisfying assignment of Φ into a $6p$-cluster graph G_0 close to G. To ease the presentation, in this extended abstract we show the proof for $\varepsilon = 1$.

4.1 Preprocessing of the formula

We start with a step that regularizes the input formula Φ, while increasing its size only by a constant factor. The purpose of this step is to ensure that, when we translate a satisfying assignment of Φ into a cluster graph G_0 in the completeness step, the clusters are of the same size, and therefore contain the minimum possible number of edges. This property is crucial in the argumentation of the soundness step. The proof of the following lemma consists of several steps that ensure consecutive properties of formula Φ' by syntactic modifications, like copying variables and clauses.

▶ **Lemma 13** (♠). *There exists a polynomial-time algorithm that, given a 3-CNF formula Φ with n variables and m clauses and an integer $p \leq n$, constructs a 3-CNF formula Φ' with n' variables and m' clauses together with a partition of the variable set* $\mathrm{Vars}(\Phi')$ *into p parts* Vars^r, $1 \leq r \leq p$, *such that the following properties hold:*

(a) Φ' *is satisfiable iff* Φ *is;*

(b) in Φ' *every clause contains exactly three literals corresponding to different variables;*

(c) in Φ' *every variable appears exactly three times positively and exactly three times negatively;*

(d) n' *is divisible by p and, for each* $1 \leq r \leq p$, *we have* $|\mathrm{Vars}^r| = n'/p$ *(i.e., the variables are split evenly between the parts* Vars^r*);*

(e) if Φ' *is satisfiable, then there exists a satisfying assignment of* $\mathrm{Vars}(\Phi')$ *with the property that in each part* Vars^r *the numbers of variables set to true and to false are equal.*

(f) $n' + m' = \mathcal{O}(n + m)$.

4.2 Construction

We now sketch how to compute the graph G and the integer k' from the formula Φ' given by Lemma 13. As Lemma 13 increases the size of the formula by a constant factor, we have that $n', m' = \mathcal{O}(\sqrt{pk})$ and $|\mathrm{Vars}^r| = n'/p = \mathcal{O}(\sqrt{k/p})$ for $1 \leq r \leq p$. The idea is to pack the variables from each part Vars^r, for $1 \leq r \leq p$, into group gadgets, each costing 6 cliques. Evaluation of the variables from each part corresponds to some clustering strategy inside the group gadget. The clauses are encoded by additional groups of vertices, whose connections to group gadgets ensure that they can be split among the clusters optimally iff at least one literal satisfies the clause.

We proceed with the description of group gadgets. Let $L = 1000 \cdot \left(1 + \frac{n'}{p}\right) = \mathcal{O}(\sqrt{k/p})$. For each part Vars^r, $1 \leq r \leq p$, we create six cliques Q^r_α, $1 \leq \alpha \leq 6$, each of size L. Let $\mathcal{Q}$ be the set of all vertices of all cliques Q^r_α. In this manner we have $6p$ cliques. Intuitively, if we seek for a $6p$-cluster graph close to G, then the cliques are large enough so that merging two cliques is too expensive — in the intended solution we have exactly one clique in each cluster. One may view the construction as a procedure of assigning vertices not from $\mathcal{Q}$ to different cliques Q^r_α.

For every variable $x \in \mathrm{Vars}^r$, we create six vertices $w^x_{1,2}, w^x_{2,3}, \ldots, w^x_{5,6}, w^x_{6,1}$. Connect them into a cycle in this order; this cycle is called a *6-cycle for the variable x*. Moreover, for each $1 \leq \alpha \leq 6$ and $v \in V(Q^r_\alpha)$, create edges $vw^x_{\alpha-1,\alpha}$ and $vw^x_{\alpha,\alpha+1}$ (we assume that the indices behave cyclically, i.e., $w^x_{6,7} = w^x_{6,1}$, $Q^r_7 = Q^r_1$ etc.). Let $\mathcal{W}$ be the set of all vertices $w^x_{\alpha,\alpha+1}$ for all variables x. Intuitively, the cheapest way to cut the 6-cycle for variable x is to assign the vertices $w^x_{\alpha,\alpha+1}$, $1 \leq \alpha \leq 6$, all either to the clusters with cliques with only odd indices or only with even indices. Choosing even indices corresponds to setting x to false, while choosing odd ones corresponds to setting x to true, and both choices lead to saving exactly 3 editions inside the 6-cycle. By property (e) of formula Φ' we know that if Φ' is satisfiable, then in some satisfying assignment exactly half of the variables in each group are assigned true value, and half false. For this satisfying assignment, each clique Q^r_α will be assigned exactly the same number of vertices from $\mathcal{W}$.

We now proceed with the description of the encoding of the clauses. Let $r(x)$ be the index of the part that contains variable x, that is, $x \in \mathrm{Vars}^{r(x)}$. In each clause C we (arbitrarily) enumerate variables: for $1 \le \eta \le 3$, let $\mathrm{var}(C, \eta)$ be the variable in the η-th literal of C, and $\mathrm{sgn}(C, \eta) = 0$ if the η-th literal is negative and $\mathrm{sgn}(C, \eta) = 1$ otherwise.

For every clause C create nine vertices: $s^C_{\beta,\xi}$ for $1 \le \beta, \xi \le 3$. Let $\mathcal{S}$ be the set of all the vertices created in this manner. Let us first focus on vertices $s^C_{1,1}, s^C_{1,2}, s^C_{1,3}$.

- For each $1 \le \eta \le 3$ and each $\xi \in \{1, 2, 3\}$, create an edge $s^C_{1,\xi} w^{\mathrm{var}(C,\eta)}_{2\eta-1,2\eta}$;
- for each $1 \le \eta \le 3$ connect $s^C_{1,1}$ to all the vertices of one of the cliques adjacent to $w^{\mathrm{var}(C,\eta)}_{2\eta-1,2\eta}$ depending on the sign of the η-th literal in C, that is, the clique $Q^{r(\mathrm{var}(C,\eta))}_{2\eta-\mathrm{sgn}(C,\eta)}$;
- for each $1 \le \eta \le 3$ and $\xi \in \{2, 3\}$, connect $s^C_{1,\xi}$ to all vertices of both cliques the vertex $w^{\mathrm{var}(C,\eta)}_{2\eta-1,2\eta}$ is adjacent to, that is, the cliques $Q^{r(\mathrm{var}(C,\eta))}_{2\eta-1}$ and $Q^{r(\mathrm{var}(C,\eta))}_{2\eta}$.

In this manner, vertex $s^C_{1,1}$ is adjacent to three cliques Q^r_α, while $s^C_{1,2}$ and $s^C_{1,3}$, which are twins, are adjacent to six of them. Assuming that each clique Q^r_α is in a different cluster, we need to edit two connections to the cliques for vertex $s^C_{1,1}$, and five for each of vertices $s^C_{1,2}$, $s^C_{1,3}$. Checking satisfaction of the assignment is performed on the edges between $s^C_{1,1}$ and vertices from $\mathcal{W}$. The crucial observation is that:

- if at least one of the literals in the clause is satisfied, then at least one of the three vertices from $\mathcal{W}$ adjacent to $s^C_{1,1}$ is already assigned to a clique that is connected to $s^C_{1,1}$.
- if none of the literals of the clause is satisfied, then all the vertices from $\mathcal{W}$, to which $s^C_{1,1}$ is adjacent, are assigned to cliques not connected to $s^C_{1,1}$.

Hence, if the first possibility takes place, we can save one edition by not changing adjacency between $s^C_{1,1}$ and the corresponding vertex from $\mathcal{W}$. However, if the second possibility takes place, we need to change all three adjacencies, unless we want to separate $s^C_{1,1}$ from all the three adjacent cliques Q^r_α, which is too expensive.

Vertices $s^C_{1,2}$ and $s^C_{1,3}$ help us to balance the sizes of the clusters, as we may assign them to any clique that is adjacent to them. For example, if $s^C_{1,1}$ was assigned to $Q^{r(x)}_1$, then we can assign $s^C_{1,2}$ to $Q^{r(y)}_3$ and $s^C_{1,3}$ to $Q^{r(z)}_6$. The construction of vertices $\{s^C_{2,1}, s^C_{2,2}, s^C_{2,3}\}$ and $\{s^C_{3,1}, s^C_{3,2}, s^C_{3,3}\}$ follow the same rules, but the lower indices of the cliques and vertices from $\mathcal{W}$ to which the constructed vertices are adjacent, are cyclically shifted by 2 and 4, respectively. In this manner we are able to ensure the following properties: if the assignment satisfies clause C, then vertices $s^C_{\beta,\xi}$ can be assigned to the cliques so that (i) each vertex is assigned to a clique it is connected to, (ii) for each vertex we save one edition on editing adjacencies to vertices from $\mathcal{W}$, (iii) each clique with an odd lower index is assigned one vertex if the corresponding literal appears positively in C, and zero otherwise, (iv) each clique with an even lower index is assigned one vertex if the corresponding literal appears negatively in C, and zero otherwise. By property (c) of the formula Φ' we know that for the satisfying assignment all the cliques are assigned exactly the same number of vertices from $\mathcal{S}$.

This concludes the construction. We note that $|V(G)| = 6pL + \mathcal{O}(n' + m') = \mathcal{O}(\sqrt{pk})$.

We now calculate the budget k' for edge editions in the created instance. Then we argue why in case of existence of a satisfying assignment there is a set of at most k' edge editions that turns G into a $6p$-cluster graph. (The argument for the converse is deferred to the full version.) In the constructed solution all the cliques Q^r_α will be in different clusters.

To make the presentation more clear, we split this budget into few summands. Let

$$k_{\mathcal{Q}-\mathcal{Q}} = 0, \qquad k_{\mathcal{Q}-\mathcal{WS}} = (6n' + 36m')L, \qquad k^{\mathrm{all}}_{\mathcal{WS}-\mathcal{WS}} = 6p\binom{\frac{6n'+9m'}{6p}}{2},$$

$$k^{\mathrm{exist}}_{\mathcal{WS}-\mathcal{WS}} = 6n' + 27m', \qquad k^{\mathrm{save}}_{\mathcal{W}-\mathcal{W}} = 3n', \qquad k^{\mathrm{save}}_{\mathcal{W}-\mathcal{S}} = 9m',$$

and finally

$$k' = k_{\mathcal{Q}-\mathcal{Q}} + k_{\mathcal{Q}-\mathcal{WS}} + k^{\mathrm{all}}_{\mathcal{WS}-\mathcal{WS}} + k^{\mathrm{exist}}_{\mathcal{WS}-\mathcal{WS}} - 2k^{\mathrm{save}}_{\mathcal{W}-\mathcal{W}} - 2k^{\mathrm{save}}_{\mathcal{W}-\mathcal{S}}.$$

Note that, as $p \le k$, $L = \mathcal{O}(\sqrt{k/p})$ and $n', m' = \mathcal{O}(\sqrt{pk})$, we have $k' = \mathcal{O}(k)$.

The intuition behind this split is as follows. The intended solution for the p-Cluster Editing instance $(G, 6p, k')$ creates no edges between the cliques Q^r_α, each clique is contained in its own cluster, and $k_{\mathcal{Q}-\mathcal{Q}} = 0$. For each $v \in \mathcal{W} \cup \mathcal{S}$, the vertex v is assigned to a cluster with one clique adjacent to v; $k_{\mathcal{Q}-\mathcal{WS}}$ accumulates the cost of removal of other edges in $E(\mathcal{Q}, \mathcal{W} \cup \mathcal{S})$. Finally, we count the editions in $(\mathcal{W} \cup \mathcal{S}) \times (\mathcal{W} \cup \mathcal{S})$ in an indirect way. First we cut all edges of $E(\mathcal{W} \cup \mathcal{S}, \mathcal{W} \cup \mathcal{S})$ (summand $k^{\mathrm{exist}}_{\mathcal{WS}-\mathcal{WS}}$). We group the vertices of $\mathcal{W} \cup \mathcal{S}$ into clusters and add edges between vertices in each cluster; the summand $k^{\mathrm{all}}_{\mathcal{WS}-\mathcal{WS}}$ corresponds to the cost of this operation when all the clusters are of the same size (and the number of edges is minimum possible, due to the convexity of function $t \to \binom{t}{2}$). Finally, in summands $k^{\mathrm{save}}_{\mathcal{W}-\mathcal{W}}$ and $k^{\mathrm{save}}_{\mathcal{W}-\mathcal{S}}$ we count how many edges are removed and then added again in this process: $k^{\mathrm{save}}_{\mathcal{W}-\mathcal{W}}$ corresponds to saving three edges from each 6-cycle in $E(\mathcal{W}, \mathcal{W})$ and $k^{\mathrm{save}}_{\mathcal{W}-\mathcal{S}}$ corresponds to saving one edge in $E(\mathcal{W}, \mathcal{S})$ per each vertex $s^C_{\beta,\xi}$. By the described properties of clause encoding it directly follows, that a satisfying assignment can be translated into an edition set of size at most k'.

Having sketched the completeness proof, we would like to intuitively describe the difficulties that arise in the proof of soundness, i.e., that the existence of a p'-cluster graph within edition distance at most k', for $p' \le 6p$, implies that Φ' is satisfiable. If we assume that the solution behaves 'sensibly', then the minimal possible budget given for $k^{\mathrm{all}}_{\mathcal{WS}-\mathcal{WS}}$ and the properties of clause encoding already ensure that it translates to an assignment satisfying Φ'. Unfortunately, we need to argue also that the solution does not 'cheat'; the main two ways of cheating are (i) trying to merge two cliques Q^r_α, (ii) trying to separate a vertex $s^C_{\beta,\xi}$ from all the adjacent cliques. Clearly, each of these operations is locally suboptimal, but we need to guarantee that one cheat cannot lead to a lot of further savings. For example, merging two cliques Q^r_α implies that some vertices $s^C_{\beta,\xi}$ may be separated from less cliques they are adjacent to, than intended.

Usually, one copes with such problems by creating several 'layers' of the budget and ensuring that all the possible savings from any cheating cannot compensate even cost of one cheat. In our setting, making cliques Q^r_α much bigger would solve the problem. However, then we would need to increase the budget as well and the reduction would yield a weaker lower bound. Instead, we have to provide an extremely careful bookkeeping analysis of the possible shape of the solution in order to show that, indeed, the possible gains from cheating cannot amortise the costs.

5 Conclusion and open questions

We gave an algorithm that solves p-Cluster Editing in time $\mathcal{O}(2^{\mathcal{O}(\sqrt{pk})} + n + m)$ and complemented it by a multivariate lower bound, which shows that the running time of our algorithm is asymptotically tight for all p sublinear in k.

In our multivariate lower bound it is crucial that the cliques and clusters are arranged in groups of six. However, the drawback of this construction is that Theorem 2 settles the time complexity of p-Cluster Editing problem only for $p \ge 6$ (Corollary 4). It does not seem unreasonable that, for example, the 2-Cluster Editing problem, already NP-complete [34], may have enough structure to allow an algorithm with running time $\mathcal{O}(2^{o(\sqrt{k})} + n + m)$. Can we construct such an algorithm or refute its existence under ETH?

Secondly, we would like to point out an interesting link between the subexponential parameterized complexity of the problem and its approximability. When the number of clusters drops from linear to sublinear in k, we obtain a phase transition in parameterized complexity from exponential to subexponential. As far as approximation is concerned, we know that bounding the number of clusters by a constant allows us to construct a PTAS [24], whereas the general problem is APX-hard [13]. The mutual drop of the parameterized complexity of a problem — from exponential to subexponential — and of approximability — from APX-hardness to admitting a PTAS — can be also observed for many hard problems when the input is constrained by additional topological bounds, for instance excluding a fixed pattern as a minor [17, 18, 22]. It is therefore an interesting question, whether p-Cluster Editing also admits a PTAS when the number of clusters is bounded by a non-constant, yet sublinear function of k, for instance $p = \sqrt{k}$.

Acknowledgements

We thank Christian Komusiewicz for pointing us to the recent results on Cluster Editing [7, 31] and his thesis [30]. Moreover, we thank Pål Grønås Drange, M. S. Ramanujan and Saket Saurabh for helpful discussions.

References

1 Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: ranking and clustering. In *Proc. of STOC'05*, pages 684–693. ACM, 2005.

2 Noga Alon, Daniel Lokshtanov, and Saket Saurabh. Fast FAST. In *Proc. of ICALP'09*, volume 5555 of *Lecture Notes in Comput. Sci.*, pages 49–58. Springer, 2009.

3 Noga Alon, Konstantin Makarychev, Yury Makarychev, and Assaf Naor. Quadratic forms on graphs. In *Proc. of STOC'05*, pages 486–493. ACM, 2005.

4 Sanjeev Arora, Eli Berger, Elad Hazan, Guy Kindler, and Muli Safra. On non-approximability for quadratic programs. In *Proc. of FOCS'05*, pages 206–215. IEEE Computer Society, 2005.

5 Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56:89–113, 2004.

6 Amir Ben-Dor, Ron Shamir, and Zohar Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6(3/4):281–297, 1999.

7 Sebastian Böcker. A golden ratio parameterized algorithm for cluster editing. In *Proc. of IWOCA'11*, pages 85–95, 2011.

8 Sebastian Böcker, Sebastian Briesemeister, Quang Bao Anh Bui, and Anke Truß. A fixed-parameter approach for weighted cluster editing. In *Proc. of APBC'08*, volume 6 of *Advances in Bioinformatics and Computational Biology*, pages 211–220, 2008.

9 Sebastian Böcker, Sebastian Briesemeister, and Gunnar W. Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 60(2):316–334, 2011.

10 Sebastian Böcker and Peter Damaschke. Even faster parameterized cluster deletion and cluster editing. *Inf. Process. Lett.*, 111(14):717–721, 2011.

11 Hans L. Bodlaender, Michael R. Fellows, Pinar Heggernes, Federico Mancini, Charis Papadopoulos, and Frances A. Rosamond. Clustering with partial information. *Theor. Comput. Sci.*, 411(7-9):1202–1211, 2010.

12 Yixin Cao and Jianer Chen. Cluster editing: Kernelization based on edge cuts. In *Proc. of IPEC'10*, volume 6478 of *Lecture Notes in Computer Science*, pages 60–71. Springer, 2010.

13 Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. In *Proc. of FOCS'03*, pages 524–533. IEEE Computer Society, 2003.

14 Moses Charikar and Anthony Wirth. Maximizing quadratic programs: Extending Grothendieck's inequality. In *Proc. of FOCS'04*, pages 54–60. IEEE Computer Society, 2004.

15 Jianer Chen and Jie Meng. A $2k$ kernel for the cluster editing problem. *Journal of Computer and System Sciences*, 78(1):211 – 220, 2012.

16 Peter Damaschke. Fixed-parameter enumerability of cluster editing and related problems. *Theory Comput. Syst.*, 46(2):261–283, 2010.

17 Erik D. Demaine, Fedor V. Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on graphs of bounded genus and H-minor-free graphs. *Journal of the ACM*, 52(6):866–893, 2005.

18 Erik D. Demaine and Mohammadtaghi Hajiaghayi. Bidimensionality: New connections between FPT algorithms and PTASs. In *Proc. of SODA'05*, pages 590–601, 2005.

19 R. G. Downey and M. R. Fellows. *Parameterized complexity.* Springer-Verlag, New York, 1999.

20 Michael R. Fellows, Jiong Guo, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. Graph-based data clustering with overlaps. *Discrete Optimization*, 8(1):2–17, 2011.

21 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2006.

22 Fedor V. Fomin, Daniel Lokshtanov, Venkatesh Raman, and Saket Saurabh. Bidimensionality and EPTAS. In *Proc. of SODA'11*, pages 748–759. SIAM, 2011.

23 Fedor V. Fomin and Yngve Vilanger. Subexponential parameterized algorithm for minimum fill-in. In *Proc. of SODA'12*, pages 1737–1746. SIAM, 2012.

24 Ioannis Giotis and Venkatesan Guruswami. Correlation clustering with a fixed number of clusters. In *Proc. of SODA'06*, pages 1167–1176. ACM Press, 2006.

25 Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Graph-modeled data clustering: Exact algorithms for clique generation. *Theory Comput. Syst.*, 38(4):373–392, 2005.

26 Jiong Guo. A more effective linear kernelization for cluster editing. *Theor. Comput. Sci.*, 410(8-10):718–726, 2009.

27 Jiong Guo, Iyad A. Kanj, Christian Komusiewicz, and Johannes Uhlmann. Editing graphs into disjoint unions of dense clusters. *Algorithmica*, 61(4):949–970, 2011.

28 Jiong Guo, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. A more relaxed model for graph-based data clustering: s-plex cluster editing. *SIAM J. Discrete Math.*, 24(4):1662–1683, 2010.

29 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

30 Christian Komusiewicz. *Parameterized Algorithmics for Network Analysis: Clustering & Querying.* PhD thesis, Technische Universität Berlin, 2011. Available at `http://fpt.akt.tu-berlin.de/publications/diss-komusiewicz.pdf`.

31 Christian Komusiewicz and Johannes Uhlmann. Alternative parameterizations for cluster editing. In *Proc. of SOFSEM'11*, volume 6543 of *Lecture Notes in Computer Science*, pages 344–355. Springer, 2011.

32 Dániel Marx. What's next? future directions in parameterized complexity. In Hans L. Bodlaender, Rod Downey, Fedor V. Fomin, and Dániel Marx, editors, *The Multivariate Algorithmic Revolution and Beyond*, volume 7370 of *Lecture Notes in Computer Science*, pages 469–496. Springer, 2012.

33 Fábio Protti, Maise Dantas da Silva, and Jayme Luiz Szwarcfiter. Applying modular decomposition to parameterized cluster editing problems. *Theory Comput. Syst.*, 44(1):91–104, 2009.

34 Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004.

Bounded-width QBF is PSPACE-complete

Albert Atserias[1] **and Sergi Oliva**[2]

1 **Universitat Politècnica de Catalunya**
Barcelona, Spain
atserias@lsi.upc.edu

2 **Universitat Politècnica de Catalunya**
Barcelona, Spain
oliva@lsi.upc.edu

Abstract

Tree-width is a well-studied parameter of structures that measures their similarity to a tree. Many important NP-complete problems, such as Boolean satisfiability (SAT), are tractable on bounded tree-width instances. In this paper we focus on the canonical PSPACE-complete problem QBF, the fully-quantified version of SAT. It was shown by Pan and Vardi [LICS 2006] that this problem is PSPACE-complete even for formulas whose tree-width grows extremely slowly. Vardi also posed the question of whether the problem is tractable when restricted to instances of bounded tree-width. We answer this question by showing that QBF on instances with constant tree-width is PSPACE-complete.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Tree-width, QBF, PSPACE-complete

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.44

1 Introduction

Tree-width is a well-known parameter that measures how close a structure is to being a tree. Many NP-complete problems have polynomial-time algorithms on inputs of bounded tree-width. In particular, the Boolean satisfiability problem can be solved in polynomial time when the constraint graph of the input cnf-formula has bounded tree-width (cf. [3], [4]).

A natural question suggested by this result is whether QBF, the problem of determining if a fully-quantified cnf-formula is true or false, can also be solved in polynomial time when restricted to formulas whose cnf has bounded tree-width. In [1], Chen concludes that the problem stays tractable if the number of alternations, as well as the tree-width, is bounded. On the negative side, Gottlob, Greco and Scarcello [6] proved that the problem stays PSPACE-complete when the number of alternations is unbounded even if the constraint graph of the cnf-formula has logarithmic tree-width (and indeed, its *incidence* graph is even a tree). By different methods, and improving upon [6], Pan and Vardi [8] show that, unless P = NP, the dependence of the running time of Chen's algorithm on the number of alternations must be non-elementary, and that the QBF problem restricted to instances of tree-width $\log^*$ in the size of the input is PSPACE-complete. All these negative results hold also for path-width, which is a parameter that measures the similarity to a path and is in general smaller than tree-width. However, they leave open whether QBF is tractable for instances whose constraint graph has constant path-width, or even constant tree-width.

In this paper, we resolve this question by showing that, even for inputs of constant path-width, QBF is PSPACE-complete. Our construction builds on the techniques from [8] with two essential differences. The first difference is that instead of reducing from the so-called *tiling-game* and producing a quantified Boolean formula of $\log^*$-smaller path-width, our reduction starts at QBF itself and produces a quantified Boolean formula whose path-width is *only* logarithmically smaller. Although this looks like backward progress, it leaves us in a position where iterating the reduction makes sense. However, in order to do so, we need to analyze which properties of the output of the reduction can be exploited by the next iteration. Here comes the second main difference: we observe that the output of the reduction has not only smaller path-width, but also smaller *window-size*, which means that any two occurences of the same variable appear close to each other in some ordering of the clauses. We call such formulas *n-leveled*, where n is a bound related to the window-size. Our main lemma exploits this structural restriction in a technical way to show that the QBF problem for n-leveled formulas reduces to the QBF problem for $O(\log n)$-leveled formulas. Iterating this reduction until we reach $O(1)$-leveled formulas yields the result.

Comparison to previous work. A few more words on the differences between our methods and those in [8] and [6] are in order. The technical tool from [8] that is used to achieve n-variable formulas of $O(\log^* n)$ path-width builds on the tools from [7] and [5] that were used for showing non-elementary lower-bounds for some problems related to second-order logic. These tools are based on an encoding of natural numbers that allows the comparison of two n-bit numbers by means of an extremely smaller formula; one of size $O(\log^* n)$. It is interesting that, by explicitely avoiding this technique, our iteration-based methods take us further: beyond $O(\log^* n)$ path-width down to constant path-width. For the same reason our proof can stay purely at the level of propositional logic without the need to resort to second-order logic. Along the same lines, our method also shows that the QBF problem for n-variable formulas of constant path-width and $O(\log^* n)$ quantifier alternations is NP-hard (and Σ_iP-hard for any $i \geq 1$), while the methods from [8] could only show this for $O(\log^* n)$ path-width and $O(\log^* n)$ alternations. It is worth noting that, in view of the results in [1], these hardness results are tight up to the hidden constants in the asymptotic notation.

Structural restrictions on the generalization of QBF to unbounded domains, sometimes called QCSP, have also been studied. Gottlob et al. [6] proved that QCSP restricted to trees is already PSPACE-complete. Their hardness result for QBF of logarithmic tree-width follows from this by *booleanization*. They also identify some new tractable fragments, and some other hardness conditions. Finally, Chen and Dalmau [2] introduced a general framework for studying structural restrictions on QCSP, and characterized the restrictions that make the problem tractable under complexity-theoretic assumptions.

Paper organization. The paper is organized as follows. In section 2, we introduce the basic definitions. In section 3, we formalize the concept of leveled-qbf and state and prove the main lemma. Finally, in section 4, we present the main theorem of the paper, which shows how to iterate the lemma to obtain the desired result.

2 Preliminaries

We write $[n] := \{1, \ldots, n\}$ and $|n| := \lceil \log(n+1) \rceil$. All logarithms are base 2. Note that $|n|$ is the length of the binary encoding of n. We define $\log^{(0)} n := n$ and $\log^{(i)} n := \log(\log^{(i-1)} n)$ for $i > 0$. Also, we use $\log^* n$ as the least integer i such that $\log^{(i)} n \leq 1$.

The negation of a propositional variable x is denoted by $\overline{x}$. We also use the notation $x^{(1)}$ and $x^{(0)}$ to denote x and $\overline{x}$, respectively. Note that the notation is chosen so that $x^{(a)}$ is made *true* by the assignment $x = a$. The *underlying variable* of $x^{(a)}$ is x, and its *sign* is a. A *literal* is a variable or the negation of a variable. A *clause* is a sequence of literals. A *cnf-formula* is a sequence of clauses. The *size* of a clause is its length as a sequence, and the *size* of a cnf-formula is the sum of the sizes of its clauses. For example,

$$\phi = ((x_1, \overline{x_2}), (x_2, \overline{x_3}, x_4), (\overline{x_4})) \tag{1}$$

is a cnf-formula of size 6 made of three clauses of sizes 2, 3, and 1, respectively. If ϕ is a cnf-formula of size s, we write $\ell_1(\phi), \ldots, \ell_s(\phi)$ for the s literals of ϕ in the left-to-right order in which they appear in ϕ. For example, in (1) we have $\ell_4(\phi) = \overline{x_3}$. When ϕ is clear from the context we write ℓ_i instead of $\ell_i(\phi)$.

Let ϕ be a cnf-formula. A *path-decomposition* of ϕ is a sequence $A_1, \ldots, A_m$ of subsets of variables that satisfies the following properties:

1. for every clause C of ϕ there is some $i \in [m]$ such that all the variables of C are in A_i,
2. for every $i, j, k \in [m]$ such that $i \leq j \leq k$ we have $A_i \cap A_k \subseteq A_j$.

The *width* of the path-decomposition is the maximum $|A_i|$ minus one. The *path-width* of ϕ is the smallest width of all its path-decompositions. The path-width is bounded by the *tree-width* of the constraint graph of the cnf-formula, defined in the usual way (cf. [4]).

A *qbf* is a quantified Boolean formula of the form

$$\phi = Q_1 x_1 \cdots Q_q x_q (\phi'), \tag{2}$$

where $x_1, \ldots, x_q$ are propositional variables, the *matrix* ϕ' is a cnf-formula, and Q_i is either $\forall$ or $\exists$ for every $i \in \{1, \ldots, q\}$. The *size* of a qbf as in (2) is defined as the size of its matrix ϕ'. The path-width of a qbf is the path-width of its matrix.

3 Leveled Formulas

In this section we state and prove the main lemma. This lemma is a reduction from n-leveled qbfs to $O(\log n)$-leveled qbfs, which is progress in our iterative argument. Before stating the lemma, we formalize the concept of leveled-qbf.

Let n be a positive integer. An *n-leveled cnf-formula* is a cnf-formula ϕ in which its sequence of clauses is partitioned into *blocks* $B_1, \ldots, B_\ell$, where each block is a consecutive subsequence of clauses of ϕ, and its set of variables is partitioned into the same number of *groups* $G_1, \ldots, G_\ell$, each containing at most n variables, and such that for every $j \in \{1, \ldots, \ell - 1\}$ we have that every clause C in B_j has all its variables in $G_j \cup G_{j+1}$, and every clause C in B_ℓ has all its variables in G_ℓ. An *n-leveled qbf* is a quantified Boolean formula whose matrix is an n-leveled cnf-formula.

Observe that every qbf with n variables is an n-leveled qbf: put all clauses in a single block and all variables in a single group. However, when the sizes of the groups are limited, we get a nice structure:

▶ **Lemma 1.** *Let n be a positive integer. Every n-leveled qbf has path-width at most $2n - 1$.*

Proof. Let ϕ be an n-leveled qbf with groups $G_1, \ldots, G_\ell$. It is straightforward to check from the definition of leveled formula that the sequence $A_1, \ldots, A_\ell$ defined by $A_j = G_j \cup G_{j+1}$ for $j \in \{1, \ldots, \ell - 1\}$ and $A_\ell = G_\ell$ forms a path-decomposition of the cnf-formula in the matrix of ϕ. Since each G_j has cardinality at most n, the claim follows. ◀

Now, we can formalize the statement of the main lemma.

▶ **Lemma 2.** *There exist $c, d \geq 1$ and a polynomial-time algorithm that, for every $n, s \geq 1$, given an n-leveled qbf ϕ of size s, computes a $c \cdot |n|$-leveled qbf ψ of size $d \cdot s \cdot |n|$ such that $\phi \leftrightarrow \psi$.*

We devote the rest of the section to the proof of this lemma. In order to improve the readability of Boolean formulas, we use $+$ for disjunction and $\cdot$ for conjunction.

3.1 Definition of θ

Let ϕ be a n-leveled qbf as in (2) whose matrix ϕ' is an n-leveled cnf-formula of size s with groups $G_1, \ldots, G_\ell$ and blocks $B_1, \ldots, B_\ell$. As a first step towards building ψ we define an intermediate formula θ. The formula θ contains variables $\tau_1, \ldots, \tau_s$, one for each literal in ϕ', and is defined as

$$\theta := Q_1\boldsymbol{\tau}_1 \cdots Q_q\boldsymbol{\tau}_q(\textsc{ncons}_\forall + (\textsc{cons}_\exists \cdot \textsc{sat}))$$

where

1. each $\boldsymbol{\tau}_j$, for $j \in [q]$, is the tuple of τ-variables corresponding to all the occurrences of the variable x_j in ϕ',
2. $\textsc{cons}_Q$, for $Q \in \{\forall, \exists\}$, is a qbf to be defined later that is satisfied by an assignment to $\tau_1, \ldots, \tau_s$ if and only if all the variables from the same $\boldsymbol{\tau}_j$ with $Q_j = Q$ are given the same truth value,
3. $\textsc{ncons}_Q$ for $Q \in \{\forall, \exists\}$ is a qbf that is equivalent to the negation of $\textsc{cons}_Q$,
4. $\textsc{sat}$ is a qbf to be defined later that is satisfied by an assignment to $\tau_1, \ldots, \tau_s$ if and only if every clause of ϕ' contains at least one literal $\ell_k = x^{(a)}$ such that τ_k is given value a.

This information about the constituents of θ is enough to prove the following claim.

▶ Claim 1. $\phi \leftrightarrow \theta$

Proof. We need to prove both implications. In both cases we use a game in which two players, the existential player and the universal player, take rounds following the order of quantification of the formula to choose values for the variables quantified their way. The aim of the existential player is to show that the matrix of the formula can be made true while the aim of the universal player is to show him wrong.

In the following, for $j \in [q]$, we say that an assignment to the variables of $\boldsymbol{\tau}_j$ is *consistent* if they are given the same truth value, say $a \in \{0, 1\}$. In case the assignment is consistent, we say that a is the *corresponding assignment* for the variable x_j. Conversely, if a is an assignment to the variable x_j, the *corresponding consistent assignment* for the tuple $\boldsymbol{\tau}_j$ is the assignment that sets each variable in $\boldsymbol{\tau}_j$ to a. If an assignment to $\boldsymbol{\tau}_j$ is not consistent we call it inconsistent.

$(\rightarrow)$: Assume ϕ is true and let α be a winning strategy for the existential player in ϕ. We build another strategy β that guarantees him a win in θ. The construction of β will be based on the observation that, in the course of the game on θ, if the assignment given by the universal player to some $\boldsymbol{\tau}_j$ with $Q_j = \forall$ is inconsistent, then $\textsc{ncons}_\forall$ is true irrespective of all other variables, and hence the matrix of θ is true. With this observation in hand, the strategy β is defined as follows: at round j with $Q_j = \exists$, if all $\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_{j-1}$ have been given consistent assignments up to this point and $a_1, \ldots, a_{j-1} \in \{0, 1\}$ are the corresponding assignments to the variables $x_1, \ldots, x_{j-1}$, let a_j be the assignment given to x_j by the strategy α in this position of the game on ϕ, and let the existential player assign value a_j to every variable in $\boldsymbol{\tau}_j$. If on the other hand some $\boldsymbol{\tau}_k$ with $k < j$ has been given an inconsistent assignment, let the existential player assign an arbitrary value (say 0) to every variable in $\boldsymbol{\tau}_j$. Using the observation above and the assumption that α is a winning strategy, it is not hard to see that β is a winning strategy.

$(\leftarrow)$: Assume θ is true and let β be a winning strategy for the existential player in θ. We build a strategy α for the existential player in ϕ. In this case the construction of α will be based on the observation that, in the course of the game on θ, as long as the universal player assigns consistent values to every $\boldsymbol{\tau}_j$ with $Q_j = \forall$, the assignment given by β to each new $\boldsymbol{\tau}_j$ with $Q_j = \exists$ must be consistent. To see this note that, if not, the universal player would have the option of staying consistent all the way until the end of the game in which case both $\textsc{ncons}_\forall$ and $\textsc{cons}_\exists$ would become false, thus making the matrix of θ false. With this observation in hand, the strategy α is defined as follows: at round j with $Q_j = \exists$, let $a_1, \dots, a_{j-1} \in \{0,1\}$ be the assignment given to $x_1, \dots, x_{j-1}$ up to this point, let $\mathbf{a}_1, \dots, \mathbf{a}_{j-1}$ be the corresponding consistent assignments for $\boldsymbol{\tau}_1, \dots, \boldsymbol{\tau}_{j-1}$, and let $\mathbf{a}_j$ be the assignment given by β to $\boldsymbol{\tau}_j$ in this position of the game on θ. By the observation above, since each $\mathbf{a}_k$ with $k < j$ and $Q_k = \forall$ is consistent by definition and each $\mathbf{a}_k$ with $k < j$ and $Q_j = \exists$ has been assigned according to the strategy β, the assignment $\mathbf{a}_j$ must also be consistent. Thus the existential player can set x_j to its corresponding value a_j and continue with the game.

We need to show that α is a winning strategy for the existential player on ϕ. First, if the existential player plays according to α, then the final assignment $a_1, \dots, a_q$ that is reached in the game on ϕ is such that the corresponding assignment $\mathbf{a}_1, \dots, \mathbf{a}_q$ in the game on ψ satisfies the matrix of θ. Since each $\mathbf{a}_j$ is consistent this means that $\textsc{sat}$ must be made true by $\mathbf{a}_1, \dots, \mathbf{a}_q$, thus the matrix of ϕ is made true by $a_1, \dots, a_q$. This shows that the existential player wins. ◀

Now, we show how to construct the qbf-formulas $\textsc{sat}$, $\textsc{cons}_\exists$ and $\textsc{ncons}_\forall$. These formulas have the τ-variables as free variables and a new set of quantified variables for each literal in ϕ'. Recall that the τ-variables assign a truth value to each variable-ocurrence in ϕ'. The formula $\textsc{sat}$ will verify that these assignments satisfy all clauses of ϕ', the formula $\textsc{cons}_\exists$ will verify that each existentially quantified variable is assigned consistently, and the formula $\textsc{ncons}_\forall$ will verify that at least one universally quantified variable is assigned inconsistently.

3.2 Definition of SAT

For every $i \in [s+1]$, we have variables μ_i and ν_i. By scanning its literals left-to-right, the formula checks that every clause of ϕ' contains at least one literal $\ell_k = x^{(a)}$ such that τ_k is given value a. To do so, μ_i and ν_i indicate the status of this process when exactly $i-1$ literals have been scanned. The intended meaning of the variables is the following:

- μ_i = "just before scanning ℓ_i, the clauses already completely scanned are satisfied, and the current clause is not satisfied yet".
- ν_i = "just before scanning ℓ_i, the clauses already completely scanned are satisfied, and the current clause is satisfied as well".

Note that ℓ_{s+1} is not a literal. Therefore, "just before scanning ℓ_{s+1}" means "just after scanning the last literal" in this case. Also, variables μ_1 and ν_1 are initialized to true and false, respectively. We want to make sure that at position $i = s+1$, i.e. after scanning the last literal, μ_{s+1} is true. Later, we will axiomatize the transition between positions i and $i+1$. That will define μ_{i+1} and ν_{i+1} depending on μ_i, ν_i and ℓ_i according to its intended meaning. We will axiomatize this into the formula $\textsc{sat}(i)$. Then, $\textsc{sat}$ is defined as

$$\textsc{sat} := \exists \boldsymbol{\mu} \exists \boldsymbol{\nu} \left(\mu_1 \cdot \overline{\nu_1} \cdot \prod_{i=1}^{s} \textsc{sat}(i) \cdot \mu_{s+1} \right)$$

where $\boldsymbol{\mu} = (\mu_1, \dots, \mu_{s+1})$ and $\boldsymbol{\nu} = (\nu_1, \dots, \nu_{s+1})$.

Next, we formalize $\textsc{sat}(i)$. For every $i \in [s]$, let $a_i \in \{0,1\}$ denote the sign of ℓ_i, the i-th literal of ϕ', and let $k_i \in \{0,1\}$ be the predicate that indicates whether ℓ_i is the last in literal its clause. Then, $\textsc{sat}(i)$ is the conjunction of the following formulas:

$$\mu_{i+1} \leftrightarrow \overline{k_i}\,\mu_i\,a_i\,\overline{\tau_i} \;+\; \overline{k_i}\,\mu_i\,\overline{a_i}\,\tau_i \;+\; k_i\,\mu_i\,a_i\,\tau_i \;+\; k_i\,\mu_i\,\overline{a_i}\,\overline{\tau_i} \;+\; k_i\,\nu_i,$$
$$\nu_{i+1} \leftrightarrow \overline{k_i}\,\mu_i\,a_i\,\tau_i \;+\; \overline{k_i}\,\mu_i\,\overline{a_i}\,\overline{\tau_i} \;+\; \overline{k_i}\,\nu_i.$$

In words, the axiomatization states that μ_{i+1} holds in one of three cases: 1) if ℓ_i is the last literal in its clause and the clause has been satisfied by a previous literal ($k_i\nu_i$), or 2) if ℓ_i is the last literal in its clause, this clause is not yet satisfied by a previous literal, but the truth assignment satisfies the current one ($k_i\mu_i a_i\tau_i + k_i\mu_i\overline{a_i}\overline{\tau_i}$), or 3) if ℓ_i is not the last literal in its clause, this clause is not yet satisfied by a previous literal, and the truth assignment does not satisfy the current one either ($\overline{k_i}\mu_i a_i\overline{\tau_i} + \overline{k_i}\mu_i\overline{a_i}\tau_i$). The axiomatization of ν_{i+1} is similar.

Note that these two formulas can be written in cnf by writing $\leftrightarrow$ in terms of conjunctions and disjunctions and by distributing disjunctions over conjunctions. We call *i-link* a clause that contains variables only with indices i and $i+1$. Observe for later use that all clauses in the resulting cnf-formulas for $\textsc{sat}(i)$ are i-links. Also, the size of $\textsc{sat}$ written in cnf is $c \cdot s$ for some constant $c \geq 1$.

3.3 Definition of CONS$_\exists$

The construction of $\textsc{cons}_\exists$ is a bit more complicated. It uses universally quantified variables $\{\pi_1, \dots, \pi_s\}$ as pointers to the literals of ϕ', in one-to-one correspondance with $\{\tau_1, \dots, \tau_s\}$. We say that pointer π_i points to literal ℓ_i. If x is the underlying variable of ℓ_i, we say that π_i points to x. Pointers that are set to true are called *activated*. We say that a pointer has been scanned if its pointed literal has been scanned. The formula checks the following: whenever exactly two pointers are activated and they point to occurrences of the same existentially quantified variable, then the truth values assigned to the pointed literals are consistent. To refer to a variable, we do not encode its identifier directly. Instead, we encode the parity of its group and its index inside this group. This is enough information to distinguish between different variables in the same or neighbouring blocks. This fact is key to our argument and will be proved later in Claim 2. The point is that this compact encoding uses only $|n|+1$ bits per occurrence, where n is the number of variables per group, which may be much smaller than the total number of variables.

The formula uses the following variables for $i \in [s+1]$:

- ξ_i = "just before scanning ℓ_i, all the activated pointers already scanned point to an existentially quantified variable".
- $\sigma_{i,k}$ = "just before scanning ℓ_i, exactly k activated pointers have been scanned".
- $\chi_{i,k}$ = "just before scanning ℓ_i, exactly one activated pointer has been scanned and there have been k changes of block between the pointed literal and position i, or exactly two have been scanned and there have been exactly k changes of block between the pointed literals".
- ω_i = "just before scanning ℓ_i, exactly one activated pointer has been scanned and the parity of the group of the pointed variable is equal to the parity of the block of the clause of the pointed literal, or exactly two have been scanned and the groups of the pointed variables are the same".
- κ_i = "just before scanning ℓ_i, exactly one activated pointer has been scanned and the τ-variable at the pointed position is true, or exactly two have been scanned and the truth values of the τ-variables at the pointed positions are the same".

- $\lambda_{i,b}$ = "just before scanning ℓ_i, exactly one activated pointer has been scanned and the b-th bit of the index of the pointed variable in its group is 1, or exactly two have been scanned and the b-th bit of the indices of the pointed variables in their respective groups are the same".

The variables at step $i+1$ will be axiomatized in terms of the variables at step i and ℓ_i in the formula $\textsc{cons}_\exists(i)$. The formula $\textsc{cons}_\exists$ also requires a consistency condition for all possible combinations of activated pointers. For a given combination of these pointers, the consistency condition holds if: either there is a problem with the pointers (there are not exactly two pointers activated or one is not pointing to an existentially quantified variable), or the pointed variables are not comparable (are not of the same group or do not have the same index in the group) or, they are comparable and both receive the same truth value. This consistency condition will be encoded in the formula $\textsc{cons}_\exists^{\mathrm{acc}}$. Also, the value of the variables at position $i=1$ will be encoded in the formula $\textsc{cons}_\exists^{\mathrm{ini}}$. Now,

$$\textsc{cons}_\exists := \forall\boldsymbol{\pi}\exists\boldsymbol{\xi}\exists\boldsymbol{\sigma}\exists\boldsymbol{\chi}\exists\boldsymbol{\omega}\exists\boldsymbol{\kappa}\exists\boldsymbol{\lambda}\left(\textsc{cons}_\exists^{\mathrm{ini}}\cdot\prod_{i=1}^{s}\textsc{cons}_\exists(i)\cdot\textsc{cons}_\exists^{\mathrm{acc}}\right)$$

where $\boldsymbol{\pi} = (\pi_i \,|\, 1 \le i \le s)$, $\boldsymbol{\xi} = (\xi_i \,|\, 1 \le i \le s+1)$, $\boldsymbol{\sigma} = (\sigma_{i,k} \,|\, 1 \le i \le s+1, 0 \le k \le 2)$, $\boldsymbol{\chi} = (\chi_{i,k} \,|\, 1 \le i \le s+1, 0 \le k \le 1)$, $\boldsymbol{\omega} = (\omega_i \,|\, 1 \le i \le s+1)$, $\boldsymbol{\kappa} = (\kappa_i \,|\, 1 \le i \le s+1)$ and $\boldsymbol{\lambda} = (\lambda_{i,b} \,|\, 1 \le i \le s+1, 1 \le b \le |n|)$.

Next we axiomatize the introduced variables, but before that we need to introduce some notation.

Let $g_i \in [\ell]$ be the group-number of the variable underlying literal ℓ_i, let $n_i \in [|G_{g_i}|]$ be the index of this variable within G_{g_i}, and recall $a_i \in \{0,1\}$ denotes the sign of ℓ_i. For every $i \in [s]$, let $h_i \in \{0,1\}$ be the predicate that indicates whether the i-th literal ℓ_i is the last in its block or not (recall that the blocks are subsequences of consecutive clauses that partition the sequence of clauses), and recall that $k_i \in \{0,1\}$ is the predicate that indicates whether the i-th literal ℓ_i is the last in its clause or not. Next we encode the quantification of ϕ in a way that the type of quantification of each variable can be recovered from each of its occurrences: for every $i \in [s]$, let $q_i \in \{0,1\}$ be the predicate that indicates whether the variable that underlies the i-th literal ℓ_i is universally or existentially quantified in ϕ.

Finally, observe that the definition of leveled formula implies that if $b_i \in [\ell]$ is the number of the block that contains the clause to which the i-th literal belongs, then the group-number g_i is either b_i or b_i+1 whenever $1 \le b_i \le \ell-1$, and is equal to ℓ if $b_i = \ell$. Accordingly, let $e_i \in \{0,1\}$ be such that $g_i = b_i - e_i + 1$ for every $i \in [s]$. In other words, e_i indicates whether the parities of g_i and b_i agree or not.

The following claim shows that, although the number ℓ of groups is in general unbounded, a constant number of bits of information are enough to tell if the underlying variables of two literals belong to the same group:

▶ Claim 2. Let i, j be such that $1 \le i < j \le s$. Then, the underlying variables of ℓ_i and ℓ_j belong to the same group if and only if one of the following conditions holds:

1. $e_i = e_j$ and $b_i = b_j$, or
2. $e_i = 0$, $e_j = 1$, and $b_i = b_j - 1$.

Proof. For the only if side, we have $g_i = g_j$. Then, $b_i - e_i = b_j - e_j$ and also b_i is either b_j or $b_j - 1$. If $b_i = b_j$, then $e_i = e_j$. If $b_i = b_j - 1$, then necessarily $e_i = 0$ and $e_j = 1$.

For the if side, in the first case, $g_i = b_i - e_i + 1 = b_j - e_j + 1 = g_j$. In the second case, $g_i = b_i - e_i + 1 = b_j - 1 + 1 = b_j - e_j + 1 = g_j$. Therefore, $g_i = g_j$. ◀

Using this claim, we axiomatize $\textsc{cons}_\exists(i)$ as the conjunction of the following formulas:

$$\begin{aligned}
\xi_{i+1} &\leftrightarrow \overline{\pi_i}\,\xi_i \;+\; \pi_i\,\xi_i\,q_i \\
\sigma_{i+1,0} &\leftrightarrow \sigma_{i,0}\,\overline{\pi_i} \\
\sigma_{i+1,1} &\leftrightarrow \sigma_{i,0}\,\pi_i \;+\; \sigma_{i,1}\,\overline{\pi_i} \\
\sigma_{i+1,2} &\leftrightarrow \sigma_{i,1}\,\pi_i \;+\; \sigma_{i,2}\,\overline{\pi_i} \\
\chi_{i+1,0} &\leftrightarrow \sigma_{i,0}\,\pi_i\,\overline{h_i} \;+\; \sigma_{i,1}\,\overline{\pi_i}\,\chi_{i,0}\,\overline{h_i} \;+\; \sigma_{i,1}\,\pi_i\,\chi_{i,0} \;+\; \sigma_{i,2}\,\chi_{i,0} \\
\chi_{i+1,1} &\leftrightarrow \sigma_{i,0}\,\pi_i\,h_i \;+\; \sigma_{i,1}\,\overline{\pi_i}\,\chi_{i,0}\,h_i \;+\; \sigma_{i,1}\,\overline{\pi_i}\,\chi_{i,1}\,\overline{h_i} \;+\; \sigma_{i,1}\,\pi_i\,\chi_{i,1} \;+\; \sigma_{i,2}\,\chi_{i,1} \\
\omega_{i+1} &\leftrightarrow \sigma_{i,0}\,\pi_i\,e_i \;+\; \sigma_{i,1}\,\overline{\pi_i}\,\omega_i \;+\; \sigma_{i,1}\,\pi_i\,(\chi_{i,0}\,\omega_i\,e_i \;+\; \chi_{i,0}\,\overline{\omega_i}\,\overline{e_i} \;+\; \chi_{i,1}\,\overline{\omega_i}\,e_i) \;+\; \sigma_{i,2}\,\omega_i \\
\kappa_{i+1} &\leftrightarrow \sigma_{i,0}\,\pi_i\,\tau_i \;+\; \sigma_{i,1}\,\overline{\pi_i}\,\kappa_i \;+\; \sigma_{i,1}\,\pi_i\,\kappa_i\,\tau_i \;+\; \sigma_{i,1}\,\pi_i\,\overline{\kappa_i}\,\overline{\tau_i} \;+\; \sigma_{i,2}\,\kappa_i
\end{aligned}$$

and, for all $b \in [|n|]$,

$$\lambda_{i+1,b} \leftrightarrow \sigma_{i,0}\,\pi_i\,n_{i,b} \;+\; \sigma_{i,1}\,\overline{\pi_i}\,\lambda_{i,b} \;+\; \sigma_{i,1}\,\pi_i\,\lambda_{i,b}\,n_{i,b} \;+\; \sigma_{i,1}\,\pi_i\,\overline{\lambda_{i,b}}\,\overline{n_{i,b}} \;+\; \sigma_{i,2}\,\lambda_{i,b}$$

where $n_{i,b}$ is the b-th bit of the binary encoding of n_i.

Also, we define $\textsc{cons}_\exists^{\mathrm{ini}}$ as the conjunction of the following unit clauses:

$$\xi_1, \sigma_{1,0}, \overline{\sigma_{1,1}}, \overline{\sigma_{1,2}}, \overline{\chi_{1,0}}, \overline{\chi_{1,1}}, \overline{\omega_1}, \overline{\kappa_1}, \overline{\lambda_{1,1}}, \dots, \overline{\lambda_{1,|n|}}.$$

Furthermore, we define $\textsc{cons}_\exists^{\mathrm{acc}}$ as the following clause:

$$\overline{\xi_{s+1}} + \overline{\sigma_{s+1,2}} + \overline{\omega_{s+1}} + \sum_{b=1}^{|n|} \overline{\lambda_{s+1,b}} + \kappa_{s+1}.$$

Again, note that each of these formulas can be written in cnf just by writing $\leftrightarrow$ in terms of conjunctions and disjunctions and by distributing disjunctions over conjunctions, and that the clauses in the resulting cnf-formulas for $\textsc{cons}_\exists(i)$ are i-links: the (first) index of the variables they contain is either i or $i+1$. Also, the size of $\textsc{cons}_\exists$ written in cnf is $c \cdot s \cdot |n|$ for some constant $c \geq 1$.

3.4 Definition of NCONS$_\forall$

The formula $\textsc{ncons}_\forall$ is very similar to $\textsc{cons}_\exists$, since it verifies for universally quantified variables exactly the opposite of what $\textsc{cons}_\exists$ verifies for existentially quantified variables. For this reason, we proceed to its axiomatization directly.

The formula $\textsc{ncons}_\forall$ is defined as

$$\textsc{ncons}_\forall := \exists\boldsymbol{\pi}\exists\boldsymbol{\xi}\exists\boldsymbol{\sigma}\exists\boldsymbol{\chi}\exists\boldsymbol{\omega}\exists\boldsymbol{\kappa}\exists\boldsymbol{\lambda}\left(\textsc{ncons}_\forall^{\mathrm{ini}} \cdot \prod_{i=1}^{s} \textsc{ncons}_\forall(i) \cdot \textsc{ncons}_\forall^{\mathrm{acc}}\right)$$

where $\boldsymbol{\pi}$, $\boldsymbol{\xi}$, $\boldsymbol{\sigma}$, $\boldsymbol{\chi}$, $\boldsymbol{\omega}$, $\boldsymbol{\kappa}$, $\boldsymbol{\lambda}$ are defined as before, $\textsc{ncons}_\forall^{\mathrm{ini}} := \textsc{cons}_\exists^{\mathrm{ini}}$, the formula $\textsc{ncons}_\forall(i)$ is axiomatized identically to $\textsc{cons}_\exists(i)$ except by replacing every occurrence of q_i by $\overline{q_i}$ for every $i \in [s]$, and the formula $\textsc{ncons}_\forall^{\mathrm{acc}}$ is the negation of $\textsc{cons}_\exists^{\mathrm{acc}}$, i.e. the following set of unit clauses:

$$\xi_{s+1}, \sigma_{s+1,2}, \omega_{s+1}, \lambda_{s+1,1}, \dots, \lambda_{s+1,|n|}, \overline{\kappa_{s+1}}.$$

In cnf, the formula $\textsc{ncons}_\forall(i)$ is again a set of i-links, and its size is $c \cdot s \cdot |n|$ for some $c \geq 1$.

3.5 Converting θ to leveled-qbf

Recall that θ was defined as $Q_1\boldsymbol{\tau}_1 \cdots Q_q\boldsymbol{\tau}_q(\textsc{ncons}_\forall + (\textsc{cons}_\exists \cdot \textsc{sat}))$. By writing this formula in prenex form, we obtain the equivalent formula

$$\mathbf{Qz}\ (\textsc{ncons}'_\forall + (\textsc{cons}'_\exists \cdot \textsc{sat}'))$$

where $\mathbf{Qz}$ is the appropriate prefix of quantified variables and the primed formulas are the matrices of the corresponding non-primed qbfs. We would like to write it as a leveled-qbf.

Let a and b be two new variables and let ϑ be the conjunction of the following formulas:

$$\begin{aligned} &a + \textsc{ncons}'_\forall \\ &b + \textsc{ncons}'_\forall \\ &\bar{a} + \textsc{cons}'_\exists \\ &\bar{b} + \textsc{sat}' \end{aligned}$$

It is easy to see that

$$\exists a \exists b(\vartheta) \leftrightarrow \textsc{ncons}'_\forall + (\textsc{cons}'_\exists \cdot \textsc{sat}').$$

We write ϑ in cnf. For the first disjunction $a + \textsc{ncons}'_\forall$, it is enough to add a to every clause of $\textsc{ncons}'_\forall$, and similarly for the others. Note that, except for the variables a and b, the result is a conjunction of i-links.

In order to make them proper i-links, we introduce new variables $\{a_1, \dots, a_{s+1}\}$ and $\{b_1, \dots, b_{s+1}\}$, and clauses $a_i \leftrightarrow a_{i+1}$ and $b_i \leftrightarrow b_{i+1}$ for every $i \in [s]$ to mantain consistency between the introduced variables. Now, we replace each occurrence of a and b in an improper i-link by a_i and b_i respectively. Let ψ' be the resulting formula.

Finally, define

$$\psi := \mathbf{Qz}\exists\mathbf{a}\exists\mathbf{b}(\psi')$$

where $\mathbf{a} = (a_1, \dots, a_{s+1})$ and $\mathbf{b} = (b_1, \dots, b_{s+1})$. Note that the construction guarantees $\psi \leftrightarrow \theta$, and by Claim 1, $\psi \leftrightarrow \phi$.

We partition the variables of ψ in groups $H_1, \dots, H_{s+1}$ where group H_i is the set of variables with (first) index i. We also partition the clauses of ψ in blocks $C_1, \dots, C_{s+1}$ where block C_i is the set of i-links of ψ. Note that, by the definition of i-link, all variables in C_i are contained in $H_i \cup H_{i+1}$. Therefore, ψ is a leveled-qbf with groups $H_1, \dots, H_{s+1}$ and blocks $C_1, \dots, C_{s+1}$.

Now, for every $i \in [s+1]$, the size of H_i is the number of variables with index i in ψ, namely $c \cdot |n|$ for some constant $c \geq 1$. Also, the size of ψ is $d \cdot s \cdot |n|$ for some constant $d \geq 1$. Therefore, ψ is a $c \cdot |n|$-leveled qbf of size $d \cdot s \cdot |n|$ such that $\phi \leftrightarrow \psi$.

Finally, it is clear that all the steps to produce ψ from ϕ can be performed in time polynomial in s, thus finishing the proof.

4 Main Theorem

In this section we prove the main result of the paper.

▶ **Theorem 3.** *There exists an integer $w \geq 1$ such that QBF on inputs of path-width at most w is PSPACE-complete.*

Proof. We show that there exists a constant $n_0 \geq 1$ and a polynomial-time reduction from the canonical PSPACE-complete problem QBF to the restriction of QBF itself to n_0-leveled qbfs. Then the result will follow by setting the path-width to $w = 2n_0 - 1$ and applying Lemma 1.

Let c and d be the constants from the end of section 3. We choose the constant n_0 large enough so that whenever $N \geq n_0$ the following conditions are satisfied:

1. $c \cdot |N| < N$,
2. $c \cdot |c \cdot |N|| \leq \log N$,
3. $(2\log^* N)(\log |N|) \leq \log N$,
4. $d^{2\log^* N} \leq \log N$.

All these conditions can be met simultaneously. The idea of the reduction is to start with an arbitrary qbf formula ϕ_0 with N_0 variables and size S_0, view it as an N_0-leveled qbf, and apply Lemma 2 repeatedly until we get a n_0-leveled qbf for the large fixed constant n_0. Since the final formula will be equivalent to ϕ_0, we just need to make sure that this process terminates in a small number of iterations and that the size of the resulting formula is polynomial in S_0. We formalize this below.

Let ϕ_0 be an arbitrary qbf formula with N_0 variables and size S_0. In particular ϕ_0 is an N_0-leveled qbf of size S_0. If $N_0 \leq n_0$ then ϕ_0 is already n_0-leveled and there is nothing to do. Assume then $N_0 > n_0$. We apply Lemma 2 to get an N_1-leveled qbf of size S_1 where $N_1 = c \cdot |N_0|$ and $S_1 = d \cdot S_0 \cdot |N_0|$. By condition 1 on n_0 we get $N_1 < N_0$, which is progress. Repeating this we get a sequence of formulas $\phi_0, \phi_1, \ldots, \phi_t$, where ϕ_i is an N_i-leveled qbf of size S_i with

1. $N_i = c \cdot |N_{i-1}|$, and
2. $S_i = d^i \cdot S_0 \cdot \prod_{j=0}^{i-1} |N_j|$,

for $i \geq 1$. We stop the process at the first $i = t$ such that $N_t \leq n_0$. We claim that $t \leq 2\log^* N_0$ and that $S_t \leq S_0 \cdot N_0 \cdot \log N_0$. This will be enough, since then the algorithm that computes ϕ_t from ϕ_0 is the required reduction as it runs in time polynomial in the size of the formula, and $\phi_0 \leftrightarrow \phi_t$.

▶ Claim 3. It holds that $t \leq 2\log^* N_0$.

Proof. First, by conditions 1 and 2 on n_0 we have

1. $N_i = c \cdot |N_{i-1}| < N_{i-1}$, and
2. $N_{i+1} = c \cdot |N_i| = c \cdot |c \cdot |N_{i-1}|| \leq \log N_{i-1}$

for every $i \geq 1$ such that $N_{i-1} > n_0$. In particular, this means that the process terminates and t exists. Unfolding the second inequality gives

$$N_{t-1} \leq \log^{(\lfloor (t-1)/2 \rfloor)} N_0.$$

However, by the choice of t we have $N_{t-1} > n_0 \geq 1$, which means that $\lfloor (t-1)/2 \rfloor < \log^* N_0$ and therefore $t \leq 2\log^* N_0$. ◀

Given this bound on t, we bound S_t. We have

$$S_t = d^t \cdot S_0 \cdot \prod_{j=0}^{t-1} |N_j| \leq d^t \cdot S_0 \cdot |N_0|^t,$$

where in the inequality we used the fact that $N_i \leq N_{i-1}$ for every $i \geq 1$ such that $N_{i-1} > n_0$, by condition 1 on n_0. Now:

$$|N_0|^t \leq 2^{(2\log^* N_0)(\log |N_0|)} \leq 2^{\log N_0} = N_0.$$

In the first inequality we used the bound on t, and in the second we used the assumption that $N_0 \geq n_0$ and condition 3 on n_0. Altogether, this gives

$$S_t \leq d^{2\log^* N_0} \cdot S_0 \cdot N_0 \leq S_0 \cdot N_0 \cdot \log N_0,$$

which concludes the proof. Again, we used the assumption that $N_0 \geq n_0$ and condition 4 on n_0. ◀

References

1 H. Chen. Quantified constraint satisfaction and bounded treewidth. Proceedings of the 16th European Conference on Artificial Intelligence (ECAI), pp. 161-165. IOS Press, 2004.

2 H. Chen and V. Dalmau. Decomposing Quantified Conjunctive (or Disjunctive) Formulas, Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science (LICS), pp. 205-214. IEEE Computer Society, 2012.

3 R. Dechter and J. Pearl. Tree clustering for constraint networks. Artificial Intelligence, vol. 38, pp. 353-366. Elsevier, 1989.

4 E. Freuder. Complexity of k-tree structured constraint satisfaction problems. Proceedings of the 8th National Conference on Artificial Intelligence (AAAI), vol. 1, pp. 4-9. AAAI Press, 1990.

5 M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. Proceedings of the 17th IEEE Symposium on Logic in Computer Science (LICS), pp. 215-224. IEEE Computer Society, 2002.

6 G. Gottlob, G. Greco, and F. Scarcello. The complexity of quantified constraint satisfaction problems under structural restrictions. Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI), pp. 150-155. Professional Book Center, 2005.

7 A. Meyer. Weak monadic second order theory of succesor is not elementary recursive. Logic Colloquium. Lecture Notes in Mathematics, vol. 453, pp. 132-154. Springer-Verlag, 1975.

8 G. Pan and M.Y. Vardi. Fixed-parameter hierarchies inside PSPACE. Proceedings of the 21th IEEE Symposium on Logic in Computer Science (LICS), pp. 27-36. IEEE Computer Society, 2006.

Model Counting for CNF Formulas of Bounded Modular Treewidth*

Daniel Paulusma[1], Friedrich Slivovsky[2], and Stefan Szeider[3]

1 School of Engineering and Computing Sciences, Durham University
Durham, DH1 3LE, UK daniel.paulusma@durham.ac.uk
2 Institute of Information Systems, Vienna University of Technology
A-1040 Vienna, Austria fslivovsky@gmail.com
3 Institute of Information Systems, Vienna University of Technology
A-1040 Vienna, Austria stefan@szeider.net

Abstract

The modular treewidth of a graph is its treewidth after the contraction of modules. Modular treewidth properly generalizes treewidth and is itself properly generalized by clique-width. We show that the number of satisfying assignments of a CNF formula whose incidence graph has bounded modular treewidth can be computed in polynomial time. This provides new tractable classes of formulas for which #SAT is polynomial. In particular, our result generalizes known results for the treewidth of incidence graphs and is incomparable with known results for clique-width (or rank-width) of signed incidence graphs. The contraction of modules is an effective data reduction procedure. Our algorithm is the first one to harness this technique for #SAT. The order of the polynomial time bound of our algorithm depends on the modular treewidth. We show that this dependency cannot be avoided subject to an assumption from Parameterized Complexity.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Satisfiability, Model Counting, Parameterized Complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.55

1 Introduction

Given a set $\{x_1, \dots, x_n\}$ of variables, consider a propositional formula $F_{m,n}$ defined as follows. $F_{m,n}$ consists of m distinct clauses each containing n literals over $\{x_1, \dots, x_n\}$, so that every variable occurs in every clause. It is easy to see that $F_{m,n}$ has exactly $2^n - m$ satisfying assignments. The vertices of the incidence graph of $F_{m,n}$ (i.e., the bipartite graph whose vertex classes consist of variables and clauses, and a variable is adjacent to the clauses it occurs in) can be partitioned into two large modules (a module in a graph is a set S of vertices such that for any vertex $v \notin S$, every vertex in S is a neighbor of v or every vertex in S is a non-neighbor of v). By contracting these modules, the incidence graph reduces to a single edge.

Contraction of modules is an important preprocessing step for a wide range of combinatorial optimization problems [14]. The aim of this paper is to harness its power for propositional model counting (#SAT), a well-studied problem with various applications in artifical intelligence, such as probabilistic inference [1]. We consider CNF formulas whose incidence graph

* Slivovsky and Szeider's research was supported by the ERC, grant reference 239962. Paulusma's research was supported by EPSRC, grant reference EP/G043434/1.

$$\begin{tikzcd}
 & \text{incidence } \beta\text{-hypertree width} & \\
 & \text{incidence clique-width} \arrow[u] & \\
\textbf{signed incidence clique-width} \arrow[ur] & & \textbf{modular incidence treewidth} \arrow[ul] \\
 & \textbf{incidence treewidth} \arrow[ul] \arrow[ur] & \\
 & \textbf{primal treewidth} \arrow[u] &
\end{tikzcd}$$

Figure 1 Hierarchy of structural parameters. An arc from parameter p to parameter q reads as "q is bounded whenever p is bounded." Bold type indicates parameters that are known to render #SAT polynomial-time tractable when bounded.

has bounded treewidth after contraction of modules (we call this the *modular incidence treewidth*). We will prove that #SAT is polynomial-time tractable for such formulas.

▶ **Theorem 1.** *The number of satisfying assignments of a CNF formula F with modular incidence treewidth at most k can be computed in time $\ell^{\mathcal{O}(k)}$, where ℓ is the length of F.*

This result characterizes new tractable classes for a notoriously hard problem. #SAT is #P-complete in general [24] and remains #P-hard even for monotone 2CNF formulas and Horn 2CNF formulas. It is NP-hard to approximate the number of models of a formula with n variables is within $2^{n^{1-\varepsilon}}$ for $\varepsilon > 0$. Again, this hardness result still holds for monotone 2CNF formulas and Horn 2CNF formulas [22]. These syntactic restrictions do no lead to tractability of #SAT.

By contrast, modular incidence treewidth is a so-called *structural* parameter. Structural restrictions are applied in terms of parameters (invariants) of graphs or hypergraphs associated with formulas. Figure 1 illustrates the relation of modular incidence treewidth to other structural parameters. For a detailed discussion, see Section 1.1 below. Our algorithmic result presented in Section 3 is achieved by dynamic programming on a tree decomposition of the *modular incidence graph* (the graph obtained from the incidence graph of a formula by contracting modules). Vertices in the modular incidence graph represent entire *modules* (i.e., sets of variables or sets of clauses), whose size cannot be bounded in terms of the modular incidence treewidth alone. Algorithms for #SAT on formulas of bounded treewidth typically rely on data structures indexed by the subsets of variables and clauses associated with a bag of the tree decomposition [23]. The number of subsets of variables and clauses occurring in even a single module can be exponential in the length of the input formula, so these algorithms do not yield tractability in our case. It is a significant challenge to encode the information required to perform dynamic programming in space polynomially bounded by the input size. Our main technical contribution is the use of *projections* in solving this task. We define an equivalence relation on assignments based on their projections onto a particular formula. This formula is determined by the boundary of a subgraph induced by the decomposition. The resulting equivalence relation is sufficiently precise while its rank can still be polynomially bounded. This allows our algorithm to run in polynomial time. Note that the order of the polynomial time bound in Theorem 1 is a function in the modular incidence treewidth. The hardness result presented in Section 4 shows that one cannot replace this function by a constant (subject to an assumption from Parameterized Complexity).

1.1 Related structural parameters

For sake of comparing two structural parameters p and q of CNF formulas, we say p *dominates* q if there is a function f such that $p(F) \leq f(q(F))$ for all formulas F. Parameters p and q are *equivalent* if p dominates q and q dominates p. We say p is *more general* than q if p dominates q but not the other way around. Two parameters p and q are *incomparable* if neither p dominates q nor q dominates p. The *primal treewidth* of a formula is the treewidth of its primal graph. The primal graph has as vertices the variables of the given formula, and two variables are joined by an edge if they occur together in a clause. It is well known that #SAT on formulas of bounded primal treewidth is linear-time tractable [23]. A similar result has been shown in terms of the *branchwidth* of formulas [1], a parameter that is equivalent to primal treewidth. The *incidence treewidth* of a formula is the treewidth of its incidence graph. This parameter is known to be more general than primal treewidth [13]. Again, #SAT on formulas of bounded incidence treewidth is linear-time tractable [10, 23]. *Clique-width* is a graph invariant based on graph grammars [4]. *Signed clique-width* is a variant of clique-width for directed graphs [7]. The *signed incidence clique-width* of a formula corresponds to the signed clique-width of its *signed incidence graph*, which is obtained from the incidence graph by orientating edges so as to indicate positive or negative occurrences of variables. A polynomial-time algorithm for #SAT on formulas of bounded signed incidence clique-width is due to Fischer, Makowsky, and Ravve [10]. Clique-width is typically approximated by means of another parameter known as *rank-width* [20]. A class of graphs has bounded rank-width if and only if it has bounded clique-width. But while it is open whether, for fixed $k \geq 4$, graphs of clique-width at most k can be recognized in polynomial time [9], this is known to be the case for graphs of rank-width at most k [15]. Ganian, Hlinený, and Obdrzálek proposed a polynomial-time algorithm for #SAT for formulas of bounded *signed incidence rank-width*, a parameter that is equivalent to signed incidence clique-width [12]. Signed incidence clique-width and signed incidence rank-width are currently the most general structural parameters based on width measures for which #SAT is known to be polynomial time-tractable. The following two examples show that modular incidence treewidth is incomparable with these parameters and more general than incidence treewidth. Due to space constraints, we will only sketch the proofs and refer to known results wherever possible.

▶ **Example 2** (Fischer, Makowsky, and Ravve [10])**.** Let $x_1, \ldots, x_m$ be distinct variables. The formula φ_m is defined as the set of clauses $C_{i,j}$ for $1 \leq i, j \leq m$ and $i \neq j$, where $C_{i,j} = (\{x_1, \ldots, x_m\} \setminus \{x_i, x_j\}) \cup \{\neg x_i, \neg x_j\}$. The signed incidence clique-width of φ_m tends to infinity with m. The (unsigned) incidence graph corresponds to the complete bipartite graph $K_{n,m}$ for $n = \binom{m}{2}$. As in our initial example, module contraction reduces $K_{n,m}$ to a single edge, so the modular incidence treewidth of φ_m is 1 for arbitrary m.

▶ **Example 3.** Let $x_1, \ldots, x_m, y_1, \ldots, y_m$ be distinct variables. We let ψ_m consist of the clauses C_i for $1 \leq i \leq m$ where $C_i = \{y_i, x_1, \ldots, x_m\}$, along with m singleton clauses $\{x_1\}, \ldots, \{x_m\}$. The incidence graph $I(\psi_m)$ of ψ_m has no nontrivial modules (it is *prime*), so the modular incidence treewidth and incidence treewidth of ψ_m coincide. Since $I(\psi_m)$ contains $K_{m,m}$ as a subgraph, its treewidth is at least m. By contrast, it can be shown that the signed incidence clique width of ψ_m is at most 5 for arbitrary m.

▶ **Proposition 4.** *Modular incidence treewidth and signed incidence clique-width are incomparable.*

It is readily verified that modular incidence treewidth dominates incidence treewidth: by contracting modules, we obtain an induced subgraph of the incidence graph, and the treewidth of a graph is bounded from below by the treewidth of any of its subgraphs. Also note that the sequence of formulas from Example 2 has unbounded treewidth. By combining these facts, we can conclude that modular incidence treewidth is more general than incidence treewidth. Incidence clique-width is known to be more general than signed incidence clique-width [10]. It is also more general than modular incidence treewidth.

▶ **Proposition 5.** *Incidence clique-width is more general than modular incidence treewidth.*

Proof. It is well known that there is a function that provides an upper bound on the clique-width of any graph in terms of its treewidth, and that clique-width is invariant under contraction of modules [7]. It follows that incidence clique-width dominates modular incidence treewidth. Because incidence clique-width dominates signed incidence clique-width, the sequence of formulas described in Example 3 has bounded incidence clique-width. We conclude that incidence clique-width is more general than modular incidence treewidth. ◀

The incidence β-hypertree width is parameter that is yet more general than incidence clique-width [13]. At this time, it remains open whether #SAT is polynomial-time tractable on formulas for which one of these parameters is bounded.

We note that tractability of #SAT for formulas of bounded primal treewidth, bounded incidence treewidth, or bounded signed incidence clique-width can also be established using algorithmic meta-theorems by Courcelle, Makowsky, and Rotics [5, 6]. The hardness result presented in Section 4 implies that our Theorem 1 cannot be proved in this way.

2 Preliminaries

Let X and Y be sets and let $f : X \to Y$ be a function. We write $f^{-1}(y) = \{x \in X \mid f(x) = y\}$. For a subset $X' \subseteq X$, we let $f|_{X'}$ denote the restriction of f to X'. If $Y = 2^Z$ for some set Z and $f(x) = \{z\}$ for some $z \in Z$, then we may write $f(x) = z$ instead. If $g : X^* \to Y^*$ is a function with $g(x) = f(x)$ for all $x \in X \cap X^*$, then the function $f \cup g : X \cup X^* \to Y \cup Y^*$ is defined as $(f \cup g)(x) = f(x)$ if $x \in X$ and $(f \cup g)(x) = g(x)$ if $x \in X^* \setminus X$.

We assume an infinite supply of propositional *variables*. A *literal* is a variable x or a negated variable $\overline{x}$; we put $\mathrm{var}(x) = \mathrm{var}(\overline{x}) = x$; if $y = \overline{x}$ is a literal, then we write $\overline{y} = x$. For a set S of literals we put $\overline{S} = \{\, \overline{x} \mid x \in S \,\}$; S is *tautological* if $S \cap \overline{S} \neq \emptyset$. A *clause* is a finite non-tautological set of literals. A finite set of clauses is a *CNF formula* (or *formula*, for short). The *length* of a formula F is given by $\sum_{C \in F} |C|$. The *union* of two clauses C and D denoted CD is the union of the literals of C and D. A variable x *occurs* in a clause C if $x \in C \cup \overline{C}$. We let $\mathrm{var}(C)$ denote the set of variables that occur in C. A variable x *occurs* in a formula F if it occurs in one of its clauses, and we let $\mathrm{var}(F) = \bigcup_{C \in F} \mathrm{var}(C)$.

Let F be a formula. The *incidence graph* of F is the bipartite graph $I(F)$ with vertex set $\mathrm{var}(F) \cup F$ and edge set $\{\, Cx \mid C \in F \text{ and } x \in \mathrm{var}(C) \,\}$. Two vertices are *twins* if they have the same neighbors in $I(F)$. The equivalence classes of the twin relation are called *modules*. By the definition of $I(F)$, twins either consist of two variables or of two clauses. If the vertices of a module correspond to clauses, then we call the module a *clause module*; otherwise we call it a *variable module*. By definition, all clauses of any clause module $\mathcal{C}$ of F contain all variables of any variable module X of F if and only if one clause of $\mathcal{C}$ contains at least one variable from X. This implies that the set of variable modules of $\mathcal{C}$ is a subset of the set of variable modules of F. For a set of clause or variable modules $\mathcal{S}$, we let $\langle \mathcal{S} \rangle = \bigcup_{S \in \mathcal{S}} S$ denote the union of the elements of $\mathcal{S}$. The *modular incidence graph* $I^*(F)$ is the bipartite graph obtained from $I(F)$ after removing all but one vertices of each module. A *truth assignment* is a mapping $\tau : X \to \{\, 0, 1 \,\}$ defined on some set $X \subseteq \mathrm{var}(F)$ of variables.

For $x \in X$, we define $\tau(\overline{x}) = 1 - \tau(x)$. A truth assignment τ *satisfies* a clause C if C contains some literal x with $\tau(x) = 1$. If τ satisfies all clauses of F, then τ *satisfies* F; in that case we call F satisfiable. The *satisfiability* (SAT) problem is that of testing whether a given formula is satisfiable. *Propositional model counting* (#SAT) is a generalization of SAT that asks for the number of satisfying truth assignments.

Let X be a set of variables. For a clause C, we let $C^X = \{\, \ell \in C \mid \mathrm{var}(\ell) \in X \,\}$. For a formula F, we let $F^X = \{\, C^X \mid C \in F \,\} \setminus \{\emptyset\}$. For a truth assignment τ, we let $\mathbf{clause}(\tau) = \tau^{-1}(0) \cup \overline{\tau^{-1}(1)}$. This leads to the following lemma.

▶ **Lemma 6.** *Let $\tau : X \to \{0,1\}$ be a truth assignment, and let C be a clause. Then C is not satisfied by τ if and only if $C^X = \mathbf{clause}(\tau|_{X \cap \mathrm{var}(C)})$.*

We now define the notion of a projection, which plays an important role in our paper. As an aside, Kaski, Koivisto, and Nederlof [16] recently showed that SAT can be solved in polynomial time for formulas with a bounded number of projections. Let F be a formula and X be a set of variables. We refer to the set of clauses of F *not* satisfied by a truth assignment $\sigma : X \to \{0,1\}$ as the *(negative) projection* of σ on F denoted $F(\sigma)$. We denote the set of all these projections by $\mathscr{P}_{(F,X)} = \{\, F(\sigma) \mid \sigma : X \to \{0,1\} \,\}$. If $X \supseteq \mathrm{var}(F)$, then we may write $\mathscr{P}_F$ instead, as $\mathscr{P}_{(F,X)} = \mathscr{P}_{(F,\mathrm{var}(F))}$ holds in that case. Note that F is satisfiable if and only if the *empty* projection $\emptyset$ belongs to $\mathscr{P}_F$, and that the number of satisfying truth assignments of F is equal to $|\{\sigma : \mathrm{var}(F) \to \{\,0,1\} \mid F(\sigma) = \emptyset\,\}|$. The following lemma states a useful property of projections.

▶ **Lemma 7.** *Let F be a formula and let X, Y be two sets of variables. Let $\sigma : X \to \{0,1\}$ and $\tau : Y \to \{0,1\}$ be two truth assignments that agree on $X \cap Y$. Then $F(\sigma \cup \tau) = F(\sigma) \cap F(\tau)$.*

For a clause C and a formula F we let $\mathbf{select}(F, C) = \{\, C' \in F \mid C \subseteq C' \,\}$. We will now prove two useful lemmas. The first lemma is for clause modules $\mathcal{C}$. It implies that every truth assignment on $\mathrm{var}(\langle \mathcal{C} \rangle)$ either satisfies $\mathcal{C}$ or does not satisfy a unique clause of $\mathcal{C}$. The second lemma is similar but with respect to variable modules X.

▶ **Lemma 8.** *Let $\mathcal{C}$ be a clause module of a formula F, and let τ be a truth assignment defined on a set X of variables. Then $\mathcal{C}(\tau) = \mathbf{select}(\mathcal{C}, \mathbf{clause}(\tau|_{X \cap \mathrm{var}(\mathcal{C})}))$.*

Proof. Let $C \in \mathcal{C}$. Because $\mathcal{C}$ is a clause module, $\mathrm{var}(C) = \mathrm{var}(\mathcal{C})$. Then, by using Lemma 6 and the definitions of **clause** and **select**, we find that $C \in \mathcal{C}(\tau)$ if and only if C is not satisfied by τ if and only if $C^X = \mathbf{clause}(\tau|_{X \cap \mathrm{var}(C)}) = \mathbf{clause}(\tau|_{X \cap \mathrm{var}(\mathcal{C})})$ if and only if $C \in \mathbf{select}(\mathcal{C}, \mathbf{clause}(\tau|_{X \cap \mathrm{var}(\mathcal{C})}))$. ◀

▶ **Lemma 9.** *Let X be a variable module of a formula F, and let τ be a truth assignment defined on a superset of X. If $F^X(\tau) \neq \emptyset$, then $F^X(\tau) = \mathbf{clause}(\tau|_X)$.*

Proof. Let $C \in F^X$. Because X is a variable module, $\mathrm{var}(C) = X$. Lemma 6 tells us that C is not satisfied by τ if and only if $C = \mathbf{clause}(\tau|_{X \cap \mathrm{var}(C)}) = \mathbf{clause}(\tau|_X)$. ◀

We also need the following lemma.

▶ **Lemma 10.** *Let X be a variable module of a formula F. Let $E = \{\sigma : X \to \{0,1\} \mid F^X(\sigma) = \Pi\}$ for some $\Pi \in \mathscr{P}_{F^X}$. Then $|E| = 1$ if $\Pi \neq \emptyset$, and $|E| = 2^{|X|} - |F^X|$ if $\Pi = \emptyset$.*

Proof. First suppose that $\Pi \neq \emptyset$. By Lemma 9, the only truth assignment $\tau : X \to \{0,1\}$ with $F^X(\tau) = \Pi$ is the truth assignment τ_X with $F^X(\tau_X) = \mathbf{clause}(\tau_X)$. Hence, $|E| = 1$ in this case. Now suppose that $\Pi = \emptyset$. The number of truth assignments defined on X is equal to $2^{|X|}$. By Lemma 9, each such truth assignment τ_X does not satisfy one unique clause with set of variables X, namely the clause $\mathbf{clause}(\tau_X)$. Then there are exactly $2^{|X|} - |F^X|$ truth assignments τ_X that do satisfy F^X, i.e., that have $F^X(\tau_X) = \emptyset = \Pi$. Hence, in this case, $|E| = 2^{|X|} - |F^X|$. ◀

We finish this section with some terminology on tree decompositions. Let $G = (V_G, E_G)$ be a finite, undirected graph with neither self-loops nor multiple edges. A *tree decomposition* of G is a triple (T, χ, r), where $T = (V_T, E_T)$ is a tree rooted at r and $\chi : V_T \to 2^{V_G}$ is a labeling of the vertices of T (called *nodes*) by subsets of V_G (called *bags*) such that the following conditions hold:

1. $\bigcup_{t \in V_T} \chi(t) = V_G$,
2. for each edge $uv \in E_G$, there is a node $t \in V_T$ with $\{u, v\} \subseteq \chi(t)$,
3. for each vertex $x \in V_G$, the set of nodes t with $x \in \chi(t)$ forms a connected subtree of T.

The *width* of a tree decomposition (T, χ) is the size of a largest bag $\chi(t)$ minus 1. The *treewidth* of G is the minimum width over all possible tree decompositions of G. A tree decomposition (T, χ, r) is *nice* if T is a binary tree such that the nodes of T belong to one of the following four types:

A. a *leaf node* t is a leaf of T,
B. an *introduce node* t has one child t' and $\chi(t) \setminus \{v\} = \chi(t')$ for some vertex $v \in V_G$,
C. a *forget node* t has one child t' and $\chi(t') \setminus \{v\} = \chi(t)$ for some vertex $v \in V_G$,
D. a *join node* t has two children t_1, t_2' and $\chi(t) = \chi(t_1) = \chi(t_2)$.

Kloks [17] showed that every tree decomposition of a graph G can be converted in linear time to a nice tree decomposition, such that the size of the largest bag does not increase, and the corresponding tree has at most $4|V_G|$ nodes.

Let F be a formula. We call the treewidth of $I^*(F)$ the *modular incidence treewidth* of F. Let (T, χ, r) be a nice tree decomposition of $I^*(F)$. For $t \in V_T$, we write $\chi_c(t)$ and $\chi_v(t)$ to denote the sets of clause modules and variable modules in $\chi(t)$, respectively. Note that $\chi(t) = \chi_c(t) \cup \chi_v(t)$. Moreover, we let $\mathcal{X}_t$ and $\mathcal{F}_t$ denote the set of variable modules and the set of clause modules occurring in the subtree rooted at t, respectively. We write $X_t = \langle \mathcal{X}_t \rangle$ and $F_t = \langle \mathcal{F}_t \rangle$. Note that $X_r = \text{var}(F)$ and $F_r = F$.

3 Solving #SAT for Formulas of Bounded Modular Treewidth

In this section, we present an algorithm for computing the number of satisfying truth assignments of a formula F. This algorithm runs in polynomial time provided that the modular incidence treewidth of F is bounded. We begin by explaining the main ideas.

Let F be a formula and X be a set of variables. We can partition truth assignments defined on X into equivalence classes with respect to a relation $\sim_{(F,X)}$, which is defined as follows. Let $\sigma, \tau : X \to \{0, 1\}$ be two distinct truth assignments. Then $\sigma \sim_{(F,X)} \tau$ if and only if σ and τ satisfy exactly the same set of clauses of F, or equivalently, if and only if $F(\sigma) = F(\tau)$. Due to the latter equivalence, we can speak about *the* projection of an equivalence class of $\sim_{(F,X)}$ on F. Recall that the number of satisfying truth assignments of F is equal to $|\{\sigma : \text{var}(F) \to \{0, 1\} \mid F(\sigma) = \emptyset\}|$, which is the size of the equivalence class of $\sim_{(F,\text{var}(F))}$ corresponding to the empty projection.

Now let (T, χ, r) be a nice tree decomposition of $I^*(F)$. We will apply dynamic programming over (T, χ, r). As is usual, we start in the leaves of the tree and, using the parent-child relation, move to nodes closer to the root, and we stop after having processed the root. For each node $t \in V_T$, we define the formula

$$F_t^* \;=\; \big\{F^X \mid X \in \chi_v(t)\big\} \,\cup\, F_t \;=\; \big\{F^X \mid X \in \chi_v(t)\big\} \,\cup\, \langle \chi_c(t)\rangle \,\cup\, F_t \setminus \langle \chi_c(t)\rangle,$$

and we compute the sizes of those equivalence classes $[\tau]$ of $\sim_{(F_t^*,X_t)}$ that consist of truth assignments τ with $(F_t \setminus \langle\chi_c(t)\rangle)(\tau) = \emptyset$; we call such equivalence classes *transferable*. Below we explain the reasons why we do this.

First, the union of the transferable equivalence classes of $\sim_{(F_r^*,X_r)}$ that consist of truth assignments τ with $\langle\chi_c(r)\rangle(\tau) = \emptyset$ in addition to $(F_r \setminus \langle\chi_c(t)\rangle)(\tau) = (F \setminus \langle\chi_c(t)\rangle)(\tau) = \emptyset$ contains all satisfying truth assignments of F. Note that such truth assignments may not satisfy some formula F^X for some $X \in \chi_v(r)$, but in that case only formulas in $F^X \setminus F_r = F^X \setminus F$ are not satisfied, and these are irrelevant for our output.

Second, we do not have to compute the sizes of any non-transferable equivalence classes of $\sim_{(F_t^*,X_t)}$. The reason for this is that these equivalence classes only contain truth assignments τ that cannot be extended to satisfying truth assignments of F. This can be seen as follows. Let τ be a truth assignment from a non-transferable equivalence class of $\sim_{(F_t^*,X_t)}$. By definition, $F_t \setminus \langle\chi_c(t)\rangle$ contains a clause C not satisfied by τ. Then C must contain at least one variable $x \in X_r \setminus X_t$ in order to be satisfied by an extension of τ. Let $\mathcal{C}$ be the clause module that contains C. Let X be the variable module that contains x. Then $X\mathcal{C} \in I^*(F)$. Hence, by condition 2 of the definition of a tree decomposition, there exists a node $t' \in V_T$ with $\{X, \mathcal{C}\} \subseteq \chi(t')$. Because $x \in X_r \setminus X_t$, we find that $X \in \mathcal{X}_r \setminus \mathcal{X}_t$. Because $X \in \chi(t')$, this means that t' is not a node of the subtree of T rooted at t. Because $\mathcal{C} \in \chi(t')$, we then find that $\mathcal{C} \in \mathcal{F}_r \setminus \mathcal{F}_t$. However, as $C \in F_t \setminus \langle\chi_c(t)\rangle$, we also have $\mathcal{C} \in \mathcal{F}_t \setminus \chi_c(t)$. This violates condition 3 of the definition of a tree decomposition. Hence, non-transferable equivalence classes may be discarded during our dynamic programming.

Third, we must keep track of how truth assignments that not yet satisfy all clauses in F can be extended to truth assignments that do satisfy F in a later stage of the dynamic programming. In particular, such truth assignments may not yet satisfy clauses C that belong to clause modules in $\chi_c(t)$ or that contain variables from variable modules in $\chi_v(t)$; in the latter case their restriction C^X belongs to F^X for some $X \in \chi_v(t)$. In order to do this bookkeeping we must partition truth assignments that satisfy $F_t \setminus \langle\chi_c(t)\rangle$ into equivalence classes of truth assignments that satisfy exactly the same clauses of any F^X with $X \in \chi_v(t)$ and exactly the same clauses of any $\mathcal{C} \in \chi_c(t)$. The reason why the partitioning does not cause an exponential blow-up if the modular incidence treewidth of F is bounded is due to two of our lemmas from Section 2. For a clause module $\mathcal{C} \in \chi_c(t)$, the number of equivalence classes of $\sim_{(\mathcal{C},X_t)}$ on is bounded by $|\mathcal{C}|+1$ due to Lemma 8. For a variable module $X \in \chi_v(t)$, the number of equivalence classes of $\sim_{(F^X,X_t)}$ is bounded by $|F|+1$ due to Lemma 9. Hence, the total number of different transferable equivalence classes of $\sim_{(F_t^*,X_t)}$ is at most

$$\prod_{\mathcal{C}\in\chi_c(t)} (|\mathcal{C}|+1) \cdot \prod_{X\in\chi_v(t)} (|F|+1) \le (|F|+1)^{|\chi_c(t)|+|\chi_v(t)|} = (|F|+1)^{|\chi(t)|} \le (|F|+1)^k, \qquad (1)$$

where k denotes the treewidth of $I^*(F)$, i.e., the modular incidence treewidth of F. We observe that this bound is polynomial if k is fixed.

In order to describe the transferable equivalence classes, we use some terminology introduced by Ganian, Hlinený and Obdrzálek [12], which we adjusted for our purposes. Let $t \in T$. A *shape* for t is a pair of mappings (α, θ) where α has domain $\chi_v(t)$ with $\alpha(X) \in \mathscr{P}_{F^X}$ for all $X \in \chi_v(t)$ and θ has domain $\chi_c(t)$ with $\theta(\mathcal{C}) \in \mathscr{P}_{(\mathcal{C},X_t)}$ for all $\mathcal{C} \in \chi_c(t)$. An assignment $\tau : X_t \to \{0,1\}$ is said to be *of shape* (α, θ) if it satisfies the following three conditions:

(a) $F^X(\tau) = \alpha(X)$ for all $X \in \chi_v(t)$
(b) $\mathcal{C}(\tau) = \theta(\mathcal{C})$ for all $\mathcal{C} \in \chi_c(t)$
(c) $(F_t \setminus \langle\chi_c(t)\rangle)(\tau) = \emptyset$.

In other words, the set of assignments τ that are of shape (α, θ) describes exactly one transferable equivalence class of $\sim_{(F_t^*, X_t)}$. From now we denote this class by $N_t(\alpha, \theta)$, and we write $n_t(\alpha, \theta) = |N_t(\alpha, \theta)|$. We denote the set of all shapes for t that correspond to a transferable equivalence class by $\mathscr{S}_t$. By (1), we have $|\mathscr{S}_t| \leq (|F| + 1)^k$ for all nodes $t \in V_T$. Also note that any truth assignment $\tau : X_t \to \{0, 1\}$ has a (unique) shape if and only if $(F_t \setminus \langle\chi_c(t)\rangle)(\tau) = \emptyset$. We sometimes denote the shape of such a truth assignment τ by $(\alpha^t_\tau, \theta^t_\tau)$, where $\alpha^t_\tau(X) = F^X(\tau)$ for all $X \in \chi_v(t)$ and $\theta^t_\tau(\mathcal{C}) = \mathcal{C}(\tau)$ for all $\mathcal{C} \in \chi_c(t)$. Because equivalence classes are nonempty by definition, not all pairs (α, θ) with $\alpha(X) \in \mathscr{P}_{F^X}$ for all $X \in \chi_v(t)$ and $\theta(\mathcal{C}) \in \mathscr{P}_{(\mathcal{C}, X_t)}$ for all $\mathcal{C} \in \chi_c(t)$ form a shape for a node $t \in V_T$. We make this more explicit in our next lemma (see condition (ii) in particular).

▶ **Lemma 11.** *Let $(\alpha, \theta) \in \mathscr{S}_t$ with $t \in V_T$, and let $\chi_v^*(t) \subseteq \chi_v(t)$. Moreover, let $\tau : X_t \to \{0, 1\}$ satisfy $F^X(\tau) = \alpha(X)$ for all $X \in \chi_v^*(t)$. For all $\mathcal{C} \in \chi_c(t)$, the following three conditions hold:*

(i) If $\mathcal{C}$ has no variable modules in $\chi_v^*(t)$, then $\mathcal{C}(\tau|_{\langle\chi_v^*(t)\rangle}) = \mathcal{C}$.
(ii) If $\mathcal{C}$ has some variable module $X \in \chi_v^*(t)$ with $\alpha(X) = \emptyset$, then $\mathcal{C}(\tau|_{\langle\chi_v^*(t)\rangle}) = \mathcal{C}(\tau) = \emptyset$, and moreover, $\theta(\mathcal{C}) = \emptyset$ should $\tau \in N_t(\alpha, \theta)$.
(iii) If $\mathcal{C}$ has exactly $p \geq 1$ variable modules $X_1, \ldots, X_p$ in $\chi_v^*(t)$ and $\alpha(X_i) \neq \emptyset$ for $i = 1, \ldots, p$, then $\mathcal{C}(\tau|_{\langle\chi_v^*(t)\rangle}) = \mathbf{select}(\mathcal{C}, \alpha(X_1) \cdots \alpha(X_p))$.

We will now give the exact details of our dynamic programming, i.e., how we compute all sizes $n_t(\alpha, \theta)$ over all $t \in V_T$ in order to be able to compute the desired output

$$n = \sum_{(\alpha,\theta)\in\mathscr{S}_r^*} n_r(\alpha, \theta),$$

where $\mathscr{S}_r^*$ consists of those $(\alpha, \theta) \in \mathscr{S}_r$ with $\theta(\mathcal{C}) = \emptyset$ for all $\mathcal{C} \in \chi_c(r)$. Note that $\mathscr{S}_r^* = \emptyset$ is possible; in that case F is not satisfiable and $n = 0$.

Recall that the nodes of a tree of a nice tree decomposition can be partitioned into four types of nodes. Our next four lemmas show how to compute the sizes $n_t(\alpha, \theta)$ for these four types, i.e., for leaf nodes, introduce nodes, forget nodes and join nodes t, respectively.

▶ **Lemma 12.** *Let t be a leaf node and $(\alpha, \theta) \in \mathscr{S}_t$. Then $n_t(\alpha, \theta) = \prod_{X\in\alpha^{-1}(\emptyset)}(2^{|X|} - |F^X|)$.*

Let $(\alpha, \theta) \in \mathscr{S}_t$ for some $t \in V_T$. We define a mapping g with domain $\chi_c(t) \times \chi_v(t)$ as follows. When $X \in \chi_v(t)$ is not a variable module of a clause $\mathcal{C} \in \chi_c(t)$, we let $g(\mathcal{C}, X) = \mathcal{C}$. Otherwise, we let $g(\mathcal{C}, X) = \emptyset$ if $\alpha(X) = \emptyset$, and $g(\mathcal{C}, X) = \mathbf{select}(\mathcal{C}, \alpha(X))$ if $\alpha(X) \neq \emptyset$.

▶ **Lemma 13.** *Let $t \in T$ be an introduce node with child t', such that $\chi(t) \setminus \{S\} = \chi(t')$ for a module $S \in \chi(t)$. Let $(\alpha, \theta) \in \mathscr{S}_t$. Moreover, let $\alpha' = \alpha|_{\chi_v(t')}$ and $\theta' = \theta|_{\chi_c(t')}$.*

(i) If $S \in \chi_v(t)$, then $$n_t(\alpha, \theta) = \begin{cases} \sum_{\theta^*\in\mathscr{T}} n_{t'}(\alpha', \theta^*) & \text{if } \alpha(S) \neq \emptyset \\ (2^{|S|} - |F^S|) \sum_{\theta^*\in\mathscr{T}} n_{t'}(\alpha', \theta^*) & \text{if } \alpha(S) = \emptyset, \end{cases}$$

where $\mathscr{T} = \{\, \theta^* \mid (\alpha', \theta^*) \in \mathscr{S}_{t'}$ and $\theta^*(\mathcal{C}) \cap g(\mathcal{C}, S) = \theta(\mathcal{C})$ for all $\mathcal{C} \in \chi_c(t) \,\}$.

(ii) If $S \in \chi_c(t)$, then $n_t(\alpha, \theta) = n_{t'}(\alpha, \theta')$.

▶ **Lemma 14.** *Let $t \in T$ be a forget node with child t', such that $\chi(t) = \chi(t') \setminus \{S\}$ for a module $S \in \chi(t')$. Let $(\alpha, \theta) \in \mathscr{S}_t$.*

(i) If $S \in \chi_v(t')$, then $n_t(\alpha, \theta) = \sum_{\Pi \in \mathscr{P}_{FS}} n_{t'}(\alpha \cup \{(S, \Pi)\}, \theta)$.
(ii) If $S \in \chi_c(t')$, then $n_t(\alpha, \theta) = n_{t'}(\alpha, \theta \cup \{(S, \emptyset)\})$.

▶ **Lemma 15.** *Let $t \in T$ be a join node with children t_1 and t_2. Let $(\alpha, \theta) \in \mathscr{S}_t$. Moreover, let $\mathscr{T}_{1,2} = \{\, (\theta_1, \theta_2) \mid (\alpha, \theta_1) \in \mathscr{S}_{t_1}, (\alpha, \theta_2) \in \mathscr{S}_{t_2}$, and $\theta_1(\mathcal{C}) \cap \theta_2(\mathcal{C}) = \theta(\mathcal{C})$ for all $\mathcal{C} \in \chi_c(t) \,\}$. Then the following equality holds:*

$$n_t(\alpha, \theta) = \frac{1}{\prod\limits_{X \in \alpha^{-1}(\emptyset)} (2^{|X|} - |F^X|)} \sum_{(\theta_1, \theta_2) \in \mathscr{T}_{1,2}} n_{t_1}(\alpha, \theta_1) \cdot n_{t_2}(\alpha, \theta_2).$$

We are now ready to present the proof of our main result, which we restate below.

Theorem 1. *The number of satisfying assignments of a CNF formula F with modular incidence treewidth at most k can be computed in time $\ell^{\mathcal{O}(k)}$, where ℓ is the length of F.*

Proof. Let F be a formula with modular incidence treewidth at most k. We first construct $I(F)$ and perform module contraction to obtain $I^*(F)$. Clearly, this can be done in time $\mathcal{O}(\ell^c)$ for some constant c independent of F. By using Bodlaender's algorithm [2] we obtain in linear time a tree decomposition of $I^*(F)$ of width at most k. Recall that Kloks [17] showed that such a tree decomposition can be converted in linear time to a nice tree decomposition (T, χ, r) of width at most k, and at most $4|V_{I^*(F)}| \leq 4\ell$ nodes. Also recall that the desired output is

$$n = \sum_{(\alpha, \theta) \in \mathscr{S}_r^*} n_r(\alpha, \theta),$$

where $\mathscr{S}_r^*$ consists of those $(\alpha, \theta) \in \mathscr{S}_r$ with $\theta(\mathcal{C}) = \emptyset$ for all $\mathcal{C} \in \chi_c(r)$. In order to compute n, we compute the sizes $n_t(\alpha, \theta)$ for all $t \in V_T$. Here we follow a bottom-up approach starting at the leaves. For each node $t \in V_T$, we use one of the Lemmas 12–15 depending on the type of t, i.e., whether t is a leaf, introduce, forget or join node, respectively. The correctness of our algorithm follows from these lemmas and some extra arguments: when any new clause module $\mathcal{C}$ is introduced in a node t (that is either a leaf or an introduce node) we find that $\theta(\mathcal{C}) = \mathcal{C}(\tau) = \mathcal{C}(\tau|_{\langle \chi_v(t) \rangle})$ should τ be a truth assignment in $N_t(\alpha, \theta)$ for some pair (α, θ), and then Lemma 11 tells us that $\theta(\mathcal{C})$ must be fixed to some set of clauses. Hence, if $\theta(\mathcal{C})$ is not equal to this set of clauses, then we must discard the pair (α, θ).

If t is a leaf node, then we first determine which pairs (α, θ) may belong to $\mathscr{S}_t$. We do this by using Lemma 11, where we choose $\chi_v^*(t) = \chi_v(t) = \mathcal{X}_t$; the latter equality follows from the fact that t is a leaf. We then find that there can only exist a truth assignment $\tau \in N_t(\alpha, \theta)$ if the following three conditions hold for all $\mathcal{C} \in \chi_c(t)$.

(i) $\theta(\mathcal{C}) = \mathcal{C}(\tau) = \mathcal{C}(\tau|_{\langle \chi_v(t) \rangle}) = \mathcal{C}$ if $\mathcal{C}$ has no variable modules in $\chi_v(t)$.
(ii) $\theta(\mathcal{C}) = \emptyset$ if $\mathcal{C}$ has a variable module in $\chi_c(t)$ with $\alpha(X) = \emptyset$.
(iii) $\theta(\mathcal{C}) = \mathcal{C}(\tau) = \mathcal{C}(\tau|_{\langle \chi_v(t) \rangle}) = \mathbf{select}(\mathcal{C}, \alpha(X_1) \cdots \alpha(X_p))$ If $\mathcal{C}$ has exactly $p \geq 1$ variable modules $X_1, \ldots, X_p$ in $\chi_v(t)$ and $\alpha(X_i) \neq \emptyset$ for $i = 1, \ldots, p$.

Note that checking these three conditions for all $\mathcal{C} \in \chi_c(t)$ takes linear time for a given pair (α, θ). If these conditions are violated for some $\mathcal{C} \in \chi_c(t)$, then we discard the pair (α, θ). Otherwise, i.e., if these conditions are satisfied for all $\mathcal{C} \in \chi_c(t)$, then we apply Lemma 12. If we find that $n_t(\alpha, \theta) = 0$, then $(\alpha, \theta) \notin \mathscr{S}_t$ (as equivalence classes are nonempty by definition) and we discard this pair. If t is a introduce node that introduces a variable module, then we apply Lemma 13 (i). Here, we find that a pair $(\alpha, \theta) \in \mathscr{S}_t$ only if $n_t(\alpha, \theta) > 0$. If t is a introduce node that introduces a clause module $\mathcal{C}$, then we we first determine which pairs (α, θ) may belong to $\mathscr{S}_t$. We do this by using Lemma 11, where we choose $\chi_v^*(t) = \chi_v(t)$. Because $\mathcal{C}$ is introduced by t, we find that $\mathcal{C} \notin \mathcal{F}_t \setminus \chi_c(t)$. Hence, by definition of a tree decomposition, $\mathcal{C}$ contains no variable modules from $\mathcal{X}_t \setminus \chi_v(t)$. This means that $\mathcal{C}(\tau) = \mathcal{C}(\tau|_{\langle \chi_v(t) \rangle})$ should τ be a truth assignment in $N_t(\alpha, \theta)$. Lemma 11 tells us that this can only happen if the above conditions (i)–(iii) hold. Note that checking these three conditions for $\mathcal{C}$ takes linear time for a given pair (α, θ). If these conditions are satisfied for $\mathcal{C}$, then we apply Lemma 13 (ii). If we find that $n_t(\alpha, \theta) = 0$, then $(\alpha, \theta) \notin \mathscr{S}_t$ and we discard this pair. If t is a forget node that forgets a variable module or a clause module, then we apply Lemmas 14 (i) and (ii), respectively. If t is a join node, then we apply Lemma 15. In all these three cases, we find that a pair $(\alpha, \theta) \in \mathscr{S}_t$ only if $n_t(\alpha, \theta) > 0$.

As soon as we are of distance two from a node t, we can forget the sizes $n_t(\alpha, \theta)$. Recall that $|\mathscr{S}_t| \leq (|F| + 1)^k$ for all nodes $t \in V_T$ due to (1). This bound on the number of transferable equivalence classes for a node t has the following two consequences. First, as can be seen from the equations in Lemmas 12–15, it means that it takes at most $p(\ell)(|F|+1)^k(|F|+1)^k = (|F|+1)^{2k}$ time to compute the size $n_t(\alpha, \theta)$ of an equivalence class $N_t(\alpha, \theta)$, where $p(\ell)$ is a polynomial that only depends on ℓ, which also includes the additional time necessary to verify whether a pair (α, θ) may belong to $\mathscr{S}_t$ in case of Lemma 12 and Lemma 13 (ii). Second, it means that for each node we must compute and verify at most $(|F| + 1)^k$ sizes $n_t(\alpha, \theta)$. As the total number of nodes is at most 4ℓ, we find that the total running time is at most $4\ell \cdot (|F| + 1)^k \cdot p(\ell) \cdot (|F| + 1)^{2k} = \ell^{\mathcal{O}(k)}$. ◀

4 A (Parameterized) Hardness Result

The polynomial-time algorithm developed in the proof of Theorem 1 runs in time $\ell^{\mathcal{O}(k)}$ for formulas of length ℓ and modular incidence treewidth at most k. That is, the order of the polynomial depends on k. The question arises whether this dependency is necessary: *Is there a better algorithm with a running time of, say, $\mathcal{O}(\ell^c)$ where c is a constant independent of k?* We give a negative answer subject to the complexity theoretic assumption W[1] $\neq$ FPT from the area of Parameterized Complexity.

We briefly review basic concepts of Parameterized Complexity; for more information we refer to other sources [8, 11, 18]. An instance of a parameterized problem is a pair (x, k), where x is the *main part* and k (usually a non-negative integer) is the *parameter*. A parameterized problem is *fixed-parameter tractable* if it can be solved in time $\mathcal{O}(f(k)|x|^c)$ where f is a computable function and c is a constant independent of k. FPT denotes the class of all fixed-parameter tractable decision problems. Parameterized Complexity offers a completeness theory similar to the theory of NP-completeness for non-parameterized problems. A parameterized problem P *fpt-reduces* to a parameterized problem Q if we can transform an instance (x, k) of P into an instance (x', k') of Q with $k' \leq g(k)$ in time $\mathcal{O}(f(k)|x|^c)$ (f, g are arbitrary computable functions, c is a constant) such that (x, k) is a yes-instance of P if and only if (x', k') is a yes-instance of Q. A parameterized complexity class is the class of parameterized decision problems fpt-reducible to a certain parameterized decision problem. Of particular interest is the class W[1] which is considered as the parameterized analog to NP. For example, the Clique problem (given a graph G and an integer k, decide whether G contains a *k-clique* a complete subgraph on k vertices), parameterized by k, is well-known to be W[1]-complete. It is believed that FPT $\neq$ W[1], and there is strong theoretical evidence that supports this belief; for example, FPT = W[1] implies that the Exponential Time Hypothesis fails [11].

Ordyniak, Paulusma, and Szeider [19] showed that satisfiability is W[1]-hard when parameterized by the incidence β-hypertree width (see Section 1.1), using an fpt-reduction from the following problem, which is W[1]-complete [21] (a k-partite graph is *balanced* if its k partition classes are of the same size): The input is a balanced k-partite graph $G = (V_1, \ldots, V_k, E)$, the parameter is k. The question is whether G contains a k-clique.

Let $G = (V_1, \ldots, V_k)$ be a balanced k-partite graph for $k \geq 2$. The reduction [19] maps the instance (G, k) to an instance (F, k) such that k is an upper bound on the β-hypertree width of $I(F)$. By taking a closer look at the incidence graph $I(F)$, we will show that the modular incidence treewidth of F can also be bounded by a function of k. Let $V_i = \{v_1^i, \ldots, v_n^i\}$. The incidence graph $I(F)$ of F is structured as follows. One vertex class (corresponding to variables of F) contains the vertices of G plus new vertices z_j^i for $1 \leq i \leq k$ and $1 \leq j \leq n-1$. The other vertex class (corresponding to clauses of F) consists of vertices $C_{u,v}$ for $u \in V_i, v \in V_j$ $(i \neq j)$ and $uv \notin E(G)$ such that $N_{I(F)}(C_{u,v}) = V_i \cup V_j$. Moreover, for $1 \leq i \leq k$, it contains the vertices $D_1^i, \ldots, D_n^i$ with $N_{I(F)}(D_1^i) = \{z_1^i, v_1^i, v_2^i\}$, $N_{I(F)}(D_j^i) = \{z_j^i, z_{j-1}^i, v_{j+1}^i\}$ for $2 \leq j \leq n-1$ and $N_{I(F)}(D_n^i) = \{z_{n-1}^i\}$.

The set of vertices $C_{u,v}$ can be partitioned into modules $\mathcal{C}_1, \ldots, \mathcal{C}_m$, where $m \leq \binom{k}{2}$. By deleting these modules, we obtain a graph $I'(F)$ that consists of k connected components corresponding to the subgraphs of $I(F)$ induced by $\{v_1^i, \ldots, v_n^i, z_1^i, \ldots, z_{n-1}^i, D_1^i, \ldots, D_n^i\}$ for $1 \leq i \leq k$. Note that these component are trees, so the treewidth of $I'(F)$ is 1. Thus the graph obtained from $I'(F)$ by contracting modules has treewidth 1. We can turn the corresponding tree decomposition into a tree decomposition of $I^*(F)$ by simply adding the set of clause modules $\{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ to each bag. So the modular incidence treewidth of F is at most $m + 1 \leq \binom{k}{2} + 1$. This proves the following result.

▶ **Theorem 16.** *The* SATISFIABILITY *problem is* W[1]*-hard, when parameterized by an upper bound on the modular incidence treewidth of the input formula.*

That is, already deciding whether $\#(F) > 0$ for a formula F of length ℓ and bounded modular incidence treewidth cannot be done in time $\mathcal{O}(\ell^c)$ for constant c unless FPT = W[1] (where $\#(F)$ denotes the number of satisfying truth assignments of F). In particular, this implies a negative answer to the question raised at the beginning of this section.

5 Conclusion

In this paper, we proved that #SAT becomes polynomial-time tractable on formulas of bounded *modular incidence treewidth*. Modular incidence treewidth combines treewidth and *module contraction*, a powerful preprocessing technique widely used in combinatorial optimization. The resulting parameter is incomparable with the most general structural parameters for which #SAT is known to be tractable.

With this result, we approach the frontier of tractability from a new direction. On the other side, one can find incidence β-hypertree width and incidence clique-width. It remains open whether #SAT becomes tractable when these parameters are bounded. We think that this work is a significant step towards proving tractability of #SAT on formulas of bounded clique-width. Graphs of bounded clique-width that do not contain large bipartite subgraphs are known to have bounded treewidth [3]. This gives us reason to believe that our techniques carry over to the case of bounded incidence clique-width.

References

1 F. Bacchus, S. Dalmao, and T. Pitassi. Algorithms and complexity results for #SAT yand Bayesian inference. In *FOCS'03*, pages 340–351, 2003.

2 H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

3 B. Courcelle. The monadic second-order logic of graphs XIV: Uniformly sparse graphs and edge set quantifications. *Theoretical Computer Science*, 299(1–3):1 – 36, 2003.

4 B. Courcelle, J. Engelfriet, and G. Rozenberg. Handle-rewriting hypergraph grammars. *J. of Computer and System Sciences*, 46(2):218–270, 1993.

5 B. Courcelle, J. A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.

6 B. Courcelle, J. A. Makowsky, and U. Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discr. Appl. Math.*, 108(1-2):23–52, 2001.

7 B. Courcelle and S. Olariu. Upper bounds to the clique-width of graphs. *Discr. Appl. Math.*, 101(1-3):77–114, 2000.

8 R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer Verlag, 1999.

9 M. R. Fellows, F. A. Rosamond, U. Rotics, and S. Szeider. Clique-width is NP-complete. *SIAM J. Discrete Math.*, 23(2):909–939, 2009.

10 E. Fischer, J. A. Makowsky, and E. R. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discr. Appl. Math.*, 156(4):511–529, 2008.

11 J. Flum and M. Grohe. *Parameterized Complexity Theory.* Springer Verlag, 2006.

12 R. Ganian, P. Hlinený, and J. Obdrzálek. Better algorithms for satisfiability problems for formulas of bounded rank-width. In K. Lodaya and M. Mahajan, editors, *FSTTCS 2010*, volume 8 of *LIPIcs*, pages 73–83. 2010.

13 G. Gottlob and R. Pichler. Hypergraphs in model checking: Acyclicity and hypertree-width versus clique-width. In *ICALP 2001*, volume 2076 of *Lecture Notes in Computer Science*, pages 708–719, 2001.

14 M. Habib and C. Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010.

15 P. Hlinený and S. il Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008.

16 P. Kaski, M. Koivisto, and J. Nederlof. Homomorphic hashing for sparse coefficient extraction. In *IPEC 2012*, 2012.

17 T. Kloks. *Treewidth: Computations and Approximations.* Springer Verlag, 1994.

18 R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

19 S. Ordyniak, D. Paulusma, and S. Szeider. Satisfiability of acyclic and almost acyclic CNF formulas. In *FSTTCS 2010*, volume 8 of *LIPIcs*, pages 84–95. 2010.

20 S. Oum and P. Seymour. Approximating clique-width and branch-width. *J. Combin. Theory Ser. B*, 96(4):514–528, 2006.

21 K. Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *J. of Computer and System Sciences*, 67(4):757–771, 2003.

22 D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1-2):273–302, 1996.

23 M. Samer and S. Szeider. Algorithms for propositional model counting. *J. of Discrete Algorithms*, vol. 8, no. 1, pp. 50–64, 2010.

24 L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

Backdoors to q-Horn

Serge Gaspers[1], Sebastian Ordyniak[2], M. S. Ramanujan[3], Saket Saurabh[3], and Stefan Szeider[4]

1 **The University of New South Wales** and **National ICT Australia**
`sergeg@cse.unsw.edu.au`
2 **Masaryk University, Brno**
`sordyniak@gmail.com`
3 **The Institute of Mathematical Sciences, Chennai**
`{msramanujan | saket}@imsc.res.in`
4 **Institute of Information Systems, Vienna University of Technology**
`stefan@szeider.net`

Abstract

The class q-Horn, introduced by Boros, Crama and Hammer in 1990, is one of the largest known classes of propositional CNF formulas for which satisfiability can be decided in polynomial time. This class properly contains the fundamental classes of Horn and Krom formulas as well as the class of renamable (or disguised) Horn formulas. In this paper we extend this class so that its favorable algorithmic properties can be made accessible to formulas that are outside but "close" to this class. We show that deciding satisfiability is fixed-parameter tractable parameterized by the distance of the given formula from q-Horn. The distance is measured by the smallest number of variables that we need to delete from the formula in order to get a q-Horn formula, i.e., the size of a smallest deletion backdoor set into the class q-Horn. This result generalizes known fixed-parameter tractability results for satisfiability decision with respect to the parameters distance from Horn, Krom, and renamable Horn.

1998 ACM Subject Classification G.2.1, F.2.2

Keywords and phrases Algorithms and data structures. Backdoor sets. Satisfiability. Fixed Parameter Tractability.

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.67

1 Introduction

The satisfiability problem (SAT) is a well-known fundamental problem in Computer Science [3]. Many hard combinatorial problems including problems from the domains of hardware and software verification, Artificial Intelligence, planning and scheduling can be encoded as SAT instances [2, 4, 15, 17, 23]. However, the problem is known to be NP-hard and thus we cannot hope to solve it polynomial time [7]. In spite of this, over the last two decades, SAT-solvers have become quite successful in solving formulas with hundreds of thousands of variables that encode problems arising from various application areas (see, e.g., [14]), but theoretical performance guarantees are far from explaining this empirically observed efficiency. In fact, there is an enormous gap between theory and practice.

The discrepancy between theory and practice can be potentially explained by the presence of a certain "hidden structure" in real-world problem instances. One such "hidden structure" in real-world instances of SAT is the presence of *small backdoor sets* [24]. There are three variants of backdoor sets with respect to a particular base class $\mathcal{C}$ of polynomial-time decidable CNF formulas: strong $\mathcal{C}$-backdoor sets, where for each truth assignment to the backdoor variables, the reduced formula belongs to $\mathcal{C}$; deletion $\mathcal{C}$-backdoor sets, where deleting all backdoor variables and their negations from the formula moves the formula into the base class $\mathcal{C}$; and weak backdoor sets where, for at least one truth assignment to the backdoor variables, the reduced formula belongs to $\mathcal{C}$ and is satisfiable. Given a backdoor set of a formula with respect to a particular tractable base class $\mathcal{C}$, the satisfiability of the formula can be decided by guessing an assignment to the variables in the backdoor set and deciding the satisfiability of the reduced formula, which is guaranteed to be in $\mathcal{C}$, using a sub-solver for $\mathcal{C}$. An equivalent view of this is to consider the size of the backdoor set to be the "distance" of the formula from the class $\mathcal{C}$. The objective is to extend the favorable algorithmic properties of the class $\mathcal{C}$ to formulas which are "close" to this class. Ideally, we would want the class $\mathcal{C}$ to be as large as possible.

In a 1990 paper [5], Boros, Crama and Hammer introduced an interesting class of CNF formulas, later called q-Horn [6], with favorable algorithmic properties: both recognition as well as satisfiability decision of q-Horn formulas can be carried out in linear-time [5, 6]. This class q-Horn properly contains the fundamental classes of Horn and Krom formulas [22], and the class of renamable (or disguised) Horn formulas [16, 1]:

$$\text{Horn} \quad \subsetneq \quad \text{renamableHorn} \quad \subsetneq \quad \text{q-Horn} \quad \supsetneq \quad \text{Krom}.$$

The fact that this class is so large serves as an additional motivation for choosing it as our base class of interest. In this paper, we study the problem of finding small backdoor sets with respect to the class of q-Horn formulas and obtain algorithmic as well as hardness results.

1.1 Contribution

The main contribution of this paper is an algorithm that, given a CNF formula F of length ℓ with n variables and an integer $k \geq 0$, runs in time $O(6^k k\ell n)$, and either returns a deletion q-Horn-backdoor set for F of size at most $k^2 + k$, or concludes correctly that no such set of size at most k exists. As a consequence, we obtain that SAT is fixed-parameter tractable with the size of the smallest deletion q-Horn-backdoor set as the parameter, as we can use this algorithm to reduce the satisfiability problem of a CNF formula F of distance k from being q-Horn to testing the satsfiability of $2^{O(k^2)}$-many q-Horn formulas. Our result simultaneously generalizes the known fixed-parameter tractability results for SAT parameterized by the deletion distance from the class of renamable Horn formulas [20] and from the class of Krom formulas [19].

At the highest level, our algorithm works by finding a bounded number of variables whose deletion results in an instance with an optimal solution strictly smaller than that of the original instance. By repeatedly computing such a set and deleting it, we obtain the approximate solution. The main technical part of the paper is the algorithm to compute the bounded set of variables with the required properties. This algorithm relies on a characterization of q-Horn formulas in terms of their *quadratic cover* by Boros, Hammer, and Sun [6]. We use this characterization to model the problem of finding a small deletion q-Horn-backdoor set as a problem of *hitting* certain types of paths in an auxiliary digraph related to the formula. Using this characterization, we show that if we are guaranteed that an optimal solution hits all paths between a carefully chosen pair of vertices in this digraph, then we can compute in polynomial time a set of variables whose size is bounded by some $f(k)$ such that (a) there is a minimal (though not necessarily optimal) solution containing these variables and (b) deletion of these variables results in a formula whose solution is strictly smaller than the solution for the formula we started with. A standout feature of our algorithm is that at its core, it reduces to computing flows in a directed graph whose size is linear in the input size. As a result, our algorithm is quite efficient not only with respect to the dependence of the running time on the parameter, but also with respect to the dependence on the input size, along with having only a small hidden constant factor in the asymptotic running time. Finally, towards the end of the paper we also provide parameterized complexity results regarding the detection of weak and strong backdoor sets with respect to the class q-Horn.

1.2 Related Work

The parameterized complexity of finding small backdoor sets was initiated by Nishimura et al. [19] who showed that for the base classes of Horn formulas and Krom formulas, the detection of strong backdoor sets is fixed-parameter tractable. Their algorithms exploit the fact that for these two base classes, strong and deletion backdoor sets coincide, and that deletion backdoor sets with respect to Horn and Krom can be characterized in terms of vertex covers and hitting sets of certain graphs and 3-uniform hypergraphs associated with the input formula, respectively. For base classes other than Horn and Krom, strong backdoor sets can be much smaller than deletion backdoor sets, and their detection is more difficult. In particular, for the base classes of renamable Horn and q-Horn, there are formulas that have a strong backdoor set of size 1 but require an arbitrarily large deletion backdoor set. In fact, Razgon and O'Sullivan [20] showed that the detection of deletion backdoor sets with respect to the base class renamable Horn is fixed-parameter tractable although the detection of strong backdoor sets is W[2]-hard [13]. For more recent results, the reader is referred to a survey on the parameterized complexity of backdoor sets [13].

2 Preliminaries

2.1 Formulas

We assume an infinite supply of propositional *variables*. A *literal* is a variable x or a negated variable $\bar{x}$; if $y = x$ or $y = \bar{x}$ is a literal for some variable x, then we write $\bar{y}$ to denote $\bar{x}$ or x, respectively. For a set S of literals we put $\bar{S} = \{\, \bar{x} : x \in S \,\}$; S is *consistent* if $S \cap \bar{S} = \emptyset$. A *clause* is a finite consistent set of literals; we consider a clause as a disjunction of its literals. A finite set of clauses is a *CNF formula* (or *formula*, for short); we consider a formula to be the conjunction of its clauses. A formula is *Horn* if each of its clauses contains at most one positive literal, a formula is *Krom* (or *2CNF*, or *quadratic*) if each clause contains at most two literals. A variable x *occurs* in a clause C if $x \in C \cup \bar{C}$; $\mathrm{var}(C)$ denotes the set of variables which occur in C. For a set X of variables, $\mathrm{lit}(X)$ denotes the set of literals of the variables in X, that is, $\mathrm{lit}(X) = X \cup \bar{X}$ and for a set L of literals, $\mathrm{var}(L)$ denotes the set of variables whose literals are in L, that is, $\mathrm{var}(L) = \{\, x : x \in L \text{ or } \bar{x} \in L \,\}$. A variable x *occurs* in a formula F if it occurs in one of its clauses, and we let $\mathrm{var}(F) = \bigcup_{C \in F} \mathrm{var}(C)$ and $\mathrm{lit}(F) = \mathrm{var}(F) \cup \overline{\mathrm{var}(F)}$. The *length* of a CNF formula F, denoted by $\|F\|$, is defined as $\sum_{C \in F} |C|$. If F is a formula and X a set of variables, then we denote by $F - X$ the formula obtained from F after removing all literals in $\mathrm{lit}(X)$ from the clauses in F. If $X = \{x\}$ we simply write $F - x$ instead of $F - \{x\}$.

Let F be a formula and $X \subseteq \mathrm{var}(F)$. A *truth assignment* is a mapping $\tau : X \to \{\, 0, 1 \,\}$ defined on some set X of variables; we write $\mathrm{var}(\tau) = X$. For $x \in \mathrm{var}(\tau)$ we define $\tau(\bar{x}) = 1 - \tau(x)$. For a truth assignment τ and a formula F, we define $F[\tau] = \{\, C \setminus \tau^{-1}(0) : C \in F,\ C \cap \tau^{-1}(1) = \emptyset \,\}$, i.e., $F[\tau]$ denotes the result of instantiating variables according to τ

and applying the usual simplifications, i.e., removing clauses that are satisfied by τ and removing unsatisfied literals from clauses. A truth assignment τ *satisfies* a clause C if C contains some literal x with $\tau(x) = 1$; τ satisfies a formula F if it satisfies all clauses of F. A formula is *satisfiable* if it is satisfied by some truth assignment; otherwise it is *unsatisfiable*.

The SATISFIABILITY (SAT) problem asks whether a given CNF formula is satisfiable.

2.2 Parameterized Complexity

An instance of a parameterized problem is a pair (I, k) where I is the *main part* and k is the *parameter*; the latter is usually a non-negative integer. A parameterized problem is *fixed-parameter tractable* if there exist a computable function f and a constant c such that instances (I, k) can be solved in time $O(f(k)\|I\|^c)$ where $\|I\|$ denotes the size of I. FPT is the class of all fixed-parameter tractable decision problems and algorithms which run in the time specified above are called FPT algorithms.

An FPT-*reduction* is a many-one reduction where the parameter for one problem maps into the parameter for the other. More specifically, given two parameterized decision problems L and L', problem L reduces to problem L' if there is a mapping R from instances of L to instances of L' such that (i) (I, k) is a yes-instance of L if and only if $(I', k') = R(I, k)$ is a yes-instance of L', (ii) $k' \leq g(k)$ for a computable function g, and (iii) R can be computed in time $O(f(k)\|I\|^c)$ where f is a computable function and c is a constant.

The *Weft Hierarchy* consists of parameterized complexity classes $\mathrm{W}[1] \subseteq \mathrm{W}[2] \subseteq \cdots$ which are defined as the closure of certain parameterized problems under FPT-reductions (see [9, 11] for definitions). There is strong theoretical evidence that parameterized problems that are hard for classes $\mathrm{W}[i]$ are not fixed-parameter tractable. For example $\mathrm{FPT} = \mathrm{W}[1]$ implies that the Exponential Time Hypothesis (ETH) fails; that is, $\mathrm{FPT} = \mathrm{W}[1]$ implies the existence of a $2^{o(n)}$ algorithm for n-variable 3SAT [11].

An *FPT-approximation algorithm* with ratio ρ for a minimization problem P is an FPT algorithm that, given an instance x of P and a positive integer k, either determines that there is no solution of size at most k or computes a solution of size at most $k\rho(k)$ (see, e.g., [10]). The definition can be adapted to maximization problems. Note that the approximation ratio ρ is a function of k and not the input size: intuitively, if k is small, then $k\rho(k)$ can be still considered small. We say that a problem is *FPT-approximable* if it has an FPT-approximation algorithm for some function ρ.

2.3 Backdoors

Here, we introduce the basic terminology for backdoors and the class of q-Horn formulas. For further information on backdoors and other tractable base classes of SATISFIABILITY we refer the reader to [13].

Backdoors are defined with respect to a fixed class $\mathcal{C}$ of CNF formulas, the *base class* (or *target class*, or more figuratively, *island of tractability*). We say a class $\mathcal{C}$ of formulas is *clause-induced* if it is closed under subsets, i.e., if $F \in \mathcal{C}$ implies $F' \in \mathcal{C}$ for each $F' \subseteq F$.

A *strong $\mathcal{C}$-backdoor set* of a CNF formula F is a set B of variables such that $F[\tau] \in \mathcal{C}$ for each assignment $\tau : B \to \{0, 1\}$. A *weak $\mathcal{C}$-backdoor set* of F is a set B of variables such that $F[\tau]$ is satisfiable and $F[\tau] \in \mathcal{C}$ holds for some assignment $\tau : B \to \{0, 1\}$. A *deletion $\mathcal{C}$-backdoor set* of F is a set B of variables such that $F - B \in \mathcal{C}$. Backdoor sets where independently introduced by Crama et al. [8] and by Williams et al. [24], the latter authors coined the term "backdoor".

If we know a strong $\mathcal{C}$-backdoor set of F of size k, we can reduce the satisfiability of F to the satisfiability of 2^k formulas in $\mathcal{C}$. Thus SAT becomes fixed-parameter tractable with k as the parameter. If we know a weak $\mathcal{C}$-backdoor set of F, then F is clearly satisfiable, and we can verify it by trying for each $\tau \in 2^k$ whether $F[\tau]$ is in $\mathcal{C}$ and satisfiable. If $\mathcal{C}$ is clause-induced, every deletion $\mathcal{C}$-backdoor set of F is a strong $\mathcal{C}$-backdoor set of F. For several base classes, deletion backdoor sets are of interest because they are easier to detect than strong backdoor sets. The challenging problem is to find a strong, weak, or deletion $\mathcal{C}$-backdoor set of size at most k if it exists. For each class $\mathcal{C}$ of CNF formulas, the various backdoor detection problems are defined as follows.

DELETION $\mathcal{C}$-BACKDOOR SET DETECTION	**Parameter:** k
Input: A CNF formula F and a positive integer k	
Question: Does F have a deletion $\mathcal{C}$-backdoor set of size at most k?	

2.4 q-Horn Formulas

In this paper we are mainly interested in the class of q-Horn formulas [5, 6]. A CNF formula F is in this class if there is a *certifying function* $\beta : \mathrm{var}(F) \cup \overline{\mathrm{var}(F)} \to \{0, \frac{1}{2}, 1\}$ with $\beta(x) = 1 - \beta(\bar{x})$ for every $x \in \mathrm{var}(F)$ such that $\sum_{l \in C} \beta(l) \leq 1$ for every clause C of F.

In the following sense, strong q-Horn-backdoor sets are more general than deletion q-Horn-backdoor sets: For every positive integer n there is a formula F_n such that F_n has a strong q-Horn-backdoor set of size 1 but every deletion q-Horn-backdoor set of F has size at least n. To see this, take for instance $F = \bigcup_{1 \leq i \leq n} \{\{x_i, y_i, z_i, a\}, \{\bar{x}_i, \bar{y}_i, \bar{z}_i, \bar{a}\}\}$. Evidently, $\{a\}$ is a strong q-Horn-backdoor set of F. However, every deletion q-Horn-backdoor set of F must contain at least one variable x_i, y_i, or z_i for every $1 \leq i \leq n$.

3 FPT-approximation for DELETION q-Horn BACKDOOR SET DETECTION

In this section we prove our main result:

▶ **Theorem 1.** *There is an algorithm that, given an instance (F, k) of* DELETION q-Horn BACKDOOR SET DETECTION, *runs in time $O(6^k k \ell n)$ and either correctly concludes that F has no deletion* q-Horn-*backdoor set of size at most k or returns a deletion* q-Horn-*backdoor set of F of size at most $k^2 + k$, where ℓ is the length of F and n is the number of variables in F.*

3.1 Quadratic covers, implication graphs and separators

In this subsection we give some definitions regarding quadratic covers, implication graphs and separators in implication graphs, which will be required for the description of our algorithm. The following definition of the quadratic cover of a CNF formula was used Boros et al. [6] to give a linear time algorithm to recognize q-Horn formulas.

▶ **Definition 2.** Given a CNF formula F, the **quadratic cover** of F, is a Krom formula denoted by F_2 and is defined as follows. Let $x_1, \dots, x_n$ be the variables of F. For every clause C, we have $|C| - 1$ new variables $y_1^C, \dots, y_{|C|-1}^C$. We order the literals in each clause according to their variables, that is, a literal of x_i will occur before a literal of x_j if $i < j$. Let $l_1^C, \dots, l_{|C|}^C$ be the literals of the clause C in this order. The quadratic cover is defined as

$$F_2 = \bigcup_{C\in F}\bigcup_{1\le i\le |C|-1}\{\{l_i^C, y_i^C\}, \{\bar{y}_i^C, l_{i+1}^C\}\} \quad \cup \quad \bigcup_{C\in F}\bigcup_{1\le i\le |C|-2}\{\{\bar{y}_i^C, y_{i+1}^C\}\}.$$

▶ **Definition 3.** Given a CNF formula F, the **implication graph** of F_2 is denoted by $\mathrm{D}(F_2)$ and defined as follows. The vertex set of the graph is the set of literals of F_2 and for every clause $\{l_1, l_2\}$ in F_2, we have arcs $(\bar{l}_1, l_2)$ and $(\bar{l}_2, l_1)$. We refer to the vertices of the implication graph as literals since there is a one to one correspondence between the two. Given a set $X \subseteq var(F)$ of variables, we define the graph $\mathrm{D}(F_2){-}X$ as the graph obtained from $\mathrm{D}(F_2)$ by deleting $\mathrm{lit}(X)$.

The following observations are direct consequences of the definition of an implication graph.

▶ Observation 1. Let F be a CNF formula of length ℓ.
(a) If there is a path from l_1 to l_2 in $\mathrm{D}(F_2)$, then there is also a path from $\bar{l}_2$ to $\bar{l}_1$ in $\mathrm{D}(F_2)$.
(b) The number of arcs in $\mathrm{D}(F_2)$ is $O(\ell)$.
(c) Let $C = \{l_1, \ldots, l_r\}$ be a clause of F. Then, for any $1 \le i < j \le r$, $\mathrm{D}(F_2)$ contains a path from $\bar{l}_i$ to l_j and from $\bar{l}_j$ to l_i whose internal vertices are all disjoint from $\mathrm{lit}(F)$.
(d) Let $X \subseteq var(F)$ and $F' = F - X$. Then, for any literal $l \in \mathrm{lit}(F) \setminus \mathrm{lit}(X)$, there is a path from l to $\bar{l}$ in $\mathrm{D}(F'_2)$ if and only if there is a path from l to $\bar{l}$ in $\mathrm{D}(F_2){-}X$.

▶ **Definition 4.** Given a CNF formula F and a set L of literals of F, we denote by $N_F^+(L)$ the set of literals in $\mathrm{lit}(F) \setminus L$ which can be reached from L in $\mathrm{D}(F_2)$ via a path whose internal vertices are disjoint from $\mathrm{lit}(F)$.

▶ **Definition 5.** ([6]) Given a CNF formula F, define a **canonical function** $\hat{\beta} : \mathrm{lit}(F) \to \{0, \frac{1}{2}, 1\}$ as follows. Consider a topological ordering π of the strongly connected components of $\mathrm{D}(F_2)$. For every literal $l \in \mathrm{lit}(F)$ such that the strongly connected component containing l appears before the one containing $\bar{l}$ in π, set $\hat{\beta}(l) = 1$ and for every literal l such that the strongly connected component containing l also contains $\bar{l}$, set $\hat{\beta}(l) = \frac{1}{2}$.

▶ **Lemma 6.** ([6]) *A CNF formula F is* q-Horn *if and only if the function $\hat{\beta}$ defined above is a certifying function for F.*

▶ **Definition 7.** A clause C of a given CNF formula is called a **violating clause** if $\sum_{l\in C}\hat{\beta}(l) > 1$. Any three literals l_1, l_2, l_3 of a violating clause such that $\sum_{i=1}^{3}\hat{\beta}(l_i) > 1$ form a **violating triple**.

▶ **Lemma 8.** *Let F be a CNF formula of length ℓ and suppose that F is not a* q-Horn *formula. Any violating clause of F has a violating triple lying entirely inside a strongly connected component of D(F_2) and we can compute such a violating triple in time $O(\ell)$.*

Because of space constraints we omit the easy proof of this lemma.
We now move on to some definitions on separators in implication graphs which will be required in the description of our algorithm.

▶ **Definition 9.** Let F be a CNF formula and $L \subseteq \mathrm{lit}(F)$ be a consistent set of literals. We say that a set $J \subseteq \mathrm{lit}(F)$ is an L-$\bar{L}$ **separator** if J is disjoint from L and $\bar{L}$ and there is no path from L to $\bar{L}$ in the graph $\mathrm{D}(F_2){-}J$. We say that J is a minimal L-$\bar{L}$ separator if no proper subset of J is an L-$\bar{L}$ separator.

▶ **Definition 10.** Let F be a CNF formula, $L \subseteq \mathrm{lit}(F)$ be a consistent set of literals and let X be a set of variables of F. We call X an L-$\bar{L}$ **variable separator** if $\mathrm{lit}(X)$ is an L-$\bar{L}$ separator. We call X a minimal L-$\bar{L}$ variable separator if no proper subset of X is an L-$\bar{L}$ variable separator. We drop the word *variable* if it is clear from the context that the set we are dealing with is a set of variables.

▶ **Definition 11.** Let F be a CNF formula, $L \subseteq \text{lit}(F)$ be a consistent set of literals and X be an L-$\bar{L}$ variable separator. We denote by $R(L, X)$ the set of literals of F that can be reached from L via directed paths in $\text{D}(F_2){-}X$, and we denote by $\bar{R}(L, X)$ the set of literals of F which have a directed path to L in $\text{D}(F_2){-}X$.

We also require the following observation.

▶ Observation 2. Let F be a CNF formula, $L \subseteq \text{lit}(F)$ be a consistent set of literals and X be an L-$\bar{L}$ variable separator. Then, the sets $R(L, X)$ and $\bar{R}(\bar{L}, X)$ are also consistent and in fact complements of each other.

3.2 The algorithm

We begin with the following simple lemma.

▶ **Lemma 12.** *Let (F, k) be an instance of* Deletion q-Horn Backdoor Set Detection*. Let (l_1, l_2, l_3) be a violating triple in a strongly connected component of $D(F_2)$ and X be a solution for the given instance disjoint from $\{var(l_1), var(l_2), var(l_3)\}$. Then, for some $1 \le i \le 3$, X is an l_i-$\bar{l}_i$ separator in $D(F_2)$.*

Proof. Let $\hat{\beta}'$ be the canonical certifying function for $F' = F - X$ obtained from the graph $\text{D}(F'_2)$. We claim that there is an $1 \le i \le 3$ such that $\hat{\beta}'(l_i) = 0$. This is true since F' contains a clause with all three literals l_1, l_2 and l_3 and it cannot be the case that any certifying function sets non zero values to all three. By definition of $\hat{\beta}'$, $\hat{\beta}'(l_i) = 0$ implies that there is no path from l_i to $\bar{l}_i$ in the graph $\text{D}(F'_2)$. If X were not an l_i-$\bar{l}_i$ separator in $\text{D}(F_2)$, then $\text{D}(F'_2)$ would also contain an l_i-$\bar{l}_i$ path (by Observation 1(d)), a contradiction. This completes the proof of the lemma. ◀

Lemma 8 combined with Lemma 12 allows us to compute in linear time, a set of three literals such that for every solution X one of the three corresponding variables is part of X or for at least one of these literals, say l, there is a path from l to $\bar{l}$ in $\text{D}(F_2)$ and X an l-$\bar{l}$ variable separator in $\text{D}(F_2)$.

▶ **Lemma 13.** *Let (F, k) be an instance of* Deletion q-Horn Backdoor Set Detection *and X be a solution such that it is disjoint from $var(l)$ and is an l-$\bar{l}$ separator for some literal $l \in lit(F)$. Consider an l-$\bar{l}$ variable separator X'. Let X'' be the set of variables of X with a literal in $R(l, X')$. Then, the set $\tilde{X} = (X \setminus X'') \cup X'$ is also a deletion* q-Horn*-backdoor set for the given instance.*

Proof. Let $F' = F - X$ and $\tilde{F} = F - \tilde{X}$. If $\tilde{X}$ were not a deletion q-Horn-backdoor set, then there is a violating clause in $\tilde{F}$ and by Lemma 8, there is a violating triple (l_1, l_2, l_3) in a strongly connected component of $\text{D}(\tilde{F}_2)$. This implies the presence of a closed walk in $\text{D}(\tilde{F}_2)$ containing all the literals of the violating triple and their complements (by Lemma 8). Since X was a solution, this closed walk could not have survived in $\text{D}(F'_2)$ and hence must contain a literal of a variable in $X \setminus \tilde{X}$. Recall that the only variables of X that are not in $\tilde{X}$ are those in X''. Let p be a literal on this closed walk which corresponds to such a variable, that is, $\text{var}(p) \in X''$. On the other hand, by definition, the literals of the variables in X'' can either reach $\bar{l}$ or be reached from l in $\text{D}(\tilde{F}_2)$, that is, they must lie in $R(l, \tilde{X})$ or $\bar{R}(\bar{l}, \tilde{X})$. Combining this path along with the closed walk and the fact that $\text{D}(\tilde{F}_2)$ is an implication graph implies the presence of a path from l to $\bar{l}$ in $\text{D}(\tilde{F}_2)$. However, by construction, $\tilde{X}$ is also an l-$\bar{l}$ separator in $\text{D}(F_2)$. Observation 1(d) implies that this is a contradiction. This completes the proof of the lemma. ◀

▶ **Lemma 14.** *Let (F, k) be an instance of* Deletion q-Horn Backdoor Set Detection *where F is a CNF formula of length ℓ, with n variables. Let X be a solution to the given instance and let l be a literal of F such that there is an l-$\bar{l}$ path in $D(F_2)$. Furthermore, suppose that X is an l-$\bar{l}$ variable separator. Then, there is an algorithm that, given F, k and l, runs in time $O(k\ell n)$ and either concludes correctly that there is no k-sized l-$\bar{l}$ variable separator in $D(F_2)$ or returns an l-$\bar{l}$ variable separator X' of size at most $2k$ such that $(X' \cup var(R(l, X'))) \cap X$ is non-empty*

Proof. We show that Algorithm 3.1 has the stated properties. The algorithm computes an l-$\bar{l}$ variable separator X' which essentially maximizes the set of literals of $\mathrm{D}(F_2)$ reachable from l after removing X'. We will then show that such a separator indeed has the required properties.

If it the algorithm returns No in Line 4, then $\mathrm{D}(F_2)$ has no l-$\bar{l}$ variable separator of size at most k. Let S be the minimal separator in $\mathrm{D}(F_2)$ which was computed in the penultimate iteration of the while loop. We claim that $X' = \mathrm{var}(S)$ satisfies the conditions in the statement of the lemma. Clearly, it must be the case that for some choice of a literal l' in $\mathrm{lit}(\mathrm{var}(S)) \cap N_F^+(L)$, the next iteration of the loop could not find an $L \cup \{l'\}$-$\bar{L} \cup \{\bar{l'}\}$ separator of size at most $2k$.

Suppose that $(X' \cup \mathrm{var}(R(l, X'))) \cap X$ is empty. Recall that when the procedure stops, $L = R(l, X')$. Furthermore, if there is at least one path from l to $\bar{l}$ in $\mathrm{D}(F_2)$ then it must be the case that $\mathrm{lit}(\mathrm{var}(S)) \cap N_F^+(L)$ is non-empty. Since X is an l-$\bar{l}$ separator and disjoint from L, X is also an L-$\bar{L}$ separator. Since X is also disjoint from X', for any $l' \in \mathrm{lit}(\mathrm{var}(S)) \cap N_F^+(L)$, X intersects all paths from $L \cup \{l'\}$ to $\bar{L} \cup \{\bar{l'}\}$. Hence, $\mathrm{lit}(X)$ is a set of size at most $2k$ which intersects all $L \cup \{l'\}$-$\bar{L} \cup \{\bar{l'}\}$ paths, which is a contradiction. Therefore, the set $(X' \cup \mathrm{var}(R(l, X'))) \cap X$ is non-empty for any l-$\bar{l}$ variable separator X of size at most k.

To bound the running time, observe that in each iteration, we only need to test if there is an L-$\bar{L}$ separator of size at most $2k$. Hence, it suffices for us to run the Ford-Fulkerson algorithm [12] for at most $2k$ steps on the graph $\mathrm{D}(F_2)$ and the number of iterations is bounded by the number of variables in the formula since in each iteration, we add a literal to L. Since the number of arcs in $\mathrm{D}(F_2)$ is $O(\ell)$ (Observation 1(b)), the claimed time bound follows. This completes the proof of the lemma. ◀

▶ **Lemma 15.** *Let (F, k) be an instance of* Deletion q-Horn Backdoor Set Detection *and let l be a literal of F disjoint from a solution X and suppose that X is an l-$\bar{l}$ variable separator in $D(F_2)$. Consider an l-$\bar{l}$ variable separator in $D(F_2)$, X', such that $(X' \cup var(R(l, X'))) \cap X$ is non-empty. Then, the instance $F - X'$ has a deletion* q-Horn*-backdoor set of size at most $|X| - 1$.*

Proof. By Lemma 13, we know that the set $\hat{X} = (X \setminus X'') \cup X'$ is a deletion q-Horn-backdoor set. Hence, $X \setminus (X'' \cup X')$ is indeed a deletion q-Horn-backdoor set for the instance $F - X'$. Since $(X' \cup X'') \cap X$ is non-empty, the size of $X \setminus (X'' \cup X')$ is at most $|X| - 1$. This completes the proof of the lemma. ◀

Lemmas 14 and 15 allow us to compute a bounded set of variables whose deletion from the formula results in an instance that has a solution which is strictly smaller than any solution of the input instance. This completes the formalization of our ideas and we are now ready to prove Theorem 1 by describing our algorithm for Deletion q-Horn Backdoor Set Detection.

```
Input  : A tuple $(F, k, l)$ where $F$ is a CNF formula, $k$ a positive integer and $l$ a literal of $F$
Output: NO provided that $\mathrm{D}(F_2)$ has no $l$-$\bar{l}$ variable separator of size at most $k$, or an $l$-$\bar{l}$
        variable separator $S$ of size at most $2k$ such that $(S \cup \mathrm{var}(R(l, S)))$ has non-empty
        intersection with some minimum deletion q-Horn-backdoor set
1  if there is an $l$-$\bar{l}$ separator of size at most $2k$ in $D(F_2)$ then
2      $S \leftarrow$ such a separator
3  end
4  else return NO
5  $L \leftarrow R(l, \mathrm{var}(S))$ // $L$ is consistent by Observation 2
6  while there is an $L \cup \{l'\}$-$\bar{L} \cup \{\bar{l'}\}$ separator of size at most $2k$ where
   $l' \in (lit(var(S)) \cap N^+_F(L))$ is an arbitrarily chosen such literal do
7      $S \leftarrow$ such a separator
8      $L \leftarrow R(L, \mathrm{var}(S))$
9  end
10 return $\mathrm{var}(S)$
```

Algorithm 3.1: Algorithm COMPUTE-SEPARATOR

3.2.1 Description of the Algorithm

Algorithm 3.2 checks whether there is a violating triple and if so, computes one and in the first 3 branches, it adds the variable corresponding to each of the literals of the violating triple to the solution, deletes it from the formula and recurses on the resulting instance with a budget of $k - 1$. In each of the next 3 branches, it picks a literal of the violating triple and continues by assuming that this literal is assigned 0 by a certifying function of $F - X$ where X is a solution. We know that there must be at least one such literal (see the proof of Lemma 12) in the violating triple. This implies that X is an l-$\bar{l}$ separator for the literal l in the violating triple which is assigned 0 by a certifying function of $F - X$. Finally, Lemma 14 is used to either conclude that there is no l-$\bar{l}$ variable separator of size at most k in which case the algorithm returns NO, or to compute an l-$\bar{l}$ variable separator of size at most $2k$ with the required properties. The variables in X' are added to our proposed approximate solution and deleted from the formula, and the algorithm recurses on the resulting instance with a budget of $k - 1$.

3.2.2 Analysis

Since Steps 2, 4, and 10 at any node of the search tree take time $O(k\ell n)$ and we have a 6-way branching at each node of the search tree with the budget k dropping by 1 in each branch, the algorithm clearly runs in the claimed time bound. Therefore, it only remains for us to prove the correctness of the algorithm. Let X be a solution for the given instance and let β be a certifying function for $F - X$. We prove the correctness of the algorithm by induction on k.

In the base case, when $k = 0$, the algorithm is correct by Lemma 6. We assume as induction hypothesis that the algorithm is correct for all values of k up to some $k' - 1$ where $k' - 1 > 0$. We now consider the case when $k = k'$.

In Lines 5–8, we consider the case when X intersects the set $\{\mathrm{var}(l_1), \mathrm{var}(l_2), \mathrm{var}(l_3)\}$ and branch accordingly. Applying the induction hypothesis, the size of any returned solution in a subsequent recursive call is at most $(k-1)^2 + (k-1)$. Hence, the size of a solution returned here is bounded by $1 + (k-1)^2 + (k-1) \leq k^2 + k$.

```
Input   : A CNF formula $F$ of length $\ell$ with $n$ variables, integer $k$
Output: Either no solution of size at most $k$ or a solution of size at most $k^2 + k$ for the
        instance $(F, k)$ of Deletion q-Horn Backdoor Set Detection
if $k < 0$ then return No
check for a violating clause by computing $D(F_2)$ and a topological ordering of $D(F_2)$
if there is no violating clause then return $\emptyset$
Compute a violating triple $(l_1, l_2, l_3)$
for $l = l_1, l_2, l_3$ do
    $S_1 \leftarrow$ DELETION-QHORN-BSD$(F - \{var(l)\}, k - 1)$
    if $S_1$ is not No then return $S_1 \cup \{var(l)\}$
end
for $l = l_1, l_2, l_3$ do
    $S \leftarrow$ COMPUTE-SEPARATOR$(F, k, l)$
    if $S$ is No then return No else
        $S_1 \leftarrow$ DELETION-QHORN-BSD$(F - \{S\}, k - 1)$
    end
    if $S_1$ is not No then return $S_1 \cup \{S\}$
end
return No
```

Algorithm 3.2: Algorithm DELETION-QHORN-BSD

In Lines 9–15, we consider the case when X is disjoint from the set of variables corresponding to l_1, l_2 and l_3. Since l_1, l_2, l_3 lie in the same clause and none of their corresponding variables are in X, by Lemma 12, X is an l_i-$\bar{l}_i$ separator for at least one of the literals l_i. Let us assume that this literal is l_1. In Line 10, we apply Lemma 14 to compute an l_1-$\bar{l}_1$ separator S of size at most $2k$ and add it to the solution we are constructing. By Lemma 15, we know that there is a solution for the instance $F - S$ of size at most $|X| - 1$. Hence, by the induction hypothesis, we obtain a solution of size at most $(k-1)^2 + (k-1)$ from the subsequent recursive call and adding to it the set S of size at most $2k$ results in a solution of size at most $k^2 + k$, which proves the correctness of the algorithm, completing the proof of Theorem 1.

In order to test the satisfiability of a given CNF formula F, it suffices to first compute a smallest deletion q-Horn-backdoor set of F and for each assignment to this set, test the satisfiability of the reduced formula which is q-Horn. Since testing satisfiability of a q-Horn formula is linear time [5], Theorem 1 has the following corollary.

▶ **Corollary 16.** *There is an algorithm that, given a formula F of length ℓ with n variables, runs in time $2^{O(k^2)}\ell n$ and decides the satisfiability of F, where k is the size of the smallest deletion* q-Horn-*backdoor set of F.*

4 Hardness

In this section we show that there is no FPT algorithm for Strong q-Horn-backdoor Set Detection or Weak q-Horn-backdoor Set Detection unless FPT=W[2]. In order to show this, we begin from the following problem, which is well-known to be W[2]-complete [9].

Hitting Set **Parameter:** k
Input: A set E of elements, a family $\mathcal{S}$ of finite subsets of E, and an integer $k > 0$.
Question: Does $\mathcal{S}$ have a hitting set, i.e., a subset H of E such that $H \cap S \neq \emptyset$ for every $S \in \mathcal{S}$, of size at most k?

▶ **Theorem 17.** STRONG q-Horn-BACKDOOR SET DETECTION *is* W[2]*-hard.*

Proof. We prove the theorem via an FPT-reduction from HITTING SET. Let $(E, \mathcal{S}, k)$ be an instance of HITTING SET. We construct a formula F that has a strong q-Horn-backdoor set of size at most k if and only if $\mathcal{S}$ has a hitting set of size at most k. The formula F has two clauses $P_S^i = S \cup \{x_i, y_i, z_i\}$ and $N_S^i = \bar{E} \cup \{\bar{x}_i, \bar{y}_i, \bar{z}_i\}$ for every $S \in \mathcal{S}$ and $1 \leq i \leq k+1$. Note that $\mathrm{var}(F) = E \cup \{\, x_i, y_i, z_i : 1 \leq i \leq k+1 \,\}$. Furthermore, for any S and for any $1 \leq i \leq k+1$, the formula comprising the two clauses P_S^i and N_S^i is clearly not q-Horn. It is not hard to verify that $\mathcal{S}$ has a hitting set of size at most k if and only if F has a strong q-Horn-backdoor set of size at most k. ◀

▶ **Theorem 18.** WEAK q-Horn-BACKDOOR SET DETECTION *is* W[2]*-hard, even for* 3*-CNF formulas.*

Proof. We prove the theorem via an FPT-reduction from HITTING SET. Let $(E, \mathcal{S}, k)$ be an instance of HITTING SET. We construct a 3-CNF formula F that has a weak q-Horn-backdoor set of size at most k if and only if $\mathcal{S}$ has a hitting set of size at most k. For every $S \in \mathcal{S}$ with $S = \{s_1, \dots, s_{|S|}\}$, every $1 \leq i \leq |S|$, and every $1 \leq j \leq k+1$ the formula F contains the clauses $\{z_i^j(S), \bar{s}_i, \bar{z}_{i+1}^j(S)\}$, $\{\bar{z}_1^j(S), z_{|S|+1}^j(S)\}$, $\{\bar{z}_1^j(S), \bar{z}_{|S|+1}^j(S)\}$, $\{z_1^j(S), z_{|S|+1}^j(S)\}$, $\{z_{|S|+1}^j(S), a^j(S), b^j(S)\}$, and $\{\bar{a}^j(S), \bar{b}^j(S)\}$. Note that $\mathrm{var}(F) = E \cup \{\, z_i^j(S) : S \in \mathcal{S}$ and $1 \leq i \leq |S|+1$ and $1 \leq j \leq k+1 \,\} \cup \{\, a^j(S), b^j(S) : S \in \mathcal{S}$ and $1 \leq j \leq k+1 \,\}$. Note furthermore that F is satisfiable by the assignment τ_{SAT} that sets the variables in $\{\, z_{|S|+1}^j(S), a^j(S) : S \in \mathcal{S}$ and $1 \leq j \leq k+1 \,\}$ to 1 and all other variables to 0. It is not hard to verify that $\mathcal{S}$ has a hitting set of size at most k if and only if F has a weak q-Horn-backdoor set of size at most k. ◀

It remains an open problem whether STRONG q-Horn-BACKDOOR SET DETECTION or WEAK q-Horn-BACKDOOR SET DETECTION are FPT-approximable. However we note that since the reductions used in the above theorems are parameter preserving, an FPT-approximation algorithm for either of these problems would imply the existence of an FPT-approximation algorithm for HITTING SET, which is an open problem [18].

5 Conclusions

In this paper we have developed an FPT-approximation algorithm for the detection of deletion q-Horn-backdoor sets (Theorem 1). This renders SAT, parameterized by the *deletion distance* from the class of q-Horn-formulas (i.e., the size of a smallest deletion q-Horn-backdoor set) fixed-parameter tractable (Corollary 16). Our result simultaneously generalizes the known fixed-parameter tractability results for SAT parameterized by the deletion distance from the class of renamable Horn formulas [20] and from the class of Krom formulas [19]. We would like to point out that our FPT-approximation algorithm is quite efficient, and its asymptotic running time does not include large hidden factors.

The deletion distance from q-Horn is incomparable with parameters for SAT based on width measures such as the treewidth of the formula's primal, dual, or incidence graph [21]. This can be easily verified, since one can define q-Horn formulas where all of these width parameters are arbitrarily large. Conversely, by adding to a formula variable-disjoint copies of itself, we can make the deletion distance from q-Horn arbitrarily large, the width however does not increase.

There are several interesting research questions that arise from our paper. First, it would be interesting whether our algorithm can be strengthened to an exact FPT-algorithm for the detection of deletion q-Horn-backdoor sets. It would also be interesting, whether the W[2]-hardness of the detection of strong q-Horn-backdoor sets (Theorem 17) also holds if the input formula is in 3CNF. Finally, our hardness results contribute additional attention and significance to the problem of whether the parameterized Hitting Set problem has an FPT-approximation algorithm [18].

Acknowledgments

The authors acknowledge support from the OeAD (Austrian Indian collaboration grant, IN13/2011). Serge Gaspers, Sebastian Ordyniak, and Stefan Szeider acknowledge support from the European Research Council (COMPLEX REASON, 239962) and Serge Gaspers acknowledges support from the Australian Research Council (DE120101761).

References

1 B. Aspvall, M. F. Plass, and R. E. Tarjan. A linear-time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.

2 A. Biere. Bounded model checking. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, pages 457–481. IOS Press, 2009.

3 A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability.* IOS Press, 2009.

4 P. Bjesse, T. Leonard, and A. Mokkedem. Finding bugs in an alpha microprocessor using satisfiability solvers. In *Proceedings CAV 2001*, pages 454–464, 2001.

5 E. Boros, Y. Crama, and P. L. Hammer. Polynomial-time inference of all valid implications for horn and related formulae. *Ann. Math. Artif. Intell.*, 1:21–32, 1990.

6 E. Boros, P. L. Hammer, and X. Sun. Recognition of q-Horn formulae in linear time. *Discr. Appl. Math.*, 55(1):1–13, 1994.

7 S. A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd Annual Symp. on Theory of Computing*, pages 151–158, Shaker Heights, Ohio, 1971.

8 Y. Crama, O. Ekin, and P. L. Hammer. Variable and term removal from Boolean formulae. *Discr. Appl. Math.*, 75(3):217–230, 1997.

9 R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Monographs in Computer Science. Springer Verlag, New York, 1999.

10 R. G. Downey, M. R. Fellows, and C. McCartin. Parameterized approximation problems. In *Proceedings IWPEC 2006*, volume 4169 of *LNCS*, pages 121–129. Springer Verlag, 2006.

11 J. Flum and M. Grohe. *Parameterized Complexity Theory*, Springer Verlag, Berlin, 2006.

12 L. R. Ford, Jr. and D. R. Fulkerson. Maximal flow through a network. *Canadian J. Math.*, 8:399–404, 1956.

13 S. Gaspers and S. Szeider. Backdoors to satisfaction. In H. L. Bodlaender, R. Downey, F. V. Fomin, and D. Marx, editors, *The Multivariate Algorithmic Revolution and Beyond*, volume 7370 of *LNCS*, pages 287–317. Springer Verlag, 2012.

14 C. P. Gomes, H. Kautz, A. Sabharwal, and B. Selman. Satisfiability solvers. In *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 89–134. Elsevier, 2008.

15 H. A. Kautz and B. Selman. Planning as satisfiability. In *Proceedings ECAI 1992*, pages 359–363, 1992.

16 H. R. Lewis. Renaming a set of clauses as a Horn set. *J. of the ACM*, 25(1):134–135, Jan. 1978.

17 A. G. M. Prasad, A. Biere. A survey of recent advances in SAT-based formal verification. *Software Tools for Technology Transfer*, 7(2):156–173, 2005.

18 D. Marx. Can you beat treewidth? *Theory of Computing*, 6:85–112, 2010.

19 N. Nishimura, P. Ragde, and S. Szeider. Detecting backdoor sets with respect to Horn and binary clauses. In *Proceedings of SAT 2004*, pages 96–103, 2004.
20 I. Razgon and B. O'Sullivan. Almost 2-SAT is fixed parameter tractable. *J. of Computer and System Sciences*, 75(8):435–450, 2009.
21 M. Samer and S. Szeider. Algorithms for propositional model counting. *J. Discrete Algorithms*, 8(1):50–64, 2010.
22 T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of STOC 1978)*, pages 216–226. ACM, 1978.
23 M. N. Velev and R. E. Bryant. Effective use of Boolean satisfiability procedures in the formal verification of superscalar and VLIW microprocessors. *J. Symbolic Comput.*, 35(2):73–106, 2003.
24 R. Williams, C. Gomes, and B. Selman. Backdoors to typical case complexity. In *Proceedings of IJCAI 2003*, pages 1173–1178. Morgan Kaufmann, 2003.

On Polynomial Kernels for Sparse Integer Linear Programs

Stefan Kratsch[*]

Technical University Berlin, Germany
stefan.kratsch@tu-berlin.de

Abstract

Integer linear programs (ILPs) are a widely applied framework for dealing with combinatorial problems that arise in practice. It is known, e.g., by the success of CPLEX, that preprocessing and simplification can greatly speed up the process of optimizing an ILP. The present work seeks to further the theoretical understanding of preprocessing for ILPs by initiating a rigorous study within the framework of parameterized complexity and kernelization.

A famous result of Lenstra (Mathematics of Operations Research, 1983) shows that feasibility of any ILP with n variables and m constraints can be decided in time $\mathcal{O}(c^{n^3} \cdot m^{c'})$. Thus, by a folklore argument, any such ILP admits a kernelization to an equivalent instance of size $\mathcal{O}(c^{n^3})$. It is known, that unless NP $\subseteq$ coNP/poly and the polynomial hierarchy collapses, no kernelization with size bound polynomial in n is possible. However, this lower bound only applies for the case when constraints may include an arbitrary number of variables since it follows from lower bounds for SAT and Hitting Set, whose bounded arity variants admit polynomial kernelizations.

We consider the feasibility problem for ILPs $Ax \leq b$ where A is an r-row-sparse matrix parameterized by the number of variables. We show that the kernelizability of this problem depends strongly on the range of the variables. If the range is unbounded then this problem does not admit a polynomial kernelization unless NP $\subseteq$ coNP/poly. If, on the other hand, the range of each variable is polynomially bounded in n then we do get a polynomial kernelization. Additionally, this holds also for the more general case when the maximum range d is an additional parameter, i.e., the size obtained is polynomial in $n + d$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases integer linear programs, kernelization, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.80

1 Introduction

The present work seeks to initiate a study of the preprocessing properties of integer linear programs (ILPs) within the framework of parameterized complexity. Generally, preprocessing (or data reduction) is a universal strategy for coping with combinatorially hard problems and can be combined with other strategies like approximation, brute-force, exact exponential-time algorithms, local search, or heuristics. Unlike those other approaches, preprocessing itself incurs only a polynomial-time cost and is error free (or, in rare cases, with negligible error); recall that under standard assumptions we do not expect to exactly solve any NP-hard problem in polynomial time. Thus, preprocessing before applying other paradigms is essentially free and saves solution quality and/or runtime on parts of the input that are sufficiently easy to

[*] Work done in part while postdoc at Utrecht University supported by NWO project “KERNELS” and in part at Max-Planck-Institute for Informatics, Saarbrücken, Germany.

handle in polynomial time (see e.g. [24]). For a long time, preprocessing has been neglected in theoretical research for lack of appropriate tools[1] and research was limited to experimental evaluation of preprocessing strategies. The introduction of parameterized complexity and its notion of kernelization has sparked a strong interest in theoretically studying preprocessing with proven upper and lower bounds on its performance.

Integer linear programs are widely applied in theory and practice. There is a huge body of scientific literature on ILPs both as a topic of research itself and as a tool for solving other problems. From a theoretical perspective, many fundamental problems that revolve around ILPs are hard, e.g., checking feasibility of a 0/1-ILP is NP-hard by an easy reduction from the classic Satisfiability problem [15]. Similarly, it is easy to express Vertex Cover or Independent Set, thus showing that simple covering and packing ILPs are NP-hard to optimize. Thus, for worst-case complexity considerations, the high expressive power of ILPs comes at the price of encompassing plenty of hard problems and, effectively, inheriting all their lower bounds (e.g., approximability).

In practice, the expressive power of ILPs makes them a versatile framework for encoding and solving many combinatorially hard problems. Coupled with powerful software packages for optimizing ILPs this has created a viable way for solving many practical problems on real-world instances. We refer to a survey of Atamtürk and Savelsbergh [1] for an explanation of the capabilities of modern ILP solvers; this includes techniques such as probing and coefficient reduction. One of the most well-known solvers is the CPLEX package, which is, in particular, known for its extensive preprocessing options and parameters.[2] It is known that appropriate preprocessing and simplification of ILPs can lead to strong improvements in running time, e.g., reducing the range of variables or eliminating them altogether, or reducing the number of constraints. Given the large number of options that a user has for controlling the preprocessing in CPLEX, e.g., the number of substitution rounds to reduce rows and columns, this involves some amount of engineering and has a more heuristic flavor. In particular, there are no performance guarantees for the effect of the preprocessing.

Naturally, this leads to the question of whether there are theoretical performance guarantees for the viability of preprocessing for ILPs. To pursue this question in a rigorous and formal way, we take the perspective of parameterized complexity and its notion of (polynomial) kernelization. Parameterized complexity studies classical problems in a more fine-grained way by introducing one or more additional parameters and analyzing time- and space-usage as functions of input size and parameter. In particular, by formalizing a notion of *fixed-parameter tractability*, which requires efficient algorithms when the parameter is small, this makes the parameter a quantitative indicator of the hardness of a given instance (see Section 2 for formal definitions). This in turn permits us to formalize preprocessing as a reduction to an equivalent instance of size bounded in the parameter, a so-called *kernelization*. The intuition is that relatively easy instances should be reducible to a computationally hard, but small core, and we do not expect to reduce instances that are already fairly hard compared to their size (e.g., instances that are already reduced). While classically, no efficient algorithm can shrink *each* instance of an NP-hard problem [17], the notion of kernelization has been successfully applied to a multitude of problems (see recent surveys by Guo and Niedermeier [16] and Bodlaender [4]). Due to many interesting upper bound

[1] In fact, it has been observed that no polynomial-time algorithm can shrink all instances of some NP-hard problem unless P = NP [17]; this issue can be avoided in parameterized complexity.

[2] The interested reader is referred to the online documentation and manual of ILOG CPLEX 12.4 at `http://pic.dhe.ibm.com/infocenter/cosinfoc/v12r4/index.jsp` (see "presolve", "preprocessing").

results (e.g., [6, 13, 21]) but also the fairly recent development of a lower bound framework for polynomial kernels [17, 14, 5, 9], the existence or non-existence of polynomial kernels (which reduce to size polynomial in the parameter) is receiving high interest.

In this work, we focus on the effect that the dimension, i.e., the number of variables, has on the preprocessing properties of ILPs. Feasibility and optimization of ILPs with low dimension has been studied extensively already, see e.g. [19, 18, 22, 23, 20, 8, 11, 12]. The most important result for our purpose is a well-known work of Lenstra [22], who showed that feasibility of an ILP with n variables and m constraints can be decided in time $\mathcal{O}(c^{n^3} \cdot m^{\mathcal{O}(1)})$; this also means that the problem is fixed-parameter tractable with respect to n. This has been improved further, amongst others by Kannan [20] to $\mathcal{O}(n^{\mathcal{O}(n)})$ dependence on the dimension and by Clarkson [8] to (expected) $\mathcal{O}((cn)^{n/2+\mathcal{O}(1)})$ dependence. We take these results as our starting point and consider the problem of determining feasibility of a given ILP parameterized by the number of variables, formally defined as follows.

Integer Linear Program Feasibility(n)– ILPF(n)
Input: A matrix $A \in \mathbb{Q}^{m\times n}$ and a vector $b \in \mathbb{Q}^m$.
Parameter: n.
Output: Is there a vector $x \in \mathbb{Z}^n$ such that $Ax \leq b$?

It is known by a simple folklore argument that any parameterized problem is fixed-parameter tractable if and only if it admits a kernelization; unfortunately the implied size guarantee is usually impractical as it is exponential in the parameter. As an example, using the runtime given by Kannan [20] we only get a kernel size of $\mathcal{O}(n^{cn})$.[3] Unsurprisingly, we are more interested in what kernel sizes can be achieved by nontrivial preprocessing rules. In particular, we are interested in the conditions under which an ILP with n variables can be reduced to size polynomial in n, i.e., in the existence of polynomial kernels for Integer Linear Program Feasibility(n).

Related work. Regarding the existence of polynomial kernels for Integer Linear Program Feasibility(n) only little is known. In general, parameterized by the number of variables, ILPF(n) admits no polynomial kernelization unless NP $\subseteq$ coNP/poly and the polynomial hierarchy collapses. This follows for example from the results of Dell and van Melkebeek [9] regarding lower bounds for the compressibility of the satisfiability problem, since there is an immediate reduction from SAT to ILPF(n). Similarly, it follows also from earlier results of Dom et al. [10] who showed that Hitting Set parameterized by the universe size admits no polynomial kernelization under the same assumption.

We note that both ways of excluding polynomial kernels for Integer Linear Program Feasibility(n) use reductions from problems with unbounded arity. Crucially, both d-Hitting Set and d-SAT admit polynomial kernels of size roughly $\mathcal{O}(n^d)$, where n is the number of elements and variables respectively, which can be obtained trivially by discarding duplicate sets or clauses, respectively. Surprisingly perhaps, the work of Dell and van Melkebeek [9] shows that these bounds are tight, assuming NP $\nsubseteq$ coNP/poly, i.e., there are no reductions to size $\mathcal{O}(n^{d-\epsilon})$ for any $\epsilon > 0$. We emphasize that this also implies the lower bound of Integer Linear Program Feasibility(n) since it can express, e.g., Hitting Set with sets of unbounded size (exceeding any constant d).

Motivated by these facts about the kernelization lower bound for Integer Linear Program Feasibility(n) and the existing straightforward polynomial kernels for d-Hitting

[3] If the instance is larger than $\mathcal{O}(n^{cn})$, then Kannan's algorithm runs in polynomial time and we may simply return the answer or a trivial **yes**- or **no**-instance. Otherwise, the claimed bound trivially holds.

Set and d-SAT, we study the influence of arity on the existence of polynomial kernels for ILPF(n). Regarding the considered integer linear programs with constraints $Ax \leq b$ this translates to A being r-row-sparse, i.e., to have at most r nonzero variables in each row.

Our results. We study Integer Linear Program Feasibility(n) for the case that the constraint matrix A is r-row-sparse; we call this problem r-Sparse Integer Linear Program Feasibility(n) (r-SILPF(n)). Note that r is a constant that is fixed as a part of the problem (it makes no sense to study r as an additional parameter since we already know that constraints involve at most all n variables, but already for SAT parameterized by the number of variables this is not enough to avoid a kernelization lower bound).

Our main result is that r-SILPF(n) admits no polynomial kernelization assuming that NP $\nsubseteq$ coNP/poly, for any $r \geq 3$. Thus we see that unlike the simpler problems d-Hitting Set and d-SAT, a restriction on the arity (or row-sparseness) is not enough to ensure a polynomial kernelization. For this result we give a cross-composition (introduced by Bodlaender et al. [7]; see Section 2) from Clique to r-SILPF(n). Concretely, we encode t instances of Clique into a single instance of r-SILPF(n) with parameter value bounded polynomially in the largest Clique instance plus $\log t$, such that our obtained instance is **yes** if and only if at least one of the Clique instances is **yes**.

Unlike other proofs via compositions or cross-compositions, the parameterization by the number of variables combined with the row-sparseness restriction prevent many standard tricks. For example, without the row-sparseness we could simply encode the selection of an instance number of one of the t Clique instances. Then we could add constraints that encode all the edges of the input graphs, but which are only valid when the binary encoding of the instance number matches the constraint. Unfortunately, this involves constraints with $\mathcal{O}(\log t)$ variables.[4] (Of course without row-sparseness, a lower bound is known already.) Similarly, if we could use t slack variables we could very easily control the constraints and have only those for a single instance of Clique be relevant; however, we cannot afford this.

Our solution goes by using a significantly larger domain for the variables that encode the selection of a clique in one of the t input graphs. We use a variable s for the instance number, and add (linear) constraints that enforce $r = s^2$. This permits us to use indicator variables for the desired clique whose feasible values depend quadratically on the chosen instance number. Accordingly, we can arrange the constraints for the edges of all input graphs G_i, such that they intersect this feasible region when $i = s$. In this way, depending on s, only the constraints from one instance will restrict the choice of values for the indicator variables (beyond the restriction imposed directly by s and $r = s^2$). This is presented in Section 3.

Complementing our lower bound, and recalling the large domain required for the construction, we analyze the effect of the maximum variable range on the preprocessing. It turns out that we can efficiently reduce row-sparse ILPs of form $Ax \leq b$ to a size that is polynomial in $n + d$, where n is the number of variables and d is the maximum range of any variable. In other words, r-Sparse Integer Linear Program Feasibility admits a polynomial kernelization with respect to the combined parameter $n + d$, or when d is polynomially bounded in n; this is showed in Section 4. Together our upper and lower bound show that the existence for r-Sparse Integer Linear Program Feasibility depends strongly on the permitted range for the variables. We emphasize that small range without row-sparseness does not suffice by the mentioned reductions from SAT and Hitting Set.

Furthermore, let us point out that for the case of an ILP of form $Ax = b$, $x \geq 0$, r-

[4] We can emulate a few such constraints by use of auxiliary variables, but we cannot afford to do this for the constraints corresponding to all t instances.

row-sparseness does suffice to reduce the number of constraints. (Note that this problem is polynomially solvable for $r \leq 2$ and NP-hard for $r \geq 3$, cf. [3].) This follows easily from standard tools for solving systems of linear equations, namely Gaussian elimination. We briefly explain this in Section 5. Note that, while in general there are trivial transformations between $Ax \leq b$ and $A'x' = b'$, going from $Ax \leq b$ to $A'x' = b'$ uses one slack variable per constraint and hence would increase our parameter (the number of variables) by the number of constraints; this would make any further reduction arguments pointless.

2 Preliminaries

Parameterized complexity and kernelization. A *parameterized problem* over some finite alphabet Σ is a language $\mathcal{P} \subseteq \Sigma^* \times \mathbb{N}$. The problem $\mathcal{P}$ is *fixed-parameter tractable* if $(x,k) \in \mathcal{P}$ can be decided in time $f(k) \cdot (|x| + k)^{\mathcal{O}(1)}$, where f is an arbitrary computable function. A polynomial-time algorithm K is a kernelization for $\mathcal{P}$ if, given input (x,k), it computes an equivalent instance (x',k') with $|x'| + k' \leq h(k)$ where h is some computable function; K is a *polynomial* kernelization if h is polynomially bounded (in k). By relaxing the restriction that the created instance (x',k') must be of the same problem and allow the output to be an instance of any classical decision problem we get the notion of *(polynomial) compression*. Almost all lower bounds for kernelization apply also for this weaker notion.

For our lower bound proof we use the concept of an (OR-)cross-composition of Bodlaender et al. [7] which builds on a series of earlier results [14, 5, 9] that created a framework for ruling out polynomial kernelizations for certain problems. Cross-composition streamlines and extends the earlier notion of a composition by allowing arbitrary source problems and simplifying padding arguments via the concept of a polynomial equivalence relation.

▶ **Definition 1** ([7]). An equivalence relation $\mathcal{R}$ on Σ^* is called a *polynomial equivalence relation* if the following two conditions hold:

1. There is a polynomial-time algorithm that decides whether two strings belong to the same equivalence class (time polynomial in $|x| + |y|$ for $x, y \in \Sigma^*$).
2. For any finite set $S \subseteq \Sigma^*$ the equivalence relation $\mathcal{R}$ partitions the elements of S into a number of classes that is polynomially bounded in the size of the largest element of S.

▶ **Definition 2** ([7]). Let $L \subseteq \Sigma^*$ be a language, let $\mathcal{R}$ be a polynomial equivalence relation on Σ^*, and let $\mathcal{P} \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. An OR-*cross-composition of L into $\mathcal{P}$* (with respect to $\mathcal{R}$) is an algorithm that, given t instances $x_1, x_2, \ldots, x_t \in \Sigma^*$ of L belonging to the same equivalence class of $\mathcal{R}$, takes time polynomial in $\sum_{i=1}^{t} |x_i|$ and outputs an instance $(y,k) \in \Sigma^* \times \mathbb{N}$ such that:

1. The parameter value k is polynomially bounded in $\max_i |x_i| + \log t$.
2. The instance (y,k) is **yes** for $\mathcal{P}$ if and only if *at least one* instance x_i is **yes** for L.

We then say that L OR-cross-composes into $\mathcal{P}$.

▶ **Theorem 3** ([7]). *If an NP-hard language L* OR*-cross-composes into the parameterized problem $\mathcal{P}$, then $\mathcal{P}$ does not admit a polynomial kernelization or polynomial compression unless* NP $\subseteq$ coNP/poly *and the polynomial hierarchy collapses.*

3 A kernelization lower bound for sparse ILP Feasibility

In this section we show our main result, namely that a restriction to row-sparse matrices is not enough to ensure a polynomial kernelization for INTEGER LINEAR PROGRAM FEASIBILITY parameterized by the number of variables. The problem is defined as follows.

r-Sparse Integer Linear Programming Feasibility(n) – r-SILPF(n)
Input: An r-row-sparse matrix $A \in \mathbb{Q}^{m\times n}$ and a vector $b \in \mathbb{Q}^m$.
Parameter: n.
Output: Is there a vector $x \in \mathbb{Z}^n$ such that $Ax \leq b$?

To prove the kernelization lower bound for r-SILPF we give an or-cross-composition from the NP-hard Clique problem, i.e., a reduction of many Clique instances into a single instance of r-SILPF. The idea behind the construction is to use a fairly large domain in order to recycle the same variables for the constraints that correspond to many different instances.

As a first step we state two propositions which together allow us to "compute" the square of a variable inside an ILP, i.e., to add constraints such that some variable is exactly the square of another in all feasible solutions.

▶ **Proposition 1.** *Let s_i, s_j, s_{ij}, and d_{ij} denote integer variables with range $\{0,1\}$ each. Then any feasible assignment for $s_{ij} = \frac{1}{2}(s_i + s_j - d_{ij})$ satisfies $s_{ij} = s_i \cdot s_j$. Conversely, for any choice of s_i, s_j, and s_{ij} such that $s_{ij} = s_i \cdot s_j$, there is a choice of $d_{ij} \in \{0,1\}$ such that $s_{ij} = \frac{1}{2}(s_i + s_j - d_{ij})$ holds.*

▶ **Proposition 2.** *Let $s \in \{0,\dots,t-1\}$ with $t = 2^\ell$ and let $s_0,\dots,s_{\ell-1} \in \{0,1\}$ denote the binary expansion of s, i.e., $s = \sum_{i=0}^{\ell-1} 2^i s_i$. Then*

$$s^2 = \sum_{i=0}^{\ell-1}\left(\sum_{j=0}^{\ell-1} 2^{i+j} s_i \cdot s_j\right).$$

Together the two propositions provide a way of forcing some variable in an ILP to take a value exactly equal to the square of another value. If $s \in \{0,\dots,t-1\}$ this requires $\mathcal{O}(\log^2 t)$ auxiliary variables and $\mathcal{O}(\log^2 t)$ constraints. Now we will give our construction.

▶ **Theorem 4.** *Let $r \geq 3$ be an integer. The r-SILPF problem does not admit a polynomial kernelization or compression unless NP $\subseteq$ coNP/poly and the polynomial hierarchy collapses.*

Proof. We give an or-cross-composition from the NP-hard Clique problem. Let t instances of Clique be given. By a polynomial equivalence relation that partitions instances according to number of vertices and requested clique size it suffices to consider instances that ask for the same clique size k and such that each input graph has n vertices. We denote the instances $(G_0,k),\dots,(G_{t-1},k)$; for convenience, assume that all t graphs have the same vertex set V and edge sets E_i for $i \in \{0,\dots,t-1\}$. We will create a single instance of r-Sparse Integer Linear Program Feasibility(n) that is **yes** if and only if at least one instance (G_i,k) is **yes** for Clique. Without loss of generality, we assume that $t = 2^\ell$; otherwise we could copy some instance sufficiently often (at most doubling the input size).
Construction–essential part. For the sake of readability we first describe the matrix A by writing down the constraints in a succinct way ignoring the sparsity requirement; there will be a small number of constraints on more than three variables which will be converted later. We also specify explicit ranges for the variables which can be enforced by the obvious constraints. Note that n, t, ℓ, k, i, and j are constants in the ILP; i and j are used in sums but the expansion of each sum is a constraint where i and j have constant values.

The first group of variables, namely s and $s_0,\dots,s_{\ell-1}$ serve to pick an instance number $s \in \{0,\dots,t-1\}$ and enforce the variables s_i to equal the binary expansion of s.

$$s \in \{0,\dots,t-1\} \tag{1}$$

$$s_0,\dots,s_{\ell-1} \in \{0,1\} \tag{2}$$

$$s = \sum_{i=0}^{\ell-1} 2^i s_i \tag{3}$$

Next we create a variable r and auxiliary variables s_{ij} and d_{ij} with the sole purpose of enforcing $r = s^2$ but using only linear constraints.

$$r \in \{0, \dots, (t-1)^2\} \tag{4}$$

$$s_{ij}, d_{ij} \in \{0, 1\} \qquad \text{for all } i, j \in \{0, \dots, \ell - 1\} \tag{5}$$

$$s_{ij} = \frac{1}{2}(s_i + s_j - d_{ij}) \qquad \text{for all } i, j \in \{0, \dots, \ell - 1\} \tag{6}$$

$$r = \sum_{i=0}^{\ell-1} \left(\sum_{j=0}^{\ell-1} 2^{i+j} s_{ij} \right) \tag{7}$$

We introduce variables y_v for all $v \in V$ which will encode a k-clique in instance s. These variables are restricted to take one of two values that depend on s in a quadratic way (using $r = s^2$; recall that t is a constant).

$$y_v \le 2ts - r + 2 \qquad \text{for all } v \in V \tag{8}$$

$$y_v \ge 2ts - r + 1 \qquad \text{for all } v \in V \tag{9}$$

That is, we restrict y_v to $y_v \in \{2ts - r + 1, 2ts - r + 2\} \subseteq \{0, \dots, 2t^2\}$.

Now we get to the central piece of the ILP, namely the constraints which will enforce the non-edges of the graph G_s. However, we of course need to add those constraints for all input graphs G_i. It is crucial that only the constraints for $i = s$ have an effect on the y-variables (beyond the restriction already imposed by (8) and (9)). We add the following for all $\{u, v\} \subseteq V$ and instance numbers $i \in \{0, \dots, t-1\}$ if $\{u, v\}$ is not an edge of G_i.

$$\text{if } \{u, v\} \notin E_i \text{ then} \qquad y_u + y_v \le 4 \cdot (t - i) \cdot s + 2i^2 + 3 \tag{10}$$

Finally, we take the sum over all y_v, deduct n times the minimum value $2ts - r + 1$ and check that this is at least as large as the specified target value k.

$$\left(\sum_{v \in V} y_v \right) - n \cdot (2ts - r + 1) \ge k \tag{11}$$

This completes the essential part of the construction. Formally we still need to convert all constraints into form $Ax \le b$ and to use only three variables in each constraint. However, the proof will be given regarding the more accessible constraints stated above.

Construction–formal part. We use x to refer to the vector of all variables used above, e.g., $x = (s, s_0, \dots, s_{\ell-1}, r, s_{00}, \dots, s_{\ell-1,\ell-1}, d_{00}, \dots, d_{\ell-1,\ell-1}, y_{v_1}, \dots, y_{v_n})$. Thus, at this point, we use $1 + \ell + 1 + 2 \cdot \ell^2 + n \in \mathcal{O}(n + \ell^2) = (n + \log t)^{\mathcal{O}(1)}$ variables.

To formally complete the construction one now needs to translate all constraints to form $Ax \le b$. Furthermore, using auxiliary variables, one needs to convert this to $A'x' \le b'$ such that A' has at most three non-zero entries in each row. It is clear that all range constraints, namely (1), (2), (4), and (5) can be expressed by two linear inequalities with one variable each. Also the constraints (8), (9), and (10) need no further treatment since they are already linear inequalities with at most three variables each (that is, it suffices to rearrange them to have all variables on one side when transforming to $Ax \le b$).

For the remaining constraints, namely (3), (6), (7), and (11) we need to use auxiliary variables to replace them by small sets of linear inequalities with at most three variables each. We sketch this for (3), which requires expressing a sum using partial sums. We introduce ℓ new variables $z_0, \dots, z_{\ell-1}$ and replace $s = \sum_{i=0}^{\ell-1} 2^i s_i$ as follows; the intuition is that $z_j = \sum_{i=0}^{j} 2^i s_i$.

$$\begin{aligned} z_0 - s_0 &\le 0 & -z_0 + s_0 &\le 0 & \\ z_i - z_{i-1} - 2^i s_i &\le 0 & -z_i + z_{i-1} + 2^i s_i &\le 0 & \text{for } i \in \{1, \dots, \ell-1\} \\ s - z_{\ell-1} &\le 0 & -s + z_{\ell-1} &\le 0 & \end{aligned}$$

We use ℓ variables for constraint (3), ℓ^2 variables for constraints (6), ℓ^2 variables for constraint (7), and $n+2$ variables for constraint (11). Altogether we use $\mathcal{O}(n+\ell^2) = \mathcal{O}(n+\log^2 t)$ additional variables. In total our ILP uses $\mathcal{O}(n+\log^2 t) = \mathcal{O}((n+\log t)^{\mathcal{O}(1)})$ variables, which is consistent with the definition of a cross-composition (polynomial in the largest input instance plus the logarithm of the number of instances).

Completeness. To show correctness, let us first assume that some instance (G_{i^*}, k) is **yes** for Clique, and let $C \subseteq V$ be some k-clique in G_{i^*}. We will determine a value $x' = x'(i^*, C)$ such that $A'x' \le b'$ (this is the system obtained by transforming all constraints to inequalities in at most three variables). Again, for clarity, we will simply pick values only for all variables used in the succinct representation (i.e., all variables occurring in (1)–(11)) and check that all (in-)equalities are satisfied. It is obvious how to extend this to the auxiliary variables that are required for formally writing down all constraints as $A'x' \le b'$.

First of all, we set $s = i^* \in \{0, \dots, t-1\}$ and set the variables $s_0, \dots, s_{\ell-1} \in \{0,1\}$ such that they match the binary expansion of s. Clearly, this satisfies constraint (3) as well as the range of each encountered variable. It follows from Proposition 1 that we can set $s_{ij} = s_i \cdot s_j \in \{0,1\}$ and also find feasible values for all d_{ij} such that all constraints (6) are satisfied. Hence, by Proposition 2 we can set $r = s^2$ while satisfying constraint (7).

Now, let us assign values to variables y_v for $v \in V$ as follows

$$y_v = \begin{cases} 2ts - r + 2 & \text{if, } v \in C \\ 2ts - r + 1 & \text{if } v \notin C. \end{cases}$$

It is easy to see that this choice satisfies both constraints (9) and (11), since $|C| = k$.

Finally, we have to check that the (non-)edge constraints (10) are satisfied for all $i \in \{0, \dots, t-1\}$ and all edges $\{u, v\}$. There are two cases, namely $i = i^*$ and $i \ne i^*$, i.e., we have to satisfy constraints for G_{i^*} (using the fact that C is a clique) but also constraints created for graphs G_i with $i \ne i^*$.

Let us first consider the case $i \ne i^*$; concretely, we take the maximum value for $y_u + y_v$, namely $2\cdot(2ts - r + 2)$, and compare it to the value of constraint (10), namely $4\cdot(t-i)\cdot s + 2i^2 + 3$, using that $r = s^2$ and $s = i^*$:

$$\begin{aligned} & 4\cdot(t-i)\cdot s + 2i^2 + 3 \ge 2\cdot(2ts - r + 2) \\ \Leftrightarrow\quad & 4ts - 4is + 2i^2 + 3 \ge 4ts - 2s^2 + 4 \\ \Leftrightarrow\quad & 2s^2 - 4is + 2i^2 - 1 \ge 0 \\ \Leftrightarrow\quad & 2(s-i)^2 - 1 \ge 0. \end{aligned}$$

Since $s = i^*$ the last inequality holds if $i \ne i^*$, which is exactly what we assumed. Thus all non-edge constraints for graphs G_i with $i \ne i^*$ are satisfied.

We now consider the non-edge constraints for G_{i^*}. We compute the difference between the bound of constraint (10) and the minimum value of $y_u + y_v$, namely $2\cdot(2ts - r + 1)$, to check that our assignment to y-variables is feasible. Note that $r = s^2$ and $s = i^* = i$:

$$(4\cdot(t-i)\cdot s + 2i^2 + 3) - 2\cdot(2ts - r + 1) = 4ts - 4is + 2i^2 + 3 - 4ts + 2s^2 - 2 = 1.$$

Thus, if $\{u,v\} \notin E_{i^*}$ then at most one of y_u and y_v can take value $2ts-r+2$ without violating constraint (10). Otherwise, if $\{u,v\} \in E_{i^*}$, then, from the perspective of this edge, both variables may take value $2ts-r+2$. Clearly, this is consistent with our assignment to the y-variables, since the larger value $2ts-r+2$ is assigned to all variables that correspond to the vertices of the k-clique C.

Soundness. For soundness, let us assume that we have a feasible solution x' such that $A'x' \leq b'$. Again, we consider only the variables of constraints (1)–(11). Recall that $s \in \{0,\dots,t-1\}$. We claim that the graph G_s must have a clique of size at least k.

Observe that all variables y_v for $v \in V$ have value $2ts-r+2$ or $2ts-r+1$ in x due to constraints (8) and (9). We define a vertex subset $C \subseteq V$ by stating that it contains exactly those vertices v with $y_v = 2ts-r+2$. The goal is to show that C is a clique in G_s.

As for the converse direction, feasible solutions are required to have $r = s^2$, which follows from Propositions 1 and 2; note that obviously the variables $s_0,\dots,s_{\ell-1}$ need to equal the binary expansion of s due to constraint (3).

Now, we consider the non-edge constraints (10) for G_s and compare them to the lower bound of $2ts-r+1$ for variables y_v; we already did this computation earlier, again we have $r = s^2$ and $s = i$:

$$4\cdot(t-i)\cdot s + 2i^2 + 3 - 2\cdot(2ts-r+1) = 1.$$

Hence, for every non-edge $\{u,v\}$ of G_s among y_u and y_v at most one of the two variables can take the larger value $2ts-r+2$. Therefore, when $y_u = y_v = 2ts-r+2$, then $\{u,v\}$ is an edge of G_s. Thus, C is a clique in G_s. It follows from $y_v \in \{2ts-r+1, 2ts-r+2\}$ that constraint (11) enforces that $y_v = 2ts-r+2$ for at least k vertices $v \in V$. Therefore, C is of size k. This completes the OR-cross-composition from CLIQUE.

By Theorem 3, r-SPARSE INTEGER LINEAR PROGRAM FEASIBILITY(n) has no polynomial kernelization unless NP $\subseteq$ coNP/poly and the polynomial hierarchy collapses [7]. ◀

4 A polynomial kernelization for sparse ILP with bounded range

We have seen that for r-SPARSE INTEGER LINEAR PROGRAM FEASIBILITY(n) there is no polynomial kernelization unless NP $\subseteq$ coNP/poly. The proof relies strongly on having variables of high range in order to encode the constraints of t instances of CLIQUE. It is natural to ask, whether a similar result can be proven when the maximum range of any variable is small, e.g., polynomial in the number of variables. We show that this is not the case by presenting a polynomial kernelization for the variant where the maximum range is an additional parameter. The problem is defined as follows.

r-SPARSE BOUNDED INTEGER LINEAR PROGRAM FEASIBILITY(n,d)
Input: An r-row-sparse matrix $A \in \mathbb{Q}^{m\times n}$ and a vector $b \in \mathbb{Q}^m$.
Parameter: $n+d$.
Output: Is there a vector $x \in \{0,\dots,d-1\}^n$ such that $Ax \leq b$?

Note that we restrict to the seemingly special case where each variable is not only restricted to d different consecutive values, but in fact all variables must take values from $\{0,\dots,d-1\}$. It can be easily checked that this is as general as allowing any d consecutive integers, since we could shift variables to range $\{0,\dots,d-1\}$ without changing feasibility (by changing b).

▶ **Theorem 5.** r-SPARSE BOUNDED INTEGER LINEAR PROGRAMMING FEASIBILITY*(n,d) admits a polynomial kernelization with size $\mathcal{O}(n^r \cdot d^r \cdot \log nd)$.*

Proof. We assume that $r \geq 3$ since otherwise the problem can be solved in time $\mathcal{O}(m \cdot d)$ by work of Bar-Yehuda and Rawitz [2] and the theorem follows trivially. Recall that for $r \geq 3$ the problem is NP-hard by a reduction from 3-SAT.

The kernelization works by considering all choices of r of the n variables and replacing the constraints (i.e., inequalities) in $Ax \leq b$ which contain only those variables. The starting observation is that there are d^r choices of picking values for r variables, and the considered constraints prevent some of those from being feasible. It can be efficiently checked which of the d^r assignments are feasible. For each infeasible point $P = (p_1, \ldots, p_r)$ we show how to give a small number of constraints that exactly exclude this point. Together, all those new constraints have the same effect as the original ones, allowing the latter to be discarded.

Let $x_1, \ldots, x_r$ be any r of n variables and let $\hat{P}$ denote the set of all points $P = (p_1, \ldots, p_r)$ that are infeasible for constraints only involving $x_1, \ldots, x_r$. (Note that the whole ILP might be infeasible, but locally we only care for an equivalent replacement of the constraints.) We show constraints that enforce $(x_1, \ldots, x_r) \neq (p_1, \ldots, p_r)$:

$$\forall i \in \{1, \ldots, r\} : \qquad x_i = p_i + s_i - d \cdot t_i \qquad s_i \in \{0, \ldots, d-1\}, t_i \in \{0, 1\} \tag{12}$$

$$\sum_{i=1}^{r} s_i \geq 1 \tag{13}$$

This requires $2r$ variables and $r + 1$ constraints; a few more variables and constraints are required to transform the constraints into an equivalent set of inequalities with at most r variables each: For constraint (13) it suffices to flip the sign since it is already an inequality on r variables. For constraints (12) we can replace each equality by two equalities using a new auxiliary variable (in fact this is only needed when $r = 3$) and replacing both equalities in turn by two inequalities. We use $3r$ variables and $4r + 1$ constraints total. Note that all coefficients have values in $\{-1, 0, 1, d\}$ and can be encoded by $\mathcal{O}(\log d)$ bits (in fact two bits suffice easily for four values).

Again, we will argue correctness on the more succinct representation, i.e., on (12) and (13).

Assume first that $(x_1, \ldots, x_r) = (p_1, \ldots, p_r)$. Thus $0 = x_i - p_i = s_i - d \cdot t_i$, which implies that $s_i = t_i = 0$ (taking into account the domains of s_i and t_i) for all i. Thus constraint (13) is violated, making $(x_1, \ldots, x_r) = (p_1, \ldots, p_r)$ infeasible. On the other hand, if $(x_1, \ldots, x_r) \neq (p_1, \ldots, p_r)$, then there is a position j with $x_j \neq p_j$. It follows that $0 < |x_j - p_j| < d$ (due to the range of x_j) which in turn implies that $s_j \neq 0$ since the contribution of $d \cdot t_j$ to the equality is a multiple of d. Thus constraint (13) is fulfilled.

It follows that we are able to add constraints which exclude any desired point for $x_1, \ldots, x_r$. Let us complete the proof. Clearly, if a vector x fulfills $Ax \leq b$ then any choice of r variables from x fulfills all constraints that contain only these variables. This in turn means that those variables avoid the points that are excluded by the constraints, which implies that they satisfy all our new constraints (since avoiding those points is all that is needed).

Conversely, assume that a vector x fulfills all new constraints and hence any choice of r variables avoids all forbidden points. Since any of the original constraints contains at most r variables, it comes down to forbidding some set of points. Since x fulfills our new constraints it also avoids all infeasible points for $Ax \leq b$. Thus, x satisfies also all original constraints.

Summarizing, we are able to replace all constraints by new constraints with small coefficients, which have the same outcome. Clearly the computations can be performed in polynomial time (the input size dominates n, m, and the encodings of all coefficients in A and b). Since for any r variables there are at most d^r infeasible points, we need at most $(4r + 1) \cdot d^r \cdot \binom{n}{r} = \mathcal{O}(d^r \cdot n^r)$ constraints and $3r \cdot d^r \cdot n^r = \mathcal{O}(d^r \cdot n^r)$ variables. The generated equivalent instance can be encoded by $r \cdot \mathcal{O}(\log(d^r \cdot n^r)) \cdot \mathcal{O}(d^r \cdot n^r) = \mathcal{O}(d^r \cdot n^r \cdot \log dn)$

bits, by encoding each constraint (on r variables) as the binary encoded names of the variables with nonzero coefficients followed by the values of the coefficients. ◀

5 Preprocessing for sparse Equality ILP

Now we will briefly describe how to reduce the number of constraints for the feasibility problem of r-row-sparse ILPs of form $Ax = b$, $x \geq 0$. Note that we make no explicit use of the non-negativity of x but the problem is polynomially solvable without it.

r-Sparse Equality Integer Linear Programming Feasibility(n) – r-SEILPF(n)
Input: An r-row-sparse matrix $A \in \mathbb{Q}^{m\times n}$ and a vector $b \in \mathbb{Q}^m$.
Parameter: n.
Output: Is there a vector $x \in \mathbb{Z}^n$ such that $Ax = b$ and $x \geq 0$?

▶ **Theorem 6.** *There is a polynomial-time algorithm that reduces any instance of r-Sparse Equality Integer Linear Program Feasibility(n) to an equivalent instance with at most $\mathcal{O}(n^r)$ constraints.*

Proof. Given an instance of r-SEILPF, apply the following procedure for each of the $\binom{n}{r}$ choices of r of the n variables. For a given set of variables, say $x_1, \ldots, x_r$, list all constraints in $Ax = b$ that contain only $x_1, \ldots, x_r$ as variables. If there are more than r such constraints, then apply Gaussian elimination to either find a redundant constraint, or to find out that there is no feasible assignment for this set of variables. In the latter case, clearly, the whole instance is infeasible and we return **no** (or a dummy **no**-instance). Repeat while there are more than r constraints. At the end, if we never terminate with **no**, we have reduced the overall number of constraints down to at most $r \cdot \binom{n}{r} = \mathcal{O}(n^r)$ constraints. ◀

We point out that the reduction of Theorem 6 does not imply a polynomial kernelization. The reason is that while we reduce to an equivalent ILP $A'x = b'$ where A' is an $\mathcal{O}(n^r) \times n$ matrix, the coefficients in A' and b' may still have arbitrary values and coding length.

6 Conclusion

We prove that the existence of polynomial kernels for r-Sparse Integer Linear Program Feasibility with respect to the number n of variables depends strongly on the maximum range of the variables. If the range is unbounded, then there is no polynomial kernelization under standard assumptions. Otherwise, if the range of each variable is polynomially bounded in n then we establish a polynomial kernelization. This holds also for the more general case of using the maximum range as an additional parameter.

Future work will be directed at more restricted cases of ILPs in order to obtain more positive kernelization results. Similarly, structural parameters of ILPs seem largely unexplored.

References

1 Alper Atamtürk and Martin W. P. Savelsbergh. Integer-programming software systems. *Annals OR*, 140(1):67–124, 2005.

2 Reuven Bar-Yehuda and Dror Rawitz. Efficient algorithms for integer programs with two variables per constraint. *Algorithmica*, 29(4):595–609, 2001.

3 Manuel Bodirsky, Gustav Nordh, and Timo von Oertzen. Integer programming with 2-variable equations and 1-variable inequalities. *Inf. Process. Lett.*, 109(11):572–575, 2009.

4 Hans L. Bodlaender. Kernelization: New upper and lower bound techniques. In *IWPEC*, volume 5917 of *LNCS*, pages 17–37. Springer, 2009.

5 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.

6 Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) Kernelization. In *FOCS*, pages 629–638, 2009.

7 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Cross-composition: A new technique for kernelization lower bounds. In *STACS*, volume 9 of *LIPIcs*, pages 165–176. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

8 Kenneth L. Clarkson. Las vegas algorithms for linear and integer programming when the dimension is small. *J. ACM*, 42(2):488–499, 1995.

9 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In *STOC*, pages 251–260. ACM, 2010.

10 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Incompressibility through colors and ids. In *ICALP (1)*, volume 5555 of *LNCS*, pages 378–389. Springer, 2009.

11 Friedrich Eisenbrand. Fast integer programming in fixed dimension. In *ESA*, volume 2832 of *LNCS*, pages 196–207. Springer, 2003.

12 Friedrich Eisenbrand and Gennady Shmonin. Parametric integer programming in fixed dimension. *Math. Oper. Res.*, 33(4):839–850, 2008.

13 Fedor V. Fomin, Daniel Lokshtanov, Saket Saurabh, and Dimitrios M. Thilikos. Bidimensionality and kernels. In *SODA*, pages 503–510. SIAM, 2010.

14 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011.

15 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

16 Jiong Guo and Rolf Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, 2007.

17 Danny Harnik and Moni Naor. On the compressibility of $\mathcal{NP}$ instances and cryptographic applications. *SIAM J. Comput.*, 39(5):1667–1713, 2010.

18 Ravi Kannan. Improved algorithms for integer programming and related lattice problems. In *STOC*, pages 193–206. ACM, 1983.

19 Ravindran Kannan. A polynomial algorithm for the two-variable integer programming problem. *J. ACM*, 27(1):118–122, 1980.

20 Ravindran Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, 1987.

21 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *FOCS*, pages 450–459. IEEE Computer Society, 2012.

22 Hendrik W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983.

23 Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *J. ACM*, 31(1):114–127, 1984.

24 Karsten Weihe. Covering trains by stations or the power of data reduction. In *Proceedings of* ALENEX, pages 1–8, 1998.

Linear kernels for (connected) dominating set on graphs with excluded topological subgraphs

Fedor V. Fomin[*1], **Daniel Lokshtanov**[1], **Saket Saurabh**[2], and **Dimitrios M. Thilikos**[†3]

1 **Department of Informatics, University of Bergen, Norway**
`{fomin|daniello}@ii.uib.no`
2 **The Institute of Mathematical Sciences,**
Chennai 600113, India.
`saket@imsc.res.in`
3 **National and Kapodistrian University of Athens, Greece.**
`sedthilk@thilikos.info`

Abstract

We give the first linear kernels for Dominating Set and Connected Dominating Set problems on graphs excluding a fixed graph H as a topological minor.

1998 ACM Subject Classification G.2.2, F.2.2

Keywords and phrases Parameterized complexity, kernelization, algorithmic graph minors, dominating set, connected dominating set

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.92

1 Introduction

Kernelization is an emerging technique in parameterized complexity. A parameterized problem is said to admit a *polynomial kernel* if there is a polynomial-time algorithm (the degree of the polynomial is independent of the parameter k), called a *kernelization* algorithm, that reduces the input instance down to an instance with size bounded by a polynomial $p(k)$ in k, while preserving the answer. This reduced instance is called a $p(k)$ *kernel* for the problem. If the size of the kernel is $O(k)$, then we call it a *linear kernel*.

The Dominating Set (DS) problem together with its numerous variants is one of the most classic and well-studied problems in algorithms and combinatorics [27]. In the Dominating Set problem, we are given a graph G and a non-negative integer k, and the question is whether G contains a set of k vertices whose closed neighborhood contains all the vertices of G. In the connected variant, Connected Dominating Set (CDS), we additionally demand the subgraph induced by the dominating set to be connected. A considerable part of the algorithmic study on these NP-complete problems has been focused on the design of parameterized and kernelization algorithms. In general, DS is W[2]-complete and therefore it cannot be solved by a parameterized algorithm, unless an unexpected collapse occurs in the Parameterized Complexity Hierarchy (see [18]) and thus also does not admit a kernel.

* Supported by European Research Council (ERC) grant "Rigorous Theory of Preprocessing", reference 267959.

† Co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: "Thales. Investing in knowledge society through the European Social Fund."

However, there are interesting graph classes where FPT-algorithms exist for the DS problem. The project of widening the horizon where such algorithms exist spanned a multitude of ideas that made DS the testbed for some of the most cutting-edge techniques of parameterized algorithm design. For example, the initial study of parameterized subexponential algorithms for DS on planar graphs [13, 24] resulted in the creation of bidimensionality theory characterizing a broad range of graph problems that admit efficient approximate schemes, fixed-parameter algorithms or kernels on a broad range of graphs [14, 15, 20, 22, 21].

One of the first results on linear kernels is the celebrated work of Alber, Fellows, and Niedermeier on DS on planar graphs [1]. This work augmented significantly the interest in proving polynomial (or preferably linear) kernels for other parameterized problems. The result of Alber et al. [1], see also [8], has been extended to a much more general graph classes like graphs of bounded genus [6] and apex-minor free graphs [22]. An important step in this direction was done by Alon and Gutner [2, 26] who obtained a kernel of size $O(k^h)$ for DS on H-minor-free and H-topological-minor free graphs, where the constant h depends on the excluded graph H. Later, Philip, Raman, and Sikdar [31] obtained a kernel of size $O(k^h)$ on $K_{i,j}$-free and d-degenerated graphs, where h depends on i, j and d. In particular, for d-degenerate graphs, a subclass of $K_{i,j}$-free graphs, the algorithm of Philip, Raman, and Sikdar [31] produces a kernel of size $\mathcal{O}(k^{d^2})$. Similarly, the sizes of kernels in [26, 31] are bounded by polynomials in k with degrees depending on the size of the excluded minor H. Alon and Gutner [2] mentioned as a challenging question to characterize the families of graphs for which the dominating set problem admits a linear kernel, i.e. a kernel of size $f(h) \cdot k$, where the function f depends *exclusively* on the graph family. In this direction, there are already results for more restricted graph classes. According to the meta-algorithmic results on kernels introduced in [6], DS has a kernel of size $f(g) \cdot k$ on graphs of genus g. An alternative meta-algorithmic framework, based on bidimensionality theory [14], was introduced in [22], implying the existence of a kernel of size $f(H) \cdot k$ for DS on graphs excluding an apex graph H as a minor. Recently, the result on linear kernels on apex-minor-free graphs was extended to graphs excluding an arbitrary graph H as a minor [23]. Prior to our work, the only result on linear kernels for DS on graphs excluding H as a topological subgraph, was the result of Alon and Gutner in [2] for a very special case $H = K_{3,h}$. See Fig. 1 for the relationship between these classes.

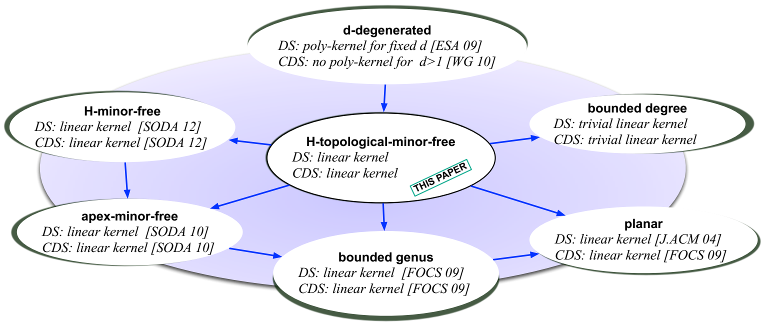

d-degenerated
DS: poly-kernel for fixed d [ESA 09]
CDS: no poly-kernel for d>1 [WG 10]
H-minor-free
DS: linear kernel [SODA 12]
CDS: linear kernel [SODA 12]
H-topological-minor-free
DS: linear kernel
CDS: linear kernel
THIS PAPER
bounded degree
DS: trivial linear kernel
CDS: trivial linear kernel
apex-minor-free
DS: linear kernel [SODA 10]
CDS: linear kernel [SODA 10]
bounded genus
DS: linear kernel [FOCS 09]
CDS: linear kernel [FOCS 09]
planar
DS: linear kernel [J.ACM 04]
CDS: linear kernel [FOCS 09]

Figure 1 Kernels for DS and CDS on classes of sparse graphs. Arrows represent inclusions of classes (where the class at the head is contained in the class at the tail). In the diagram, [J.ACM 04] is referred to the paper of Albers et al. [1], [FOCS 09] to the paper of Bodlaender et al. [6], [SODA 10] and [SODA 12] to the papers of Fomin et al. [22] and [23], [ESA 09] to the paper of Philip et al. [31], and [WG 10] to Cygan et al. [10].

It is tempting to suggest that similar improvements on kernel sizes are possible for more general graph classes like d-degenerated graphs. For example, for graphs of bounded vertex degree, a subclass of d-degenerate graphs, DS has a trivial linear kernel. Unfortunately, for d-degenerate graphs the existence of a linear kernel and even polynomial kernel with the exponent of the polynomial independent of d is very unlikely. By the very recent work of Cygan et al. [9], the kernelization algorithm of Philip, Raman, and Sikdar [31] is essentially tight—existence of a kernel of size $\mathcal{O}(k^{(d-3)(d-1)-\varepsilon})$, would imply that coNP is in NP/poly. In spite of these negative news, we show how to lift the linearity of kernelization for DS from bounded-degree graphs and H-minor free graphs to the class of graphs excluding H as a topological subgraph. Moreover, a modification of the ideas for DS kernelization can be used to obtain a linear kernel for CDS, which is usually a much more difficult problem to handle due to the connectivity constraint. For example, CDS does not have a polynomial kernel on 2-degenerated graphs unless coNP is in NP/poly [10].

The class of graphs excluding H as a topological subgraph is a wide class of graphs containing H-minor-free graphs and graphs of constant vertex degrees. The existence of a linear kernel for DS on this class of graphs significantly extends and improves previous works [23, 26]. The basic idea behind kernelization algorithms on apex-minor-free and minor-free graphs is the bidimensionality of DS. Roughly speaking, the treewidth of these graphs with dominating set k is either $o(k)$ (as in planar, bounded genus or apex-minor-free graphs [14]) or becomes $o(k)$ after applying the irrelevant vertex technique [23]. This idea can hardly work on graphs of bounded degree, and hence on graphs excluding H as a topological subgraph. The reason is that the bound $o(k)$ on the treewidth of such graphs would imply that DS is solvable in subexponential time on graphs of bounded degree, which in turn can be shown to contradict the Exponential Time Hypothesis [28]. This is why the kernelization techniques developed for H-minor-free graphs does not seem to be applicable directly in our case.

2 Preliminaries

In this section we give various definitions which we make use of in the paper. We refer to Diestel's book [16] for standard definitions from Graph Theory. Let G be a graph with vertex set $V(G)$ and edge set $E(G)$. A graph G' is a *subgraph* of G if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. For subset $V' \subseteq V(G)$, the subgraph $G' = G[V']$ of G is called the *subgraph induced by* V' if $E(G') = \{uv \in E(G) \mid u, v \in V'\}$. By $N_G(u)$ we denote the (open) neighborhood of u in graph G. That is, the set of all vertices adjacent to u and by $N[u] = N(u) \cup \{u\}$. Similarly, for a subset $D \subseteq V$, we define $N_G[D] = \cup_{v \in D} N_G[v]$ and $N_G(D) = N_G[D] \setminus D$. We omit the subscripts when they are clear from the context. Throughout the paper, given a graph G and vertex subsets Z and S, whenever we say that a subset Z *dominates all but (everything but)* S then we mean that $V(G) \setminus S \subseteq N[Z]$. Observe that a vertex of S can also be dominated by the set Z.

We denote by K_h the complete graph on h vertices. Also for a given graph G and a vertex subset S, by $K[S]$ we mean a clique on the vertex set S. For an integer $r \geq 1$ and vertex subsets $P, Q \subseteq V(G)$, we say that a subset Q is r*-dominated* by P, if for every $v \in Q$ there is a $u \in P$ such that the distance between u and v is at most r. For $r = 1$, we simply say that Q is dominated by P. We denote by $N^r_G(P)$ the set of vertices r-dominated by P.

Given an edge $e = xy$ of a graph G, the graph G/e is obtained from G by contracting the edge e, that is, the endpoints x and y are replaced by a new vertex v_{xy} which is adjacent to the old neighbors of x and y (except from x and y). A graph H obtained by a sequence of edge-contractions is said to be a *contraction* of G. We denote it by $H \leq_c G$. A graph H is a *minor* of a graph G if H is the contraction of some subgraph of G and we denote it by $H \leq_m G$. We say that a graph G is H*-minor-free* when it does not contain H as a minor.

We also say that a graph class $\mathcal{G}_H$ is *H-minor-free* (or, excludes H as a minor) when all its members are H-minor-free. An *apex graph* is a graph obtained from a planar graph G by adding a vertex and making it adjacent to some of the vertices of G. A graph class $\mathcal{G}_H$ is *apex-minor-free* if $\mathcal{G}_H$ excludes a fixed apex graph H as a minor. A *subdivision* of a graph H is obtained by replacing each edge of H by a path of at least one edge. We say that H is a *topological minor* of G if some subgraph of G is isomorphic to a subdivision of H and denote it by $H \preceq_T G$. A graph G *excludes graph H as a (topological) minor* if H is not a (topological) minor of G. For a graph H, by $\mathcal{C}_H$, we denote all graphs that exclude H as topological minor.

Tree Decompositions. A *tree decomposition* of a graph $G = (V, E)$ is a pair (M, β) where M is a rooted tree and $\beta : V(M) \to 2^V$, such that :

1. $\bigcup_{t \in V(M)} \beta(t) = V$.
2. For each edge $\{u, v\} \in E$, there is a $t \in V(M)$ such that both u and v belong to $\beta(t)$.
3. For each $v \in V$, the nodes in the set $\{t \in V(M) \mid v \in \beta(t)\}$ form a subtree of M.

The following notations are the same as that in [25]. Given a tree decomposition of graph $G = (V, E)$, we define mappings $\sigma, \gamma : V(M) \to 2^V$ and $\kappa : E(M) \to 2^V$. For all $t \in V(M)$, $\sigma(t) = \emptyset$ if t is the root of M else $\sigma(t) = \beta(t) \cap \beta(s)$ if s is the parent of t in M. We also set $\gamma(t) = \bigcup_{u \text{ is a descendant of } t} \beta(u)$. For all $e = uv \in E(M)$, $\kappa(e) = \beta(u) \cap \beta(v)$. For a subgraph M' of M by $\beta(M')$ we denote $\cup_{t \in V(M')} \beta(t)$.

Let (M, β) be a tree decomposition of a graph G. The *width* of (M, β) is $min\{|\beta(t)| - 1 \mid t \in V(M)\}$, and the *adhesion* of the tree decomposition is $max\{|\sigma(t)| \mid t \in V(M)\}$. We use $\mathbf{tw}(G)$ to denote the treewidth of the input graph, that is the minimum width of a tree-decomposition of G. For every node $t \in V(M)$, the *torso* at t is the graph $\tau(t) := G[\beta(t)] \cup E(K[\sigma(t)]) \cup \bigcup_{u \text{ child of } t} E(K[\sigma(u)])$.

Given a graph G, we say that a set $X \subseteq V(G)$ is an *r-protrusion* of G if $\mathbf{tw}(G[X]) \leq r$ and the number of vertices in X with a neighbor in $V(G) \setminus X$ is at most r.

Known Decomposition Theorem. The decomposition theorem that we use extensively for our proofs is given in the next theorem.

▶ **Theorem 1** ([25, 32]). *For every graph H, there exists a constant h, depending only on the size of H, such that for every graph G with $H \not\preceq_T G$, there is a tree decomposition (M, β) of adhesion at most h such that for all $t \in V(M)$, one of the following conditions is satisfied:*

1. *$\tau(t)$ excludes a clique of size h as a minor.*
2. *$\tau(t)$ has at most h vertices of degree at least h (we call these vertices* apices *of $\tau(t)$).*

Moreover, if G is H-minor free graph G then nodes of second type do not exist. Furthermore, there is an algorithm that, given graphs G, H of sizes n and h respectively, computes such a tree decomposition in time $h \cdot n^{O(1)}$ and computes the corresponding apex set Z_t of size at most h for every bag $\tau(t)$.

Actually, we can assume that in (M, β), for any $x, y \in V(M)$, $\beta(x) \not\subseteq \beta(y)$. That is, no bag is contained in other. See [18, Lemma 11.9] for the proof.

3 An approximation algorithm for DS on $H \not\preceq_T G$

In this section we give a constant factor approximation for DS on $\mathcal{C}_H$. It is well known that graphs in $\mathcal{C}_H$ have bounded degeneracy. In a recent manuscript a subset of the authors together with others show that DS has a $O(d^2)$ factor approximation algorithm on d-degenerate graphs [29]. To make this paper self contained we provide an approximation algorithm for DS on $\mathcal{C}_H$ here. The main idea of the approximation algorithm is to first compute the tree-decomposition (M, β) given by Theorem 1 for G and then suitably select a bag of this decomposition that still contains a vertex that is not dominated. Then we locally find an approximate dominating set for this bag by using either an approximation algorithm for DS on H-minor free graphs or on graphs of almost bounded degree. We apply this step iteratively and finally show that the dominating set returned by the algorithm is indeed a constant factor approximation. This results in the following lemma.

▶ **Lemma 2.** *Let H be a graph. Then there exists a constant $\eta(H)$ depending only on $|H|$ such that* DS *admits a $\eta(H)$-factor approximation algorithm on $\mathcal{C}_H$.*

4 Generalized Protrusions

A parameterized graph problem Π can be seen as a subset of $\Sigma^* \times \mathbb{Z}^+$ where, in each instance (x, k) of Π, x encodes a graph and k is the parameter (we denote by $\mathbb{Z}^+$ the set of all non-negative integers). Here we define the notion of *t-boundaried graphs* and various operations on them.

▶ Definition 1. [**t-Boundaried Graphs**] A t-boundaried graph is a graph G with a set $B \subseteq V(G)$ of at most t distinguished vertices and an injective labeling from B to the set $\{1, \dots, t\},$. The set B is called the *boundary* of G and vertices in B are called *boundary vertices* or *terminals*. Given a t-boundaried graph G we denote its boundary by $\delta(G)$. We use the notation $\mathcal{F}_t$ to denote the class of all t-boundaried graphs.

▶ Definition 2. [**Gluing by** $\oplus$] Let G_1 and G_2 be two t-boundaried graphs. We denote by $G_1 \oplus G_2$ the graph obtained by taking the disjoint union of G_1 and G_2 and identifying equally-labeled vertices of the boundaries of G_1 and G_2. We stress that, in $G_1 \oplus G_2$, there is an edge between two labeled vertices if there is an edge between them in G_1 or in G_2. When we are dealing with a gluing operation we use the term *common boundary* in G_1 and G_2 in order to denote the set of identified vertices in $G_1 \oplus G_2$.

▶ Definition 3. [**Gluing by** $\oplus_\delta$] The *boundaried gluing operation* $\oplus_\delta$ is similar to the normal gluing operation, but results in a t-boundaried graph rather than a graph. Specifically $G_1 \oplus_\delta G_2$ results in a t-boundaried graph where the graph is $G = G_1 \oplus G_2$ and a vertex is in the boundary of G if it was in the boundary of G_1 or G_2. Vertices in the boundary of G keep their label from G_1 or G_2.

Let $\mathcal{G}$ be a class of (not boundaried) graphs. By slightly abusing notation we say that a boundaried graph *belongs in a graph class* $\mathcal{G}$ if the underlying graph belongs in $\mathcal{G}$. By $\partial_G(X)$, we denote the *boundary* of X in G, that is the vertices of G that are not in X and are neighbours of vertices in X.

▶ Definition 4. [**Replacement**] Let G be a t-boundaried graph containing a set $X \subseteq V(G)$ such that $\partial_G(X) = \delta(G)$. Let G_1 be a t-boundaried graph. The result of *replacing X with* G_1 is the graph $G^\star \oplus G_1$, where $G^\star = G \setminus (X \setminus \partial(X))$ is treated as a t-boundaried graph, where $\delta(G^\star) = \delta(G)$.

▶ Definition 5. [**Equivalence of t-boundaried graphs**] Let Π be a parameterized graph problem whose instances are pairs of the form (G, k). Given two t-boundaried graphs G_1, G_2, we say that $G_1 \equiv_{\Pi,t} G_2$ if there exist a *transposition constant* $c \in \mathbb{Z}$ such that $\forall (F, k) \in \mathcal{F}_t \times \mathbb{Z} (G_1 \oplus F, k) \in \Pi \Leftrightarrow (G_2 \oplus F, k + c) \in \Pi$.

Note that for every t, the relation $\equiv_{\Pi,t}$ on t-boundaried graphs is an equivalence relation. Next we define a notion of “transposition-minimality” for the members of each equivalence class of $\equiv_{\Pi,t}$.

▶ Definition 6. [**Progressive representatives**] Let Π be a parameterized graph problem whose instances are pairs of the form (G, k) and let $\mathcal{C}$ be some equivalence class of $\equiv_{\Pi,t}$ for some $t \in \mathbb{Z}^+$. We say that $J \in \mathcal{C}$ is a *progressive representative* of $\mathcal{C}$ if for every $H \in \mathcal{C}$ there exist $c \in \mathbb{Z}^-$, such that $\forall (F, k) \in \mathcal{F}_t \times \mathbb{Z}$ $(H \oplus F, k) \in \Pi \Leftrightarrow (J \oplus F, k + c) \in \Pi$.

▶ **Lemma 3** ([6]). *Let* Π *be a parameterized graph problem whose instances are pairs of the form* (G, k) *and let* $t \in \mathbb{Z}^+$. *Then each equivalence class of* $\equiv_{\Pi,t}$ *has a progressive representative.*

After Lemma 3 we are in position to give the following definitions.

▶ Definition 7. A parameterized graph problem Π whose instances are pairs of the form (G, k) has *Finite Integer Index* (or simply has *FII*), if and only if for every $t \in \mathbb{Z}^+$, the equivalence relation $\equiv_{\Pi,t}$ is of finite index, that is, has a finite number of equivalence classes. For each $t \in \mathbb{Z}^+$, we define $\mathcal{S}_t$ to be a set containing exactly one progressive representative of each equivalence class of $\equiv_{\Pi,t}$. We say that a parameterized graph problem Π is *positive monotone* if for every graph G there exists a unique $\ell \in \mathbb{N}$ such that for all $\ell' \in \mathbb{N}$ and $\ell' \geq \ell$, $(G, \ell') \in \Pi$ and for all $\ell' \in \mathbb{N}$ and $\ell' < \ell$, $(G, \ell') \notin \Pi$. A parameterized graph problem Π is *negative monotone* if for every graph G there exists a unique $\ell \in \mathbb{N}$ such that for all $\ell' \in \mathbb{N}$ and $\ell' \geq \ell$, $(G, \ell') \notin \Pi$ and for all $\ell' \in \mathbb{N}$ and $\ell' < \ell$, $(G, \ell') \in \Pi$. Π is monotone if it is either positive monotone or negative monotone. We denote the integer ℓ by $\textsc{Thr}(G)$. Let Π be a monotone parameterized graph problem that is FII. Let $\mathcal{S}_t$ be a set containing exactly one progressive representative of each equivalence class of $\equiv_{\Pi,t}$. For a t-boundaried graph G by $\kappa(G)$ we denote $\max_{G' \in \mathcal{S}_t} \textsc{Thr}(G \oplus G')$.

▶ **Lemma 4.** *Let* Π *be a monotone parameterized graph problem that is FII. Furthermore, let* $\mathcal{A}$ *be an algorithm for* Π *that given a pair* (G, k) *decides whether it is in* Π *in at most* $f(|V(G)|, k)$ *steps for some function* $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. *Then for every* $t \in \mathbb{N}$, *there exists a* $\xi_t \in \mathbb{Z}^+$ *(depending on* Π *and* t*), and an algorithm that, given a* t*-boundaried graph* G *with* $|V(G)| > \xi_t$, *outputs, in* $O(\kappa(G)(f(|V(G)| + \xi_t, \kappa(G)))$ *steps, a* t*-boundaried graph* G^* *such that* $G \equiv_{\Pi,t} G^*$ *and* $|V(G^*)| < \xi_t$. *Moreover we can compute the translation constant* c *from* G *to* G^* *in the same time.*

We remark that the algorithm whose existence is guaranteed by the Lemma 4 assumes that the set $\mathcal{S}_t$ of representatives are hardwired in the algorithm and that in general there is no procedure that for FII problems Π outputs such a representative set.

5 Slice-Decomposition

In this section our objective is to show that in polynomial-time we can partition the graph G satisfying certain properties that will be useful later. To obtain our decomposition we need to use a more general notion of protrusion. More precisely, we need the following kind of protrusions.

▶ Definition 8. [r-DS-**protrusion**] Given a graph G, we say that a set $X \subseteq V(G)$ is an r-DS-*protrusion* of G if the number of vertices in X with a neighbor in $V(G) \setminus X$ is at most r and there exists a subset $S \subseteq X$ of size at most r such that S is a dominating set of $G[X]$.

The notion of r-DS-protrusion X differs from normal protrusion in the following way. In the normal protrusion we demand that $\mathbf{tw}(X)$ is at most r while in the r-DS-protrusion we demand that the dominating set of the graph induced on X is small. We can similarly define the notion of r-Π-protrusion for various other graph problems Π. The next question is what do we achieve if we get a large r-DS-protrusion (or r-CDS-protrusion). The next lemma shows that in that case we can replace it with an equivalent small graph. More precisely we have the following.

▶ **Lemma 5.** *Let* H *be a fixed graph. For every* $t \in \mathbb{Z}^+$, *there exist a* $\xi_t \in \mathbb{Z}^+$ *(depending on* DS *(*CDS*),* t *and* H*), and an algorithm* $\mathcal{A}$ *such that given a* t-DS*-protrusion (*t-CDS-*protrusion)* X, *with* $|X| > \xi_t$, *and* $H \not\preceq_T X$, $\mathcal{A}$ *outputs in* $\mathcal{O}(|X|)$ *time (*$|X|^{\mathcal{O}(1)}$*) time), a* t*-boundaried graph* X' *such that* $X \equiv_{\mathrm{DS},t} X'$ *(*$X \equiv_{\mathrm{CDS},t} X'$*) and* $|X'| \leq \xi_t$. *Moreover in the same time we can also find the translation constant* c *from* X *to* X'.

Let (M, β) be a tree decomposition of a graph G. For a subtree M_i of M, we define $\mathcal{E}(M_i)$ as the set of edges in M that have exactly one endpoint in $V(M_i)$. Furthermore we define $R_i^+ = \beta(M_i)$ and $\tau(M') := G[R_i^+] \cup \bigcup_{e \in \mathcal{E}(M_i)} K[\kappa(e)]$. Our main objective in this section is to obtain the following (α, β)-*slice decomposition* for $\alpha = \beta = \mathcal{O}(k)$.

▶ Definition 9. $[(\alpha, \beta)$**-slice decomposition**] Let G be a graph with $H \not\preceq_T G$ and let (M, β) be the tree decomposition given by Theorem 1. An (α, β)-*slice decomposition* of a graph G is a collection $\mathcal{P}$ of pairwise disjoint connected subtrees $\{M_1, \dots, M_\alpha\}$ of M such that the following holds.

- Each of $\tau(M_i)$ is either H^*-minor free for some graph H^* whose size only depends on h or $\tau(M_i)$ has at most h vertices of degree at least h.
- $\sum_{i=1}^{\rho}(\sum_{e \in \mathcal{E}(M_i)} |\kappa(e)|) \leq \beta$.

We call the sets R_i^+, $i \in \{1, \dots, \rho\}$, *slices* of $\mathcal{P}$.

Essentially, the slice-decomposition allows us to partition the input graph G into subgraphs $C_0, C_1, \dots, C_\ell$, such that $|C_0| = \mathcal{O}(k)$; for every $i \geq 1$, the neighbourhood $N(C_i) \subseteq C_0$, and $\sum_{1 \leq i \leq \ell} |N(C_i)| = \mathcal{O}(k)$. Now we define a notion of measure.

▶ Definition 10. Let (M, β) be the tree decomposition of a graph G given by the Theorem 1. For a subset $Q \subseteq V(G)$ and a subtree M' of M we define $\mu(M', Q) = |\beta(M') \cap Q|$. If we delete an edge $e = uv \in E(M)$ from the tree M then we get two trees. We call the trees as M_u and M_v based on whether they contain u or v.

▶ **Lemma 6.** *Let H be a fixed graph and $\mathcal{C}_H$ be the class of graphs excluding H as a topological minor. Then there exist two constants δ_1 and δ_2 (depending on the problem* DS *(*CDS*)) and a polynomial time algorithm such that given a yes instance (G, k) of* DS *(*CDS*), can either find*

- *a $(\delta_1 k, \delta_2 k)$-slice decomposition; or*
- *a $2h$-*DS*-protrusion (or $2h$-*CDS*-protrusion) of size more than ξ_{2h} or;*
- *a h'-protrusion of size more than $\xi_{h'}$ where h' depends only on h.*

Sketch of the proof. To obtain the slice-decomposition we introduce our marking scheme as follows.

1. Apply Lemma 2 on the input graph G and compute a $\eta(H)$-factor approximation (connected) dominating set D for G.
2. Use Theorem 1 and compute a tree-decomposition (M, β). We call a tree edge $e = uv \in E(M)$ *heavy* if $\mu(M_u, D) \geq h + 1$ and $\mu(M_v, D) \geq h + 1$. Mark all the edges of M that are heavy. We use $\mathcal{F}$ to denote all the set of edges that have been marked.

Let M^* be the subtree (requires proof) induced on all the heavy edges. We use this tree M^* to obtain the decomposition. If (G, k) is a yes instance then one can show that the number of leaves in the tree M^* is upper bounded by $\mathcal{O}(k)$. This immediately implies that the number of maximal paths consisting only of degree 2 vertices is upper bounded by $\mathcal{O}(k)$. We show that if any of these paths is too long then we can obtain a $2h$-DS-protrusion of large size. This implies that the size of the tree M^* is upper bounded by $\mathcal{O}(k)$. Now we delete all the edges appearing in M^* from M. This breaks the tree M into $\mathcal{O}(k)$ subtrees, $\mathcal{P} = \{M_1, \dots, M_\alpha\}$. We argue that these subtrees form the partition described in the definition of slice decomposition. To show that $\sum_{i=1}^{\rho}(\sum_{e \in \mathcal{F}(M_i)} |\kappa(e)|) \leq \mathcal{O}(k)$, we use the fact that any edge of M^* sees at most two trees among $\mathcal{P}$. ◀

6 Final Kernel

In this section we use slice-decomposition obtained in the last section and the reduction rules used in [23] to obtain linear kernels for DS and CDS. We first outline our algorithm for DS and then explain how we can obtain a linear kernel for CDS.

Kernelization Algorithm for DS. Given an instance (G, k) of DS we first apply Lemma 2 and find a dominating set D of G. If $|D| > \eta(H)k$ we return that (G, k) is a NO instance to DS. Else, we apply Lemma 6 and

- either find $(\delta_1 k, \delta_2 k)$-slice decomposition; or
- a $2h$-DS-protrusion X of G (or $2h$-CDS-protrusion) of size more than ξ_{2h}; or
- a h'-protrusion of size more than $\xi_{h'}$ where h' depends only on h.

In the second case we apply Lemma 5. Given X we apply Lemma 5 and obtain a boundaried graph X' such that $|X'| \le \xi_{2h}$ and $X \equiv_{\mathrm{DS},2h} X'$ ($X \equiv_{\mathrm{CDS},2h} X'$). We also compute the translation constant c between X and X'. Now we replace the graph X with X' and obtain a new equivalent instance $(G', k + c)$. (Recall that c is a non-positive integer). In the third case we apply the protrusion replacement lemma of [6, Lemma 7] to obtain a new equivalent instance (G', k') for $k' \le k$ with $|V(G')| < |V(G)|$. We repeat this process until Lemma 6 returns a slice-decomposition. For simplicity we denote by (G, k) itself the graph on which Lemma 6 returns the slice-decomposition. Since the number of times this process can be repeated is upper bounded by $n = |V(G)|$, we can obtain $(\delta_1 k, \delta_2 k)$-slice decomposition for (G, k) in polynomial-time.

Let $\mathcal{P}$ be the pairwise disjoint connected subtrees $\{M_1, \dots, M_\alpha\}$ of M coming from the slice-decomposition of G. Recall that $R_i^+ = \beta(M_i)$. Let $Q_i = \bigcup_{e \in \mathcal{E}(M_i)} \kappa(e)$, $B_i = (D \cap R_i^+) \cup Q_i$ and $b_i = |B_i|$. In this section we will treat $G_i := G[R_i^+]$ as a graph with boundary B_i. Observe that B_i is a dominating set for G_i.

We have two kinds of graphs G_i. In one case we have that G_i is H^*-minor free for a graph H^* whose size only depends on h. In the other case we have that the graph G_i has at most h' vertices of degree at least h'. To obtain our kernel we will show the following two lemmata.

▶ **Lemma 7.** *There exists a constant δ and a polynomial time algorithm that, given a graph G with boundary S where S is a dominating set for G and G has at most h' vertices of degree at least h', outputs a graph G' with boundary S such that $G' \equiv_{\mathrm{DS},|S|} G$ and $|V(G')| \le \delta|S|$. Furthermore we can also compute the translation constant c of G and G' in polynomial-time.*

▶ **Lemma 8.** *There exists a constant δ and a polynomial time algorithm that, given an H-minor free graph G with boundary S where S is a dominating set for G, outputs a graph G' with boundary S such that $G' \equiv_{\mathrm{DS},|S|} G$ and $|V(G')| \le \delta|S|$. Furthermore we can also compute the translation constant c of G and G' in polynomial-time.*

Once we have proved Lemmata 7 and 8, we obtain the linear sized kernel for DS as follows. Given the graph G we obtain the slice-decomposition and check if any of G_i has size more than δb_i. If yes then we either apply Lemma 7 or Lemma 8 based on the type of G_i and obtain a graph G_i' such that $G_i' \equiv_{\mathrm{DS},b_i} G_i$ and $|V(G_i')| \le \delta b_i$. We think $G = G_i \oplus G^\star$, where $G^\star = G \setminus (R_i^+ \setminus B_i)$ as a b_i-boundaried graph with boundary B_i. Then we obtain a smaller equivalent graph $G' = G^\star \oplus G_i'$ and $k' = k + c$. After this we can repeat the whole process once again. This implies that when we can not apply Lemmata 8 or 7 on (G, k) we have that each of $|V(G_i)| \le \delta b_i$. Furthermore notice that $\cup_{i=1}^{\alpha} R_i^+ = V(G)$. This implies that in this case we have the following: $\sum_{i=1}^{\alpha} |R_i^+| \le \delta \sum_{i=1}^{\alpha} b_i = \delta(\sum_{i=1}^{\alpha}(|Q_i| + |(D \cap R_i^+) \setminus Q_i|)) = \delta(\sum_{i=1}^{\alpha} |Q_i| + \sum_{i=1}^{\alpha} |(D \cap R_i^+) \setminus Q_i|) \le \delta\delta_2 k + \delta\eta(H)k = \mathcal{O}(k)$. This bring us to the following theorem.

▶ **Theorem 9.** DS *admits a linear kernel on graphs excluding a fixed graph H as a topological minor.*

It only remains to prove Lemmata 7 and 8 to complete the proof of Theorem 9.

Irrelevant Vertex Rule and proof for Lemma 7. For the proofs of Lemmata 7 and 8 we need to use an irrelevant vertex rule developed in [23]. Furthermore, the proof of Lemma 8 is essentially a reformulation of the results presented in [23].

If the graph G is $K_{h'}$-minor free then the irrelevant vertex rule will be used in a recursive fashion. In each recursive step it is used in order to reduce the treewidth of torsos and hence also the entire graph. Then the graph is split in two pieces and the procedure is applied recursively to the two pieces. In the bottom of the recursion when the graph becomes smaller but still big enough then we apply Lemma 5 on it and obtain an equivalent instance.

Let G be a graph given with its tree-decomposition (M, β) as described in Theorem 1, and $\tau(t)$ be one of its torsos. Let S be a dominating set of G, and $Z_t = A$, $|A| \leq h$, be the set of apices of $\tau(t)$. The reduction rule essentially "preserves" all dominating sets of size at most $|S|$ in G, without introducing any new ones. To describe the reduction rule we need several definitions. The first step in our reduction rule is to classify different subsets A' of A into feasible and infeasible sets. The intuition behind the definition is that a subset A' of A is feasible if there exists a set D in G of size at most $|S| + 1$ such that D dominates all but S and $D \cap A = A'$. However, we cannot test in polynomial-time whether such a set D exists. We will therefore say that a subset A' of A is *feasible* if the 2-approximation for DS on H-minor-free graphs [20] outputs a set D of size at most $2(|S|+2)$ such that D dominates $V(G) \setminus (A \cup S)$ and $D \cap A = A'$. Observe that if such a set D of size at most $|S| + 1$ exists then A' is surely feasible, while if no such set D of size at most $2|S| + 2$ exists, then A' is surely not feasible. We will frequently use this in our arguments. Let us remark that there always exists a feasible set $A' \subseteq A$. In particular, $A' = S \cap A$ is feasible since S dominates G. For feasible sets A' we will denote by $D(A')$ the set D output by the approximation algorithm.

For every subset $A' \subseteq A$, we select a vertex v of G such that $A' \subseteq N_G[v]$. If such a vertex exist, we call it a *representative* of A'. Let us remark that some sets can have no representatives and some distinct subsets of A may have the same representative. We define R to be the set of representative vertices for subsets of A. The size of R is at most $2^{|A|}$. For $A' \subseteq A$, the set of *dominated vertices* (by A') is $W(A') = N(A') \setminus A$. We say that vertex $v \in V(G) \setminus A$ is *fully dominated* by A' if $N[v] \setminus A \subseteq W(A')$. A vertex $w \in V(G) \setminus A$ is *irrelevant with respect to* A' if $w \notin R$, $w \notin S$, and w is fully dominated by A'. Now we are ready to state the irrelevant vertex rule.

Irrelevant Vertex Rule: If a vertex w is irrelevant with respect to every feasible $A' \subseteq A$, then delete w from G.

▶ **Lemma 10.** *Let S be a dominating set in G, and G' be the graph obtained by applying the Irrelevant Vertex Rule on G, where w was the deleted vertex. Then $G' \equiv_{\mathrm{DS},|S|} G$.*

Proof. Let the transposition constant be 0. To show that $G' \equiv_{\mathrm{DS},|S|} G$, we show that given a $|B|$-boundaried graph G_1 and a positive integer ℓ we have that $(G \oplus G_1, \ell) \in \mathrm{DS} \Leftrightarrow (G' \oplus G_1, \ell) \in \mathrm{DS}$. Let $Z \subset V(G \oplus G_1)$ be a dominating set for $G \oplus G_1$ of size at most ℓ. Let $Z_1 = V(G) \cap Z$. If $|Z_1| > |S|$ then $(Z \setminus Z_1) \cup S$ is a smaller dominating set for $G \oplus G_1$. Therefore we assume that $|Z_1| \leq |S|$. Let $A' = Z \cap A$, and observe that A' is feasible because Z_1 dominates all but S. If $w \notin Z$, then $Z' = Z$ is a dominating set of size at most ℓ for $G' \oplus G_1$. So assume $w \in Z$. Observe that $w \in Z_1$ and $w \notin S$ and therefore all the neighbors of w lie in G. Since w is irrelevant with respect to all feasible subsets of A and A' is feasible, we have that w is irrelevant with respect to A'. Hence $N_{G\oplus G_1}(w) \setminus N_{G\oplus G_1}(Z \setminus w) \subseteq A$. There is a representative $w' \in R$, $w' \neq w$ (since $w \notin R$), such that $(N_{G\oplus G_1}(w) = N_G(w)) \cap A \subseteq N_G(w') \cap A$. Hence $Z' = (Z \cup \{w'\}) \setminus \{w\}$ is a dominating set of $G' \oplus G_1$ of size at most ℓ.

Now, let $Z' \subseteq V(G' \oplus G_1)$ be a dominating set of size at most ℓ for $G' \oplus G_1$. Let $Z'_1 = V(G') \cap Z'$. As in the forward direction we can assume that $|Z'_1| \leq |S|$. We show that Z' also dominates w in $G \oplus G_1$. Specifically $Z'_1 \cup \{w\}$ is a dominating set of all but S in G of size at most $|S| + 1$ so $Z'_1 \cap A$ is feasible. Since $\{w\}$ is irrelevant with respect to $Z'_1 \cap A$, we have $w \in N_G(Z'_1 \cap A)$ and thus Z' is a dominating set for $G' \oplus G_1$ of size at most ℓ. This concludes the proof. ◀

For a graph G and its dominating set S, we apply the Irrelevant Vertex Rule exhaustively on all torsos of G, obtaining an induced subgraph G' of G. By Lemma 10 and transitivity of $\equiv_{\mathrm{DS},t}$ we have that $G' \equiv_{\mathrm{DS},|S|} G$. We now prove that a graph G which can not be reduced by the irrelevant vertex rule has a property that each of its torso has a small 2-dominating set (the proof is omitted in this extended abstract).

▶ **Lemma 11.** *There is a polynomial-time algorithm that for a given graph G and a dominating set S of G, outputs graph G' such that $G' \equiv_{\mathrm{DS}} G$ and for every torso $\tau(t)$ of the tree-decomposition (M, β) of G, we have that $\tau(t) \setminus Z_t$ has a 2-dominating set of size $\mathcal{O}(|S|)$. Furthermore if G is a H-minor free graph then $\mathbf{tw}(G) = \mathcal{O}(\sqrt{|S|})$.*

Proof of Lemma 7. We apply Lemma 11 to G with a decomposition that has a single bag containing the entire graph and the apices A of the bag being the vertices of degree at least h'. By Lemma 11, $G \setminus A$ has a 2-dominating set of size $\delta_3|S|$. Since all vertices of $G \setminus A$ have degree at most h' it follows that $|V(G)| \leq h' + \delta_3 h|S|\delta_3 h^2|S| \leq \delta|S|$. ◀

Kernelization algorithm for CDS**.** To obtain kernelization algorithm for CDS the only thing that remains to show are results analogous to Lemmata 8 and 7 for DS. However to obtain this we need to apply reduction rules developed in [23] for CDS. Finally we need to adapt the proofs of Lemmata 11, 12, 13 and 14 given in the full-version available at [23] with the new perspective. Two of these lemmata essentially shows the correctness of reduction rules for CDS and that every torso has 2-dominating set of size at most $\mathcal{O}(|S|)$. Here S is a connected dominating set of the input graph G. The only result that is not proved in [23] is the result analogous to Lemma 7 for DS. However, the size of a dominating set is at most the size of a connected dominating set. After this the proof for the case that given a graph G with at most h' vertices of degree at least h' we can return a canonically equivalent graph G' is verbatim to the proof of Lemma 7. We omit these adaptation details from this extended abstract.

▶ **Theorem 12.** CDS *admits a linear kernel on graphs excluding a fixed graph H as a topological minor.*

7 Conclusions

In this paper we give linear kernels for two widely studied parameterized problems, namely DS and CDS, for every graph class that excludes some graph as a topological minor. The emerging questions are the following two: (**1**) Can our techniques be extended to more general sparse graph classes? (**2**) Can our techniques be applied to more general families of parameterized problems? We believe that any step towards resolving the first question should be based on significant graph-theoretical advances. Our results make use of the decomposition theorem of Grohe and Marx in [25] that, in turn, can be seen as an extension of seminal results of the Graph Minor Series by Robertson and Seymour [32]. So far no similar structural theorem is known for more general sparse graph classes. We also believe that a broadening of the kernelization horizon for these two problems without the use of some tree-based structural characterization of sparsity requires completely different ideas.

The first move towards resolving the second question is to extend our techniques for more variants of the dominating set problem. Natural candidates in this direction could be the r-Domination problem (asking for a set S that is within distance r from any other vertex of the graph), the Independent Domination problem (asking for a dominating set that induces an edgeless graph), or, more interestingly, the Cycle Domination problem (asking for a set S that dominates at least one vertex from each cycle of G). However, a more general meta-algorithmic framework, including general families of parameterized problems, seems to be far from reach.

References

1 J. Alber, M. R. Fellows, and R. Niedermeier. Polynomial-time data reduction for dominating sets. *J. ACM*, 51:363–384, 2004.

2 N. Alon and S. Gutner. Kernels for the dominating set problem on graphs with an excluded minor. Technical Report TR08-066, ECCC, 2008.

3 S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese. An algebraic theory of graph reduction. *Journal of the ACM*, 40:1134–1164, 1993.

4 H. L. Bodlaender and B. de Fluiter. Reduction algorithms for constructing solutions in graphs with small treewidth. pages 199–208, 1996.

5 H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.

6 H. L. Bodlaender, F. V. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh, and D. M. Thilikos. (Meta) Kernelization. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2009)*, pages 629–638. IEEE, 2009.

7 H. L. Bodlaender and B. van Antwerpen-de Fluiter. Reduction algorithms for graphs of small treewidth. *Information and Computation*, 167:86–119, 2001.

8 J. Chen, H. Fernau, I. A. Kanj, and G. Xia. Parametric duality and kernelization: Lower bounds and upper bounds on kernel size. *SIAM J. Comput.*, 37:1077–1106, 2007.

9 M. Cygan, F. Grandoni, and D. Hermelin. Tight kernel bounds for problems on graphs with small degeneracy. Manuscript, 2012.

10 M. Cygan, M. Pilipczuk, M. Pilipczuk, and J. Wojtaszczyk. Kernelization hardness of connectivity problems in d-degenerate graphs. In *Proceedings of the 36th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2010)*, volume 6410 of *Lect. Notes Comp. Sc.*, pages 147–158. Springer, 2010.

11 B. de Fluiter. *Algorithms for Graphs of Small Treewidth.* PhD thesis, Utrecht University, 1997.

12 H. Dell and D. van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC 2010)*, pages 251–260, 2010.

13 E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos. Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs. *ACM Trans. Algorithms*, 1(1):33–47, 2005.

14 E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. ACM*, 52(6):866–893, 2005.

15 E. D. Demaine and M. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *The Computer Journal*, 51(3):332–337, 2007.

16 R. Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, third edition, 2005.

17 M. R. Fellows and M. A. Langston. An analogue of the myhill-nerode theorem and its use in computing finite-basis characterizations (extended abstract). In *FOCS*, pages 520–525, 1989.

18 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2006.

19 F. V. Fomin, D. Lokshtanov, N. Misra, G. Philip, and S. Saurabh. Hitting forbidden minors: Approximation and kernelization. In *Proceedings of the 8th International Symposium on Theoretical Aspects of Computer Science (STACS 2011)*, volume 9 of *LIPIcs*, pages 189–200. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

20 F. V. Fomin, D. Lokshtanov, V. Raman, and S. Saurabh. Bidimensionality and EPTAS. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2011)*, pages 748–759. SIAM, 2010.

21 F. V. Fomin, D. Lokshtanov, and S. Saurabh. Bidimensionality and geometric graphs. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 1563–1575. SIAM, 2012.

22 F. V. Fomin, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Bidimensionality and kernels. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2010)*, pages 503–510. ACM-SIAM, 2010.

23 F. V. Fomin, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Linear kernels for (connected) dominating set on H-minor-free graphs. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, http://www.ii.uib.no/ daniello/.

24 F. V. Fomin and D. M. Thilikos. Dominating sets in planar graphs: Branch-width and exponential speed-up. *SIAM J. Comput.*, 36:281–309, 2006.

25 M. Grohe and D. Marx. Structure theorem and isomorphism test for graphs with excluded topological subgraphs. In *STOC*, pages 173–192, 2012.

26 S. Gutner. Polynomial kernels and faster algorithms for the dominating set problem on graphs with an excluded minor. In *Proceedings of the 4th Workshop on Parameterized and Exact Computation (IWPEC 2009)*, Lect. Notes Comp. Sc., pages 246–257. Springer, 2009.

27 T. W. Haynes, S. T. Hedetniemi, and P. J. Slater. *Fundamentals of domination in graphs*. Marcel Dekker Inc., New York, 1998.

28 R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

29 M. Jones, D. Lokshtanov, M. S. Ramanujan, S. Saurabh, and O. Suchy. Parameterized complexity of directed steiner tree on sparse graphs. Manuscript, 2012.

30 E. J. Kim, A. Langer, C. Paul, F. Reidl, P. Rossmanith, I. Sau, and S. Sikdar. Linear kernels and single-exponential algorithms via protrusion decompositions. *CoRR*, abs/1207.0835, 2012.

31 G. Philip, V. Raman, and S. Sikdar. Solving dominating set in larger classes of graphs: FPT algorithms and polynomial kernels. In *Proceedings of the 17th Annual European Symposium on Algorithms (ESA 2009)*, volume 5757 of *Lect. Notes Comp. Sc.*, pages 694–705. Springer, 2009.

32 N. Robertson and P. D. Seymour. Graph minors. XVI. Excluding a non-planar graph. *J. Combin. Theory Ser. B*, 89(1):43–76, 2003.

The PCP theorem for NP over the reals*

Martijn Baartse and Klaus Meer[1]

1 **Computer Science Institute, BTU Cottbus**
Platz der Deutschen Einheit 1
D-03046 Cottbus, Germany
`martijnbaartse@msn.com,meer@informatik.tu-cottbus.de`

Abstract

In this paper we show that the PCP theorem holds as well in the real number computational model introduced by Blum, Shub, and Smale. More precisely, the real number counterpart $\mathrm{NP}_{\mathbb{R}}$ of the classical Turing model class NP can be characterized as $\mathrm{NP}_{\mathbb{R}} = \mathrm{PCP}_{\mathbb{R}}(O(\log n), O(1))$. Our proof structurally follows the one by Dinur for classical NP. However, a lot of minor and major changes are necessary due to the real numbers as underlying computational structure. The analogue result holds for the complex numbers and $\mathrm{NP}_{\mathbb{C}}$.

1998 ACM Subject Classification F.2.2 Complexity of Proof Procedures

Keywords and phrases PCP, real number computation, systems of polynomials

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.104

1 Introduction

One of the major achievements in theoretical computer science within the last two decades certainly was the PCP theorem. It gave a new characterization of the complexity class NP via so called probabilistically checkable proofs and had tremendous impact on the field of approximation algorithms in combinatorial optimization.

Starting point for the present paper is the real number model of computation and its related complexity theory as introduced by Blum, Shub, and Smale, henceforth called BSS-model for short, see [6, 5]. The model focusses on algebraic aspects of computation over arbitrary structures, and here in particular the real numbers. As with the Turing model the major open question in this real number framework is whether the real complexity classes $\mathrm{P}_{\mathbb{R}}$ and $\mathrm{NP}_{\mathbb{R}}$ of problems decidable and verifiable, respectively, in polynomial time are different.

The definition of probabilistically checkable proofs makes sense as well in the real number model. Given the importance of the classical PCP theorem it is only natural to ask whether the corresponding characterization holds as well in the BSS framework for $\mathrm{NP}_{\mathbb{R}}$. The goal of this paper is to prove the PCP theorem in this setting.

1.1 Previous work and outline of proof

For the classical PCP theorem, i.e., the equality $\mathrm{PCP}(O(\log n), O(1)) = \mathrm{NP}$ in the Turing model two different proofs are known, the original proof by Arora et al. [3, 2] and, more recently, the groundbreaking new proof by Dinur [7]. The first PCP type theorem for the real number model was given in [9]. There, the existence of so called transparent long proofs

* We gratefully acknowledge support of both authors by project ME 1424/7-1 of the Deutsche Forschungsgemeinschaft DFG.

for the real counterpart $\mathrm{NP}_{\mathbb{R}}$ of NP is shown, see also [4]. Due to the reals as underlying structure the proof needs considerable technical changes compared to the analogue discrete result. The latter are mainly caused by the presence of unstructured domains of certain linear real functions; on these domains, invariants of the uniform distribution on the vector space $GF(2^n)$ are lost. Those invariants, however, play a central role in proving the Turing model result.

The first characterization of real $\mathrm{NP}_{\mathbb{R}}$ by a $\mathrm{PCP}_{\mathbb{R}}$ class is presented in [10]: $\mathrm{NP}_{\mathbb{R}} = \mathrm{PCP}_{\mathbb{R}}(O(\log n), \mathit{polylog}(n))$. The proof faces similar difficulties as the one in [9], this time dealing with low-degree polynomials as coding objects instead of linear functions on certain real domains. The key point here is to embed and expand a so called low-degree test for polynomials defined on (almost) arbitrary real domains given in [8] to construct a suitable verifier for an $\mathrm{NP}_{\mathbb{R}}$-complete problem.

Whereas the design of a long transparent proof goes into both existing proofs for the PCP theorem in the Turing model, the construction of real low-degree tests structurally follows the original proof by Arora and co-authors. It is, however, unclear to the authors of this paper at the time being whether one can obtain the full real PCP theorem by continuing the proof along these lines. We comment on that at the end of the paper. Thus, it is natural to ask whether Dinur's proof can be carried over to the real number model.

Dinur's proof is based on constructing a certain reduction between 3-SAT formulas in which for unsatisfiable formulas a so called gap amplification is obtained. The existence of such a reduction is known to imply the PCP theorem. Nevertheless, at a first glance it is not clear whether similar ideas could be used for an appropriate $\mathrm{NP}_{\mathbb{R}}$-complete problem over the reals. The reason is that Dinur's proof heavily works with constraint systems that are to be solved over different finite alphabets as domains. The proof constructs several transformations between such finite alphabet constraint problems. It seems unclear whether the same can be done over uncountable structures. However, it turns out that this part depends more on the combinatorial structure of the constraints involved than on the question over which domains they are to be solved. To see this, a view of real polynomial systems – the main objects involved in real number computations – is taken that seems a bit uncommon in algorithmic semi-algebraic geometry. The latter requires not only to consider the semi-algebraic solution set of such a system (as it is usually done in algebraic geometry), but to put more focus on a suitable grouping of the polynomials involved in the system. Then, it is more important to argue about common semi-algebraic solutions of some of these groups than of the entire system. It turns out that this can be accomplished as well over $\mathbb{R}$. Consequently, at the moment Dinur's proof seems more appropriate for adaptation to real computational models than the classical one. The arguments given hold as well for the complex number BSS model and its corresponding $\mathrm{PCP}_{\mathbb{C}}$ classes and $\mathrm{NP}_{\mathbb{C}}$. Since all required changes are minor below we only add short remarks on the complex model where appropriate.

The paper is organized as follows. Section 2 introduces the main notions like real number verifiers and $\mathrm{PCP}_{\mathbb{R}}$ classes as well as the central $\mathrm{NP}_{\mathbb{R}}$-complete problem to be studied. It is a particular form of deciding solvability of polynomial systems; the problem is defined in such a way that the existence of a reduction amplifying the unsatisfiability gap will imply the $\mathrm{PCP}_{\mathbb{R}}$ theorem. The construction of this reduction is given in Section 3. A discussion about future questions ends the paper. All lacking proofs are postponed to the full version.

A word concerning the style of writing: We decided to extensively explain the flow of ideas at the cost of having to omit most of the proofs due to lacking space. This hopefully clarifies what is needed for a transformation of Dinur's proof to the reals and why this indeed is possible. A description of Dinur's proof is given in the very recommendable textbook [1].

2 Basic notions

We assume the reader to be familiar with real and complex number complexity theory, see [5]. Very briefly, a BSS machine is a uniform Random Access Machine that computes with real numbers as basic entities. An input $x \in \mathbb{R}^n$ is given the algebraic size $size_{\mathbb{R}}(x) := n$, and each operation $\{+, -, *, :, \geq 0?\}$ among real numbers can be performed with (algebraic) costs 1. The complexity class $\mathrm{NP}_{\mathbb{R}}$ consists of all decision problems L for which there exists a polynomial time verification procedure that satisfies the following requirements. Given $x \in L$ there is a proof y of polynomial size in the (algebraic) size of x such that the procedure accepts (x, y). And for every $x \notin L$ the procedure rejects all tuples (x, y), no matter what y looks like. For complex computations inputs stem from some $\mathbb{C}^n$ and only equality tests are allowed. The Quadratic Polynomial Systems problem introduced below is a typical example of a problem in $\mathrm{NP}_{\mathbb{R}}$ or in $\mathrm{NP}_{\mathbb{C}}$, respectively, depending on where coefficients lie and where solutions are searched.

Usually, the natural verification procedures for $\mathrm{NP}_{\mathbb{R}}$ problems have to inspect all components of y before the decision is made. The question one studies in relation with PCPs is whether this has to be the case. It is formalized using special randomized algorithms.

▶ **Definition 1** (Verifiers). Let $r, q : \mathbb{N} \mapsto \mathbb{N}$ be two functions. An $(r(n), q(n))$-restricted verifier V in the BSS model is a particular randomized real number algorithm working in three phases. For an input $x \in \mathbb{R}^* := \bigcup_{i \geq 1} \mathbb{R}^i$ of algebraic size n and another vector $y \in \mathbb{R}^*$ representing a potential membership proof of x in a certain set $L \subseteq \mathbb{R}^*$, the verifier in a first phase produces non-adaptively a sequence ρ of $r(n)$ many random bits (under the uniform distribution on $\{0,1\}^{r(n)}$). Given x and this sequence ρ of $r(n)$ many random bits, in the next phase, V computes deterministically the indices of $q(n)$ many components of y. Finally, in the decision phase V uses the input x, the random string ρ and the values of the chosen components of y in order to perform a deterministic polynomial time algorithm in the BSS model. At the end of this algorithm V either accepts or rejects the triple (x, y, ρ). For an input x, a guess y and a sequence of random bits ρ we denote by $V(x, y, \rho) \in \{0, 1\}$ the result of V.

Though being a real number algorithm the verifier generates discrete random bits in phase 1. The use of these bits is for addressing registers of the machine in which the basic units of a proof y, i.e., real numbers are stored. Therefore it is appropriate to work with this discrete kind of randomness. We can define the real language accepted by a verifier together with complexity classes $\mathrm{PCP}_{\mathbb{R}}(r(n), q(n))$ as follows.

▶ **Definition 2** ($\mathrm{PCP}_{\mathbb{R}}$-classes). Let $r, q : \mathbb{N} \mapsto \mathbb{N}$; a real number decision problem $L \subseteq \mathbb{R}^*$ is in class $\mathrm{PCP}_{\mathbb{R}}(r(n), q(n))$ iff there exists an $(r(n), q(n))$-restricted verifier V such that conditions a) and b) below hold:

a) For all $x \in L$ there exists a $y \in \mathbb{R}^*$ such that for all randomly generated strings $\rho \in \{0,1\}^{r(size_{\mathbb{R}}(x))}$ the verifier accepts.
b) For any $x \notin L$ and for all $y \in \mathbb{R}^*$ the verifier rejects with probability at least $\frac{3}{4}$.

In both cases the probability is chosen uniformly over all random strings ρ.

We next introduce the central decision problem to consider in this paper. It deals with polynomial systems and is a variant of the Hilbert Nullstellensatz problem. However, the viewpoint under which we structure such systems is a bit unusual, so the definition at a first glance might look confusing. The reason we take this unusual point of view is that we want this decision problem to resemble the CSP problem which plays a main role in Dinur's proof. A clarifying example follows after the definition.

▶ **Definition 3.** a) Let m, k, q, s be integers. An instance of the quadratic polynomial systems decision problem $QPS(m, k, q, s)$ is defined as follows. There are m constraints, each of which is a group of at most k real polynomials. The polynomials depend on arrays of real number variables. These arrays are disjoint, different arrays do not contain the same variable. Each array has s components and thus represents a vector of variables ranging over $\mathbb{R}^s$. The single polynomials then depend on the variables that occur as components in one of the arrays. All polynomials in one constraint have degree at most 2 and depend on at most q many arrays, i.e., on at most qs many real variables altogether.

b) A constraint is satisfied by a point $x \in \mathbb{R}^{qs}$ if x is a common zero of all polynomials in the constraint. The $QPS(m, k, q, s)$-instance is solvable if there is a common real solution for all its constraints.

c) If above coefficients belong to $\mathbb{C}$ and solutions are searched in $\mathbb{C}^{qs}$ we obtain the complex $QPS(m, k, q, s)$ problem.

▶ Remark. a) Below, we usually consider the parameters k, q, s as constants, whereas m is depending on the actual instance. In that sense it would be more correct to talk about $QPS(k, q, s)$-instances; however, at many places we argue about the changes in the number of constraints, so the above seems notationally easier.

b) Over the reals parameter k in principle is not necessary. Here, using basic arguments from [6] we could always choose $k = 1$ and the corresponding constraint to be given by a single polynomial equation $f(x) = 0$ of degree at most 4 with f being non-negative on a corresponding $\mathbb{R}^n$. However, we notationally prefer to take care of k. Firstly in order to deal as well with the complex BSS model in which the above simplification does not work. Secondly, in many cases below we believe specifying k increases readability to state explicitly which polynomial equations enter into which constraints and might cause its unsatisfiability.

c) Since mqs is an upper bound for the number of variables an instance depends on at many places below we do not specify this number for concrete instances more precisely.

▶ **Example 4.** It is well known that deciding existence of a real zero of a polynomial system $\mathcal{P} := \{p_1, \ldots, p_m\}$, where each polynomial p_i depends on at most three variables and has degree at most two is $\text{NP}_{\mathbb{R}}$-complete, see [5]. We give two formulations of this question in the new framework by specifying different choices of parameters. The examples show $\text{NP}_{\mathbb{R}}$-completeness of the QPS problem for the given choices of k, q, and s.

i) The system $\mathcal{P}$ can be formulated as an instance in $\text{QPS}(m, 1, 3, 1)$. Each constraint consists of a single polynomial p_i, thus $k = 1$; the variable arrays all have dimension $s = 1$ and each polynomial depends on at most 3 such arrays.

ii) If we take arrays of dimension $s = 3$, then in a first reduction step we consider all such arrays as depending on different variables. Then additional constraints have to be added to guarantee consistency between the same variables occurring in arrays of different polynomials. Such a constraint has a single polynomial depending on two arrays. It claims equality between variable components that have to be the same in the originally given system. We thus obtain a $\text{QPS}(\tilde{m}, 1, 2, 3)$ instance where $\tilde{m}$ is a bound for m plus the number of different pairs of variable arrays.

Note that the above instances are equivalent to $\mathcal{P}$ as far as solvability is concerned. Below it will be very important to argue about the number of constraints not satisfied if a system is unsolvable. For these arguments it is crucial to group the single polynomials into constraints.

For what follows QPS-instances with parameter $q = 2$ are most important. This is due to the possibility of canonically assigning a constraint graph to such an instance that connects arrays as vertices. It is then more important to argue about this graph than about the semi-algebraic solution set of (subsets of) the polynomials involved in the system (though the latter of course cannot not be completely disregarded).

The starting point of Dinur's proof is a simple observation which implies the PCP theorem if the existence of a very particular reduction can be established. We next recall this type of reduction and state the corresponding easy lemma for QPS and the $\text{PCP}_{\mathbb{R}}$ theorem.

▶ **Definition 5.** a) For a QPS(m,k,q,s)-instance ϕ denote by $UNSAT(\phi)$ the smallest fraction of constraints in ϕ that cannot be satisfied in common. Thus, if ϕ is satisfiable $UNSAT(\phi)=0$ and otherwise $UNSAT(\phi)\geq \frac{1}{m}$.

b) A gap reduction for QPS-instances is a polynomial time BSS algorithm that works as follows. There is a fixed $\epsilon>0$ such that given a QPS(m,k,q,s) instance ϕ the algorithm computes an instance ψ in $QPS(m',k,q,s)$ satisfying the following:

i) if ϕ is satisfiable so is ψ;

ii) if ϕ is not satisfiable, then at most a fraction of $(1-\epsilon)$ many of the constraints of ψ are satisfiable in common, i.e., $UNSAT(\psi)\geq\epsilon$.

Clearly, m' is polynomially bounded in m.

The following lemma is easy to prove. Note, however, that it seems unclear whether its converse holds as well as it does in the Turing model.

▶ **Lemma 6.** *Suppose there exists a gap reduction for QPS with a fixed $\epsilon>0$. Then the* $\text{PCP}_{\mathbb{R}}$ *theorem follows, i.e., $NP_{\mathbb{R}}=\text{PCP}_{\mathbb{R}}(O(\log n),O(1))$.*

3 The main proof

We shall now turn to the main part of the proof, the construction of a gap reduction for the $\text{NP}_{\mathbb{R}}$-complete problem QPS$(m,C,Q,1)$, see Example 4. The parameters $C\geq 1$ and $Q\geq 3$ are constants that will be specified later. The structure of the proof is similar to that of the classical PCP theorem by Dinur. Its basic idea is as follows. Starting from a QPS$(m,C,Q,1)$-instance ϕ which is unsatisfiable an amplification step is performed. It constructs in polynomial time another QPS-instance ψ out of ϕ that has an increased unsatisfiability ratio. More precisely, if $UNSAT(\phi)=\epsilon$, then $UNSAT(\psi)\geq c\cdot\epsilon$ for a suitable constant $c>1$ and ϵ small enough. Now in principle starting with $UNSAT(\phi)\geq\frac{1}{m}$ and repeating this amplification $\log m$ times the gap is increased from at least one unsatisfied constraint in ϕ to a constant fraction of unsatisfied constraints in the finally resulting instance. However, the amplification step increases the dimension of the variable arrays too much. Thus, before repeating amplification a dimension reduction step is performed that first reduces the parameter s again. Note that dimension reduction in Dinur's proof is called alphabet reduction. Over the reals, however, parameter s refers to the dimension of variable arrays, whereas the underlying alphabet is always infinite. We thus consider the changed notion to be more appropriate here.

Amplification is performed on instances having particularly structured constraint graphs. These are related to so called expanders. Therefore, in a preprocessing step it has to be shown why it is possible to start with such particular instances.

The section is organized as follows. The first subsection collects the results necessary to do the preprocessing. Since it closely follows the classical preprocessing step omit the proofs. Next, the amplification step is described. Though basically Dinur's idea works over the reals as well, a lot of small details and calculations have to be changed. We thus include the full proof, always pointing out where differences to the discrete setting occur. The dimension reduction step is given in subsection 3.3. It relies on the long transparent proofs for $\mathrm{NP}_{\mathbb{R}}$, see [9, 4].

3.1 Preprocessing

In order to apply below the main steps necessary to establish the existence of a gap reduction for QPS, namely amplification and dimension reduction, we first have to preprocess a given instance. The goal of this preprocessing step is to obtain a QPS instance that has a constraint set which in a certain sense is highly structured. Such instances are called *nice* below. Niceness is modelled using expanders, a well known concept from graph theory. Throughout this section (except for the start of the first preprocessing step) we consider QPS instances whose constraints depend on two variable arrays, i.e., for which parameter $q = 2$. This allows to canonically attach a constraint graph to the instance.

▶ **Definition 7.** (Constraint graph) For a QPS$(m, k, 2, s)$ instance ϕ we define its constraint graph as the graph which uses the variable arrays of ϕ as vertices and where two of them are connected iff they occur in a common constraint of ϕ.

Before the amplification step is performed it is necessary to guarantee that this constraint graph has a particular structure.

We shall first give the necessary graph theoretical definitions and then show why without loss of generality we can start from a nice QPS instance. Expanders are regular graphs that in a certain sense exhibit properties of random regular graphs of the same degree of regularity. In this section we define algebraic expanders.

For the definition of algebraic expansion we need the random walk matrix of a graph G.

▶ **Definition 8.** (Random walk matrix) Let $G = (V, E)$ be a graph. The random walk matrix $A(G)$ of G is defined to be the $|V| \times |V|$ matrix in which the entry A_{ij} equals the probability that in a random walk on G vertex j is chosen after vertex i. Here each edge incident with node i and not being a loop is chosen with the same probability, whereas loops are chosen with twice this probability; see Remark 3.1 below.

▶ **Definition 9.** (Algebraic expansion) Let $n, d \in \mathbb{N}, \lambda < 1$, and $G = (V, E)$ a d-regular graph with $|V| = n$. Let $\lambda(G)$ be the second largest eigenvalue in absolute value of $A(G)$. The graph G is called a d-regular expander with expansion parameter λ if $\lambda(G) \leq \lambda$.

The QPS-instances that are important below are required to have a constraint graph which is an expander having additional properties. The corresponding definition is

▶ **Definition 10.** A QPS(m, k, q, s)-instance ϕ is called nice if the following conditions hold:

i) the number q of arrays on which each constraint depends is 2;
ii) the constraint graph G of ϕ is d-regular for some absolute constant $d \in \mathbb{N}$ which is in particular independent of the parameter s. We allow G to have loops (resulting from constraints that only depend on a single array); for each vertex of the graph one third of the edges incident to that vertex are loops.
iii) The constraint graph is an expander with algebraic expansion parameter $\lambda(G) \leq 0.9$.

▶ Remark. Our main purpose when considering random walks on a constraint graph is to guarantee that all edges occur with the same probability as, say, first edge of such a walk if a vertex is chosen at random. To achieve this property we consider loops as contributing one edge which, however, in a random walk is chosen with twice the probability of edges that are no loops. There is still one constraint attached to a loop, and the loop contributes 2 to the degree of its vertex. Consequently, for the random walk matrix a loop contributes $\frac{2}{d}$ to the corresponding diagonal entry.

The following theorem summarizes preprocessing.

▶ **Theorem 11.** *There exist a constant $d \in \mathbb{N}$ and a polytime computable function from QPS instances to QPS instances which maps a $QPS(m, k, q, s)$ instance ϕ to a nice instance ψ in $QPS(3qd^2m, k + qs, 2, qs)$ such that*
- *if ϕ is satisfiable, then ψ is satisfiable;*
- *if ϕ is not satisfiable, then $UNSAT(\psi) \geq UNSAT(\phi)/(240qd^2)$.*

It is important for what follows that above both the number of constraints is increased and the unsatisfiability factor is decreased by a constant factor only.

3.2 Amplification

Given a nice QPS-instance the all-over purpose now is to perform a logarithmic number of reduction rounds to increase the unsatisfiability gap. The first step in a round is an amplification step which increases the ratio of unsatisfied constraints by a constant factor > 1. After the amplification step a dimension reduction step reduces again the dimension of the variable arrays which has been increased during amplification. In this subsection amplification is explained.

Suppose a nice QPS$(m, k, 2, s)$-instance ψ is given. Let d denote the corresponding regularity parameter and let n be the number of variable arrays. Our final goal is to construct an instance which either is satisfiable if ψ was or in which for any assignment a constant fraction $\epsilon_{final} > 0$ (to be specified) of the constraints will not be satisfied. We will concentrate our arguments on how amplification works for unsatisfiable instances ψ which have a gap that is too small, i.e., smaller than ϵ_{final}. If $UNSAT(\psi)$ is already large enough it will stay above this ϵ_{final} after the reduction, see below.

So we assume that the input instance has a gap which is smaller than some constant which we will specify later. Our goal is to construct in polynomial time an instance ψ^t in some QPS$(m(t), k(t), 2, s(t))$ such that $UNSAT(\psi^t) \geq c \cdot UNSAT(\psi)$ for a suitable constant $c > 1$. Here, $t \in \mathbb{N}$ is a suitably chosen constant that results from the construction. Before going into details here is a brief outline of how amplification was done by Dinur, adapted to the real case. The new instance ψ^t has the same number n of variable arrays as ψ. Whereas the latter range over $\mathbb{R}^s$ the former range over an enlarged $\mathbb{R}^{s(t)}$, where $s(t) := d^{t+\sqrt{t}+1} \cdot s$. The constraints in the new instance are built on base of paths with $2t$ many edges in the old constraint graph G_ψ. Each such path results in an own constraint of ψ^t. Before defining such a constraint it is necessary to describe the role of the variable arrays in ψ^t. For a vertex $i \in \{1, \dots, n\}$ the corresponding variable array $y_i \in \mathbb{R}^{s(t)}$ consists of $d^{t+\sqrt{t}+1}$ many blocks of dimension s each (therefore $s(t) = d^{t+\sqrt{t}+1} \cdot s$). Each block is thought of as an old variable array of ψ that corresponds to a vertex in G_ψ reachable within $t + \sqrt{t}$ steps from i. Since G_ψ is d-regular there are at most $d^{t+\sqrt{t}+1}$ many such vertices. In that sense we can say that an assignment for all new variable arrays $y_i \in \mathbb{R}^{s(t)}, 1 \leq i \leq n$, claims a value for all old arrays x_j for vertices j in a $(t + \sqrt{t})$-neighborhood of vertex i. Of course, different y_i might claim different values on the same old array.

Now let p be a path of length $2t$ in G_ψ from vertex i_1 to i_{2t+1}, say $p := (i_1, i_2, \dots, i_{2t+1})$. For each such path a constraint is added to ψ^t as follows: The constraint depends on the two arrays y_{i_1} and $y_{i_{2t+1}}$ and expresses two requirements:

1. Consistency-between-new-variables requirement: Since y_{i_1} claims values for $x_{i_1}, x_{i_2}, \ldots, x_{i_{t+\sqrt{t}+1}}$ and $y_{i_{2t+1}}$ claims values for $x_{i_{2t+1}}, x_{i_{2t}}, \ldots, x_{i_{t-\sqrt{t}+1}}$, the old variables x_{i_j} for $j \in \{t-\sqrt{t}+1, \ldots, t+\sqrt{t}+1\}$ are covered by both new arrays. We thus include for all those j linear equations expressing that that y_{i_1} and $y_{i_{2t+1}}$ claim the same values on all their components. This contributes $(2\sqrt{t}+1)\cdot s$ linear equations to the constraint. [1]
2. Consistency-with-old-constraints requirement: As explained above edges (i_j, i_{j+1}) of G_ψ are covered by both variable arrays $y_{i_1}, y_{i_{2t+1}}$ for $j \in \{t-\sqrt{t}+1, \ldots, t+\sqrt{t}\}$; to each of these $2\sqrt{t}$ many edges there corresponds an old constraint of ψ. The new constraint requires as well that those old constraints are satisfied by the values assigned to the old variable arrays through y_{i_1} (or equivalently, because of item 1., through $y_{i_{2t+1}}$).

Requirement 2. for each j is the same as in the given instance just changing variables. So each constraint in ψ^t is made of $\le k(t) := 2\sqrt{t}k + (2\sqrt{t}+1)\cdot s$ many polynomial equations. Since G_ψ is regular there are at most $m(t) := n \cdot d^{2t}$ many paths of length $2t$, so this bounds as well the number of constraints in ψ^t.

It is easy to see that a satisfying assignment for ψ extends to one for ψ^t. Just propagate the assignment to the y_i's according to the first consistency requirement above. The hard part to see is why an unsatisfiability ratio of a given (unsatisfiable) ψ is increased by the construction. Towards this aim we relate any assignment for the new variable arrays to a so-called plurality assignment for the old arrays. Assuming this plurality assignment (like any other) to violate a fraction of $UNSAT(\psi)$ many constraints in ψ it is then shown that the given assignment y for ψ^t violates a ratio of $\ge c \cdot UNSAT(\psi)$ many constraints of ψ^t. This reasoning closely follows Dinur's one. However, due to the fact that assignments stem from the uncountable set $\mathbb{R}^{s(t)}$ instead of a finite set some arguments have to be adjusted. One necessary change in order to perform this adjustment is the inclusion of the consistency-between-new-variables requirement above.

Now towards the details. Given an assignment $y = (y_1, \ldots, y_n) \in \mathbb{R}^{n\cdot s(t)}$ for the n variable arrays $y_i \in \mathbb{R}^{s(t)}$ of ψ^t, we first define the plurality assignment inferred from it to the old arrays $x_j \in \mathbb{R}^s$: Let $t \in \mathbb{N}$ be fixed. Consider a vertex v of G_ψ together with a random walk in G_ψ of length t starting in v. Remember the way loops are treated in such a walk, see Remark 3.1. With a certain probability the walk reaches a vertex u which obviously belongs to the t-neighborhood of v. Then for this u there is a new variable array y_u. Its assignment in particular claims a value for the old array x_v. The plurality assignment x_v^{pa} for x_v then is defined to be the assignment resulting with highest probability from y according to the above random walk process. Ties can be broken arbitrarily.

One technical difference to the discrete setting has to be pointed out here. Over the reals there is no guarantee how often the plurality assignment occurs at least. It is only clear that it occurs at least once. Contrary, if the variables take values over a finite alphabet there is a constant lower bound on the probability with which the plurality assignment occurs; this bound depends only on the alphabet size but not on t. This difference requires below a modification of the discrete arguments.

Suppose then that ψ is unsatisfiable with $UNSAT(\psi) = \epsilon > 0$. Let an arbitrary assignment y for ψ^t and the related plurality assignment x^{pa} for ψ be fixed. Every assignment x^{pa} violates $\ge m\cdot\epsilon$ constraints (where m denotes the number of constraints in ψ). Our goal is to show that y violates at least a fraction of $\epsilon(t) = c\cdot\epsilon$ constraints in ψ^t for a large enough

[1] These requirements are not included in the classical construction due to the underlying finite alphabets. It will become obvious below why it is needed over the reals.

value $c > 1$. This is achieved by analyzing two cases. Consider an edge $e = (i_j, i_{j+1})$ in G_ψ such that the plurality assignment $(x^{pa}_{i_j}, x^{pa}_{i_{j+1}})$ violates the corresponding constraint in ψ.

a) If the values $x^{pa}_{i_j}, x^{pa}_{i_{j+1}}$ have been claimed by relatively many (to be specified) endpoints of the corresponding random walks it is shown that many of the endpoints of $2t$-step paths of G_ψ in which e occurs in the middle segment claim the plurality values for $x^{pa}_{i_j}, x^{pa}_{i_{j+1}}$, i.e., y violates many constraints in ψ^t. This part is analyzed similarly as in the finite alphabet situation.

b) If at least one of the values $x^{pa}_{i_j}$ or $x^{pa}_{i_{j+1}}$ has been claimed by few endpoints only (but still represents the majority of the occurring values) we show that y violates the consistency-between-new-variables requirements in a lot of constraints. This case has to be handled because of the reals as underlying structure.

We now calculate a lower bound for the expectation of a random variable V defined as follows: V counts the number of edges e as mentioned above in a random path that cause the corresponding constraint to be unsatisfied. We also calculate an upper bound for the square $E[V^2]$ of the number of such edges in a random path. Since for any nonnegative random variable V taking integral values by an application of Chebychev's inequality it is $Pr[V > 0] \geq E[V]^2/E[V^2]$ this will then give us a lower bound on the fraction of paths for which the corresponding constraint in ψ^t is not satisfied.

Assume we have an edge $e = (i_j, i_{j+1})$ such that the corresponding constraint in ψ is violated by the plurality assignment. We start by considering case a) and assume that the plurality values of $x_{i_j}, x_{i_{j+1}}$ are claimed relatively often. At first sight this information seems pretty useless because if we look at the set of paths in which e occurs in the middle segment then it is obvious that for almost all of them the distance (along the path) from x_{i_j} and $x_{i_{j+1}}$ to the respective endpoints is not t. The plurality assignment was defined using random walks of length t, so it does not say anything directly about walks which have a different length. To solve this problem we need the many loops guaranteed to exist by the niceness condition. Their existence implies that a random walk of t steps statistically is not too different from a random walk which has a few steps more or less. If the random walk starts in u, the probability that we end in v only changes very slightly if we make our walk a few steps longer or shorter. The following lemma makes this precise.

► **Lemma 12.** *Let $t \in \mathbb{N}, \delta \leq 1/160$ and $j \in \{t - \delta\sqrt{t}, \ldots, t + \delta\sqrt{t}\}$. If the plurality assignments for x_{i_j} and $x_{i_{j+1}}$ both occur with probability at least $\frac{5}{8}$, then the following holds. For a fraction of at least $\frac{1}{4}$ of the paths of length $2t$ that have $e = (i_j, i_{j+1})$ as j-th edge the values that the starting point y_{i_1} and the endpoint $y_{i_{2t+1}}$ of the path claim for x_{i_j} and $x_{i_{j+1}}$, respectively, agree with the plurality assignments for those arrays.*

In order to obtain the desired lower bound for $E[V]$ next we have to deal with the case where the plurality assignment is claimed with a small probability only. The following shows that in this case the corresponding edge e leads in a large fraction of the paths to a violation of the corresponding constraint via case b).

► **Lemma 13.** *Let $t \in \mathbb{N}, \delta \leq 1/160$ and $j \in \{t-\delta\sqrt{t}, \ldots, t+\delta\sqrt{t}\}$. If the plurality assignment for x_{i_j} occurs with probability less than $\frac{5}{8}$ the following holds. For a fraction of at least $\frac{1}{4}$ of the paths of length $2t$ that have e as j-th edge the values that the starting point y_{i_1} and the endpoint $y_{i_{2t+1}}$ of the path claim for x_{i_j} disagree.*

The corresponding statement is true for $x_{i_{j+1}}$.

Let F denote the set of edges in the instance ψ such that the corresponding constraint is violated by the plurality assignment. Recall that our goal is to prove a lower bound for

$\Pr[V > 0]$, where V is the random variable which counts the number of edges e in a random $2t$-step path that satisfy the following: e belongs to F, it is the j-th edge in the path for a $j \in \{t - \delta\sqrt{t}, \ldots, t + \delta\sqrt{t}\}$, where $\delta = 1/160$ and causes the corresponding constraint in ψ^t to be unsatisfied by assignment y. We are now able to extract such a lower bound.

▶ **Lemma 14.** *Let $\epsilon \leq \frac{1}{d\cdot\sqrt{t}}$, let ψ be an instance with $UNSAT(\psi) = \epsilon$ and ψ^t be constructed as above. For the random variable V as defined above it is $Pr[V > 0] \geq c \cdot \epsilon$ with $c = \frac{\sqrt{t}}{3520d}$.*

The above lemmas result in

▶ **Theorem 15.** *There exists an algorithm which works in polynomial time that maps a nice $QPS(m, k, 2, s)$ instance ψ to a $QPS(d^{2t}m, 2\sqrt{t}k + (2\sqrt{t} + 1)s, 2, d^{t+\sqrt{t}+1}s)$-instance ψ^t and has the following properties:*

- *If ψ is satisfiable, then ψ^t is satisfiable.*
- *If ψ is not satisfiable and $UNSAT(\psi) < \frac{1}{d\sqrt{t}}$, then $UNSAT(\psi^t) \geq \frac{\sqrt{t}}{3520d} \cdot UNSAT(\psi)$.*

Note that the construction works precisely the same in the complex BSS model.

3.3 Dimension reduction

The amplification step increases an unsatisfiability ratio $\epsilon < \frac{1}{d\sqrt{t}}$ by a factor $c \geq \frac{\sqrt{t}}{3520d}$. Thus starting with an unsatisfiable instance ϕ that has m constraints it would be sufficient to repeat the amplification step $\Omega(\log m)$ number of times in order to end with an instance that has a constant unsatisfiability gap $\geq \frac{1}{d\sqrt{t}}$. However, doing it naively the dimension of arrays in the resulting instance would no longer remain constant. This would imply as well the query complexity to be not any longer constant, compare Lemma 6. Therefore, the dimension has to be reduced again each time an amplification step was applied. This has to be done in such a way that we do not lose too much of the gap-increase the amplification step gave.

To get around this problem one should first alter the instance in such a way that every constraint depends on at most Q variables only. Here, Q is an absolute constant independent of the array size. In Dinur's proof this is done using so-called transparent long proofs for NP. The corresponding construction is called alphabet-reduction there because the different amplification steps deal with satisfiability problems over finite alphabets of different cardinalities. With respect to the real number model it is more appropriate to consider it as a dimension reduction. All instances that occur during amplification are to be solved over the reals, i.e., there are no different 'alphabets' to deal with.

Transparent long proofs have been used already in the first proof of the classical PCP theorem, see [2], where they are crucial for applying a technique called verifier-composition. In the real and complex number model [9, 4] show the existence of transparent long proofs for all problems in $\mathrm{NP}_{\mathbb{R}}$ and $\mathrm{NP}_{\mathbb{C}}$, respectively. More precisely, a verifier for an $\mathrm{NP}_{\mathbb{R}}$-complete problem is designed which uses a superpolynomial number of random bits and inspects a constant number Q of proof components. As already said in the introduction proving this requires considerable additional work in comparison to the Turing setting. The main task is to design an algorithm for testing linearity of certain real number functions on unstructured finite subsets of some $\mathbb{R}^n$. Unstructured here means in particular that these domains are not closed under addition and scalar multiplication. This causes certain invariance properties of the uniform distribution to be violated. The latter, however, is crucially used in the finite alphabet framework to show existence of long transparent proofs.

Long transparent proofs provide a way to replace each constraint in ψ^t by many constraints all depending on at most Q real variables (i.e., arrays of dimension 1); Q denotes the constant query complexity of the long transparent proof and thus is independent of the instance. If the old constraint is not satisfiable, then at least half of the new ones will not be satisfied. Thus one gets a reduction from constraints considered as QPS-instances to QPS-instances which blows up the gap to a constant. As seeming disadvantage the size of the long proof becomes superpolynomial in the size of the instance. But the verification using long proofs will be applied to instances of constant size only, namely single constraints in ψ^t. Thus the length of the transparent long proofs in fact does not matter at all. The much more important aspect is their structure which will not be explained here due to lack of space. The main result of this subsection, of which we also omit the proof, is

▶ **Theorem 16.** *There exists a reduction which works in polynomial time and maps a* $QPS(m(t), k(t), 2, s(t))$*-instance* ψ^t *to a* $QPS(\widehat{m}(t), C, Q, 1)$*-instance* $\widehat{\psi}^t$*, where* C, Q *are constants,* $\widehat{m}(t)$ *is linear in* $m(t)$ *(the multiplication factor being double exponential in* $s(t)$*) and the following holds:*

- *If* ψ^t *is satisfiable, then so is* $\widehat{\psi}^t$ *and*
- *if* ψ^t *is unsatisfiable, then* $UNSAT(\widehat{\psi}^t) \geq UNSAT(\psi^t)/(160(d+1)^2)$*.*

3.4 Putting all together

Let $Q \geq 3$ be the $O(1)$-constant from long transparent proofs for QPS$(m, 1, 3, 1)$ and let $C \geq 1$ be the number of polynomials in a proof check ,i.e., the number of polynomials in the QPS-instance which the verifier computes out of the input instance and the random bits.

Given an instance ϕ of QPS$(m, C, Q, 1)$, applying preprocessing yields a nice instance of QPS$(3Qd^2m, C+Q, 2, Q)$ (Theorem 11), then applying amplification yields an instance of QPS$(3d^{2t+2}Qm, 2\sqrt{t}(C+Q)+(\sqrt{t}+1)Q, 2, d^{t+\sqrt{t}+1}Q)$, and finally applying dimension reduction yields an instance $\widehat{\psi^t}$ of QPS$(m', C, Q, 1)$ with the following properties.

- m' is linear in m, the multiplication factor is double exponential in Qd^t;
- if ϕ is satisfiable so is $\widehat{\psi}^t$;
- if ϕ is unsatisfiable and UNSAT$(\phi) \leq \frac{1}{d\sqrt{t}}$, then UNSAT$(\widehat{\psi}^t) \geq$ UNSAT$(\phi) \cdot \frac{\sqrt{t}}{10^{10}Qd^4}$.

Assume ϕ is not satisfiable. We now choose $t = (2 \cdot 10^{10}Qd^4)^2$ so that a gap which is smaller than $\frac{1}{d\sqrt{t}}$ will be amplified with a factor of at least 2 by this reduction. Thus from an instance of QPS$(m, 1, 3, 1)$, one builds in $\log m$ steps an instance with a gap of at least $\frac{1}{d\sqrt{t}}$.

Finally, since in every step the number of constraints increases linearly, after less than $\log m$ steps the number of constraints in the final instance is polynomial in m. Using Lemma 6 we thus arrive at the Main Theorem:

▶ **Theorem 17.** *It holds* $NP_{\mathbb{R}} = \mathrm{PCP}_{\mathbb{R}}(O(\log n), O(1))$*. The same is true in the BSS model over* $\mathbb{C}$*.*

4 Open questions

First, we consider it interesting to figure out whether the theorem as well can be proved along the lines of the first proof of the classical PCP theorem in [2, 3]. Its main ingredients are certain property testing procedures as well as a technique called verifier composition. Whereas the latter is very similar to the ideas behind the dimension reduction step above, the different property testing algorithms necessary probably result in more severe difficulties in the real number setting. Testing linear functions can be done similarly, as has been discussed above in relation with transparent long proofs for NP$_{\mathbb{R}}$. In [10] a first characterization of NP$_{\mathbb{R}}$ via PCP$_{\mathbb{R}}(O(\log n), O(polylog(n)))$ was given by designing a real algorithm for testing low-degree polynomials. This algorithm is based on testing the maximal degree of a polynomial with respect to its variables. In order to apply the verifier composition step the classical proof of the PCP theorem puts such a low-degree test into a better structure by designing a total-degree test. It is unclear whether such a test could be designed as well over the reals without loosing other important properties such as the length of a proof.

Secondly, approximation problems have not yet been studied in real number complexity to a comparable extent as in classical complexity theory. An important implication of the classical PCP theory was the non-approximability of the MAX-3-SAT optimization problem via so called polynomial time approximation schemes, see [1]. A natural problem to study in this respect is to maximize the number of commonly solvable polynomial equations in a real number polynomial system. A direct implication of the existence of a gap-reduction shows that this maximum is not efficiently approximable. More precisely, given a system and an arbitrary $\epsilon > 0$, unless $P_{\mathbb{R}} = NP_{\mathbb{R}}$ there is no real number algorithm running in polynomial time in the system's size which approximates the maximal number of commonly solvable equations within a relative factor $1 + \epsilon$. A promising direction for future research seems to get more (non)-approximability results of that type for natural real number optimization problems as consequence of the $PCP_{\mathbb{R}}$ theorem.

Thirdly, in view of the BSS model having been introduced for many further structures like rings, vector spaces or groups one might ask whether the PCP theorem as well holds in such structures.

Finally, there are of course many further questions that have been studied in the Turing model as consequence of the PCP theorem which also make sense in the BSS-model. One typical such is the problem to optimize the parameters in the $PCP_{\mathbb{R}}$ theorem.

References

1 S. Arora, B. Barak: Computational Complexity: A Modern Approach. Cambridge University Press, 2009.

2 S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy: Proof verification and hardness of approximation problems. Journal of the ACM 45 (3), 501–555, 1998. Preliminary version: Proc. 33rd Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, 14–23, 1992.

3 S. Arora, S. Safra: Probabilistic checking proofs: A new characterization of NP. Journal of the ACM 45 (1), 70–122, 1998. Preliminary version: Proc. of the 33rd Annual IEEE Symposium on the Foundations of Computer Science, 2–13, 1992.

4 M. Baartse, K. Meer: Topics in real and complex number complexity theory. Submitted to: Proc. of the Santaló Summer School ”Real Computation and Complexity”, UIMP, Santander, 2012.

5 L. Blum, F. Cucker, M. Shub, S. Smale: Complexity and Real Computation. Springer, 1998.

6 L. Blum, M. Shub, S. Smale: On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. Bull. AMS, vol. 21, 1–46, 1989.

7 I. Dinur: The PCP theorem by gap amplification. Journal of the ACM Vol. 54 (3), 2007.

8 K. Friedl, Z. Hátsági, A. Shen: Low-degree tests. Proc. SODA, 57–64, 1994.

9 K. Meer: Transparent long proofs: A first PCP theorem for $NP_{\mathbb{R}}$. Foundations of Computational Mathematics, Springer, Vol. 5, Nr. 3, 231–255, 2005.

10 K. Meer: Almost transparent short proofs for $NP_{\mathbb{R}}$. Extended abstract in: Proc. 18th International Symposium on Fundamentals of Computation Theory FCT 2011, Oslo, Lecture Notes in Computer Science 6914, 41–52, 2011.

Mutual Dimension*

Adam Case and Jack H. Lutz

Department of Computer Science, Iowa State University
Ames, IA 50011 USA

Abstract

We define the lower and upper *mutual dimensions* $mdim(x : y)$ and $Mdim(x : y)$ between any two points x and y in Euclidean space. Intuitively these are the lower and upper densities of the algorithmic information shared by x and y. We show that these quantities satisfy the main desiderata for a satisfactory measure of mutual algorithmic information. Our main theorem, the *data processing inequality* for mutual dimension, says that, if $f : \mathbb{R}^m \to \mathbb{R}^n$ is computable and Lipschitz, then the inequalities $mdim(f(x) : y) \leq mdim(x : y)$ and $Mdim(f(x) : y) \leq Mdim(x : y)$ hold for all $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^t$. We use this inequality and related inequalities that we prove in like fashion to establish conditions under which various classes of computable functions on Euclidean space preserve or otherwise transform mutual dimensions between points.

1998 ACM Subject Classification F. Theory of Computation

Keywords and phrases computable analysis, data processing inequality, effective fractal dimensions, Kolmogorov complexity, mutual information

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.116

1 Introduction

Recent interactions among computability theory, algorithmic information theory, and geometric measure theory have assigned a *dimension* $dim(x)$ and a *strong dimension* $Dim(x)$ to each individual point x in a Euclidean space $\mathbb{R}^n$. These dimensions, which are real numbers satisfying $0 \leq dim(x) \leq Dim(x) \leq n$, have been shown to be geometrically meaningful. For example, the *classical* Hausdorff dimension $dim_H(E)$ of any set $E \subseteq \mathbb{R}^n$ that is a union of Π^0_1 (computably closed) sets is now known [16, 10] to admit the pointwise characterization

$$dim_H(E) = \sup_{x \in E} dim(x).$$

More recent investigations of the dimensions of individual points in Euclidean space have shed light on connectivity [18, 22], self-similar fractals [17, 6], rectifiability of curves [9, 20, 8], and Brownian motion [11].

In their original formulations [16, 1], $dim(x)$ is $cdim(\{x\})$ and $Dim(x)$ is $cDim(\{x\})$, where $cdim$ and $cDim$ are constructive versions of classical Hausdorff and packing dimensions [7], respectively. Accordingly, $dim(x)$ and $Dim(x)$ are also called *constructive fractal dimensions*. It is often most convenient to think of these dimensions in terms of the Kolmogorov complexity characterization theorems

$$dim(x) = \liminf_{r \to \infty} \frac{K_r(x)}{r}, \; Dim(x) = \limsup_{r \to \infty} \frac{K_r(x)}{r}, \tag{1.1}$$

* This research was supported in part by National Science Foundation Grants 0652569, 1143830, and 1247051. Part of the second author's work was done during a sabbatical at Caltech and the Isaac Newton Institute for Mathematical Sciences at the University of Cambridge.

where $K_r(x)$, the Kolmogorov complexity of x at precision r, is defined later in this introduction [19, 1, 17]. These characterizations support the intuition that $dim(x)$ and $Dim(x)$ are the lower and upper *densities of algorithmic information* in the point x.

In this paper we move the pointwise theory of dimension forward in two ways. We formulate and investigate the *mutual dimensions* — intuitively, the lower and upper densities of shared algorithmic information — between two points in Euclidean space, and we investigate the *conservation* of dimensions and mutual dimensions by computable functions on Euclidean space. We expect this to contribute to both computable analysis — the theory of scientific computing [3] — and algorithmic information theory.

The analyses of many computational scenarios call for quantitative measures of the degree to which two objects are correlated. In classical (Shannon) information theory, the most useful such measure is the *mutual information* $I(X : Y)$ between two probability spaces X and Y [5]. In the algorithmic information theory of finite strings, the (*algorithmic*) *mutual information* $I(x : y)$ between two *individual* strings $x, y \in \{0,1\}^*$ plays an analogous role [15]. Under modest assumptions, if x and y are drawn from probability spaces X and Y of strings respectively, then the expected value of $I(x : y)$ is very close to $I(X : Y)$ [15]. In this sense algorithmic mutual information is a refinement of Shannon mutual information.

Our formulation of mutual dimensions in Euclidean space is based on the algorithmic mutual information $I(x : y)$, but we do not use the seemingly obvious approach of using the binary expansions of the real coordinates of points in Euclidean space. It has been known since Turing's famous correction [23] that binary notation is not a suitable representation for the arguments and values of computable functions on the reals. (See also [12, 24].) This is why the characterization theorems (1.1) use $K_r(x)$, the *Kolmogorov complexity* of a point $x \in \mathbb{R}^n$ at precision r, which is the minimum Kolmogorov complexity $K(q)$ — defined in a standard way [15] using a standard binary string representation of q — for all rational points $q \in \mathbb{Q}^n \cap B_{2^{-r}}(x)$, where $B_{2^{-r}}(x)$ is the open ball of radius 2^{-r} about x. For the same reason we base our development here on the *mutual information* $I_r(x : y)$ between points $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^n$ at *precision* r. This is the minimum value of the algorithmic mutual information $I(p : q)$ for all rational points $p \in \mathbb{Q}^m \cap B_{2^{-r}}(x)$ and $q \in \mathbb{Q}^n \cap B_{2^{-r}}(y)$. Intuitively, while there are infinitely many pairs of rational points in these balls and many of these pairs will contain a great deal of "spurious" mutual information (e.g., *any* finite message can be encoded into both elements of such a pair), a pair of rational points p and q achieving the minimum $I(p : q) = I_r(x : y)$ will only share information that their proximities to x and y force them to share. Sections 2 and 3 below develop the ideas that we have sketched in this paragraph, along with some elements of the fine-scale geometry of algorithmic information in Euclidean space that are needed for our results. A modest generalization of Levin's coding theorem (Theorem 2.1) is essential for this work.

In analogy with the characterizations (1.1) we define our mutual dimensions as the lower and upper densities of algorithmic mutual information,

$$mdim(x : y) = \liminf_{r\to\infty} \frac{I_r(x : y)}{r}, \; Mdim(x : y) = \limsup_{r\to\infty} \frac{I_r(x : y)}{r}, \tag{1.2}$$

in section 4. We also prove in that section that these quantities satisfy all but one of the desiderata (e.g., see [2]) for any satisfactory notion of mutual information.

We save the most important desideratum — our main theorem — for section 5. This is the data processing inequality for mutual dimension (actually two inequalities, one for $mdim$ and one for $Mdim$). The data processing inequality of Shannon information theory [5] says that, for any two probability spaces X and Y and any function $f : X \to Y$,

$$I(f(X) : Y) \leq I(X : Y), \tag{1.3}$$

i.e., the induced probability space $f(X)$ obtained by "processing the information in X through f" does not share any more information with Y than X shares with Y. The data processing inequality of algorithmic information theory [15] says that, for any computable partial function $f : \{0,1\}^* \to \{0,1\}^*$, there is a constant $c_f \in \mathbb{N}$ (essentially the number of bits in a program that computes f) such that, for all strings $x \in \mathit{dom}\, f$ and $y \in \{0,1\}^*$,

$$I(f(x) : y) \leq I(x : y) + c_f. \tag{1.4}$$

That is, modulo the constant c_f, $f(x)$ contains no more information about y than x contains about y.

The data processing inequality for points in Euclidean space is a theorem about functions $f : \mathbb{R}^m \to \mathbb{R}^n$ that are computable in the sense of computable analysis [3, 12, 24]. Briefly, an *oracle* for a point $x \in \mathbb{R}^m$ is a function $g_x : \mathbb{N} \to \mathbb{Q}^m$ such that $|g_x(r) - x| \leq 2^{-r}$ holds for all $r \in \mathbb{N}$. A function $f : \mathbb{R}^m \to \mathbb{R}^n$ is *computable* if there is an oracle Turing machine M such that, for every $x \in \mathbb{R}^m$ and every oracle g_x for x, the function $r \to M^{g_x}(r)$ is an oracle for $f(x)$.

Given (1.2), (1.3), and (1.4), it is natural to conjecture that, for every computable function $f : \mathbb{R}^m \to \mathbb{R}^n$, the inequalities

$$\mathit{mdim}(f(x) : y) \leq \mathit{mdim}(x : y),\ \mathit{Mdim}(f(x) : y) \leq \mathit{Mdim}(x : y) \tag{1.5}$$

hold for all $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^t$. However, this is not the case. For a simple example, there exist computable functions $f : \mathbb{R} \to \mathbb{R}^2$ that are *space-filling*, e.g., satisfy $[0,1]^2 \subseteq \mathit{range}\, f$ [4]. For such a function f we can choose $x \in \mathbb{R}$ such that $\mathit{dim}(f(x)) = 2$. Letting $y = f(x)$, we then have

$$\mathit{mdim}(f(x) : y) = \mathit{dim}(f(x)) = 2 > 1 \geq \mathit{Dim}(x) \geq \mathit{Mdim}(x : y),$$

whence both inequalities in (1.5) fail.

The difficulty here is that the above function f is extremely sensitive to its input, and this enables it to compress a great deal of "sparse" high-precision information about its input x into "dense" lower-precision information about its output $f(x)$. Many theorems of mathematical analysis exclude such excessively sensitive functions by assuming a given function f to be *Lipschitz*, meaning that there is a real number $c > 0$ such that, for all x and x', $|f(x) - f(x')| \leq c|x - x'|$. This turns out to be exactly what is needed here. In section 5 we prove prove the *data processing inequality* for mutual dimension (Theorem 5.1), which says that the conditions (1.5) hold for every function $f : \mathbb{R}^m \to \mathbb{R}^n$ that is computable and Lipschitz. In fact, we derive the data processing inequality from the more general *modulus processing lemma* (Lemma 5.2). This lemma yields quantitative variants of the data processing inequality for other classes of functions. For example, we use the modulus processing lemma to prove that, if $f : \mathbb{R}^m \to \mathbb{R}^n$ is *Hölder with exponent* α (meaning that $0 < \alpha \leq 1$ and there is a real number $c > 0$ such that $|f(x) - f(x')| \leq c|x - x'|^\alpha$ for all $x, x' \in \mathbb{R}^m$), then the inequalities

$$\mathit{mdim}(f(x) : y) \leq \frac{1}{\alpha}\mathit{mdim}(x : y),\ \mathit{Mdim}(f(x) : y) \leq \frac{1}{\alpha}\mathit{Mdim}(x : y) \tag{1.6}$$

hold for all $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^t$.

In section 5 we also derive *reverse* data processing inequalities, e.g., giving conditions under which $\mathit{mdim}(x : y) \leq \mathit{mdim}(f(x) : y)$. We then use data processing inequalities and their reverses to explore conditions under which computable functions on Euclidean space preserve, approximately preserve, or otherwise transform mutual dimensions between points.

We expect mutual dimensions and the data processing inequalities to be useful for future research in computable analysis. We also expect the development of mutual dimensions in Euclidean spaces — highly structured spaces in which it is clear that *mdim* and *Mdim* are the right notions — to guide future explorations of mutual information in more challenging contexts, including computational complexity and the computational theory of chemical reaction networks.

2 Kolmogorov Complexity in Euclidean Space

We begin by developing some elements of the fine-scale geometry of algorithm information in Euclidean space. In this context it is convenient to regard the Kolmogorov complexity of a set of strings to be the number of bits required to specify *some* element of the set.

Definition (Shen and Vereshchagin [21]). The *Kolmogorov complexity* of a set $S \subseteq \{0,1\}^*$ is

$$K(S) = \min\{K(x) \,|\, x \in S\}.$$

Note that $S \subseteq T$ implies $K(S) \geq K(T)$. Intuitively, small sets may require "higher resolution" than large sets.

Similarly, we define the *algorithmic probability* of a set $S \subseteq \{0,1\}^*$ to be

$$\mathbf{m}(S) = \sum_{\substack{\pi \in \{0,1\}^* \\ U(\pi) \in S}} 2^{-|\pi|},$$

where U is the fixed universal Turing machine used in defining Kolmogorov complexity. For a single string $x \in \{0,1\}^*$, we write $\mathbf{m}(x) = \mathbf{m}(\{x\})$.

We need a generalization of Levin's coding theorem [13, 14] that is applicable to certain systems of disjoint sets.

Notation. Let $B \subseteq \mathbb{N} \times \mathbb{N} \times \{0,1\}^*$ and $r, s \in \mathbb{N}$.

1. The (r,s)-*block* of B is the set $B_{r,s} = \{x \in \{0,1\}^* \,|\, (r,s,x) \in B\}$.
2. The r^{th} *layer* of B is the sequence $B_r = (B_{r,t} \,|\, t \in \mathbb{N})$.

Definition. A *layered disjoint system* (LDS) is a set $B \subseteq \mathbb{N} \times \mathbb{N} \times \{0,1\}^*$ such that, for all $r, s, t \in \mathbb{N}$,

$$s \neq t \Rightarrow B_{r,s} \cap B_{r,t} = \emptyset.$$

Note that this definition only requires the sets within each layer of B to be disjoint.

▶ **Theorem 2.1** (LDS coding theorem). *For every computably enumerable layered disjoint system B there is a constant $c_B \in \mathbb{N}$ such that, for all $r, t \in \mathbb{N}$,*

$$K(B_{r,t}) \leq \log \frac{1}{\mathbf{m}(B_{r,t})} + K(r) + c_B.$$

If we take $B_{r,t} = \{s_t\}$, where s_t is the t^{th} element of the standard enumeration of $\{0,1\}^*$, then Theorem 2.1 tells us that $K(x) \leq \log \frac{1}{\mathbf{m}(x)} + O(1)$, which is Levin's coding theorem.

Our next objective is to use the LDS coding theorem to obtain useful bounds on the number of times that the value $K(S)$ is attained or approximated.

Definition. Let $S \subseteq \{0,1\}^*$ and $d \in \mathbb{N}$.
1. A *d-approximate K-minimizer* of S is a string $x \in S$ for which $K(x) \leq K(S) + d$.
2. A *K-minimizer* of S is a 0-approximate K-minimizer of S.

We use the LDS coding theorem to prove the following.

▶ **Theorem 2.2.** *For every computably enumerable layered disjoint system B there is a constant $c_B \in \mathbb{N}$ such that, for all $r, t, d \in \mathbb{N}$, the block $B_{r,t}$ has at most $2^{d+K(r)+c_B}$ d-approximate K-minimizers.*

We now lift our terminology and notation to Euclidean space $\mathbb{R}^n$. In this context, a layered disjoint system is a set $B \subseteq \mathbb{N} \times \mathbb{N} \times \mathbb{R}^n$ such that, for all $r, s, t \in \mathbb{N}$,

$$s \neq t \Rightarrow B_{r,s} \cap B_{r,t} = \emptyset.$$

We lift our Kolmogorov complexity notation and terminology to $\mathbb{R}^n$ in two steps:
1. Lifting to $\mathbb{Q}^n$: Each rational point $q \in \mathbb{Q}^n$ is encoded as a string $x \in \{0,1\}^*$ in a natural way. We then write $K(q)$ for $K(x)$. In this manner, $K(S)$, $\mathbf{m}(S)$, K-minimizers, and d-approximate K-minimizers are all defined for sets $S \subseteq \mathbb{Q}^n$.
2. Lifting to $\mathbb{R}^n$. For $S \subseteq \mathbb{R}^n$, we define $K(S) = K(S \cap \mathbb{Q}^n)$ and $\mathbf{m}(S) = \mathbf{m}(S \cap \mathbb{Q}^n)$. Similarly, a K-minimizer for S is a K-minimizer for $S \cap \mathbb{Q}^n$, etc.

For each $r \in \mathbb{N}$ and each $m = (m_1, \ldots, m_n) \in \mathbb{Z}^n$, let

$$Q_m^{(r)} = [m_1 \cdot 2^{-r},\ (m+1) \cdot 2^{-r}) \times \cdots \times [m_n \cdot 2^{-r},\ (m_n+1) \cdot 2^{-r})$$

be the *r-dyadic cube* at m. Note that each $Q_m^{(r)}$ is "half-open, half-closed" in such a way that, for each $r \in \mathbb{N}$, the family

$$\mathcal{Q}^{(r)} = \{Q_m^{(r)} \mid m \in \mathbb{Z}^n\}$$

is a partition of $\mathbb{R}^n$. It follows that (modulo trivial encoding) the collection

$$\mathcal{Q} = \{Q_m^{(r)} \mid r \in \mathbb{N} \text{ and } m \in \mathbb{Z}^n\}$$

of all *dyadic cubes* is a layered disjoint system whose rth layer is $\mathcal{Q}^{(r)}$. Moreover, the set

$$\{(r, m, q) \in \mathbb{N} \times \mathbb{Z}^n \times \mathbb{Q}^n \mid q \in Q_m^{(r)}\}$$

is decidable, so Theorem 2.2 has the following useful consequence.

▶ **Corollary 2.3.** *There is a constant $c \in \mathbb{N}$ such that, for all $r, d \in \mathbb{N}$, no r-dyadic cube has more than $2^{d+K(r)+c}$ d-approximate K-minimizers. In particular, no r-dyadic cube has more than $2^{K(r)+c}$ K-minimizers.*

The Kolmogorov complexity of an arbitrary point in Euclidean space depends on both the point and a precision parameter.

Definition. Let $x \in \mathbb{R}^n$ and $r \in \mathbb{N}$. The *Kolmogorov complexity* of x at precision r is

$$K_r(x) = K(B_{2^{-r}}(x)).$$

That is, $K_r(x)$ is the number of bits required to specify *some* rational point in the open ball $B_{2^{-r}}(x)$. Note that, for each $q \in \mathbb{Q}^n$, $K_r(q) \nearrow K(q)$ as $r \to \infty$.

A careful analysis of the relationship between cubes and balls enables us to derive the following from Corollary 2.3.

▶ **Theorem 2.4.** *There is a constant $c \in \mathbb{N}$ such that, for all $r, d \in \mathbb{N}$, no open ball of radius 2^{-r} has more than $2^{d+2K(r)+c}$ d-approximate K-minimizers. In particular, no open ball of radius 2^{-r} has more than $2^{2K(r)+c}$ K-minimizers.*

3 Mutual Information in Euclidean Space

This section develops the mutual dimensions of points in Euclidean space at a given precision. As in section 2 we assume that rational points $q \in \mathbb{Q}^n$ are encoded as binary strings in some natural way. Mutual information between rational points is then defined from conditional Kolmogorov complexity in the standard way [15] as follows.

Definition. Let $p \in \mathbb{Q}^m$, $r \in \mathbb{Q}^n$, $s \in \mathbb{Q}^t$.

1. The *mutual information* between p and q is

$$I(p : q) = K(q) - K(q \,|\, p).$$

2. The *mutual information* between p and q *given* s is

$$I(p : q|s) = K(q \,|\, s) - K(q \,|\, p, s).$$

The following properties of mutual information are well known [15].

▶ **Theorem 3.1.** *Let $p \in \mathbb{Q}^m$ and $q \in \mathbb{Q}^n$.*

1. $I(p, K(p) : q) = K(p) + K(q) - K(p, q) + O(1)$.
2. $I(p, K(p) : q) = I(q, K(q) : p) + O(1)$.
3. $I(p : q) \leq \min\{K(p), K(q)\} + O(1)$.

(Each of the properties 1 and 2 above is sometimes called *symmetry of mutual information.*)

Mutual information between points in Euclidean space at a given precision is now defined as follows.

Definition. The *mutual information* of $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^t$ at *precision* $r \in \mathbb{N}$ is

$$I_r(x : y) = \min\{I(q_x : q_y) \,|\, q_x \in B_{2^{-r}}(x) \cap \mathbb{Q}^n \text{ and } q_y \in B_{2^{-r}}(y) \cap \mathbb{Q}^t\}.$$

As noted in the introduction, the role of the minimum in the above definition is to eliminate "spurious" information that points $q_x \in B_{2^{-r}} \cap \mathbb{Q}^n$ and $q_y \in B_{2^{-r}}(y) \cap \mathbb{Q}^t$ might share for reasons not forced by their proximities to x and y, respectively.

Notation. We also use the quantity

$$J_r(x : y) = \min\{I(q_x : q_y) \,|\, p_x \text{ is a K-minimizer of } B_{2^{-r}}(x) \text{ and } p_y \text{ is a K–minimizer of } B_{2^{-r}}(y)\}.$$

Although $J_r(x : y)$, having two "layers of minimization", is somewhat more involved than $I_r(x : y)$, one can imagine using it as the definition of mutual information. Using Theorem 2.4 (and hence the LDS coding theorem), we prove the useful fact that $J_r(x : y)$ does not differ greatly from $I_r(x : y)$.

▶ **Theorem 3.2.** *For all $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^t$,*

$$I_r(x : y) = J_r(x : y) + o(r)$$

as $r \to \infty$.

Using this we establish the following useful properties of $I_r(x : y)$.

▶ **Theorem 3.3.** *For all $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^t$, the following hold as $r \to \infty$.*

1. $I_r(x : y) = K_r(x) + K_r(y) - K_r(x, y) + o(r)$.
2. $I_r(x : y) \leq \min\{K_r(x), K_r(y)\} + o(r)$.
3. *If x and y are independently random, then $I_r(x : y) = o(r)$.*
4. $I_r(x : y) = I_r(y : x) + o(r)$.

4 Mutual Dimension in Euclidean Space

We now define mutual dimensions between points in Euclidean space(s).

Definition. The *lower* and *upper mutual dimensions* between $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^t$ are

$$mdim(x : y) = \liminf_{r \to \infty} \frac{I_r(x : y)}{r}$$

and

$$Mdim(x : y) = \limsup_{r \to \infty} \frac{I_r(x : y)}{r},$$

respectively.

With the exception of the data processing inequality, which we prove in section 5, the following theorem says that the mutual dimensions $mdim$ and $Mdim$ have the basic properties that any mutual information measure should have. (See, for example, [2].)

▶ **Theorem 4.1.** *For all $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^t$, the following hold.*

1. $mdim(x : y) \geq dim(x) + dim(y) - Dim(x, y)$.
2. $Mdim(x : y) \leq Dim(x) + Dim(y) - dim(x, y)$.
3. $mdim(x : y) \leq \min\{dim(x), dim(y)\}$, $Mdim(x : y) \leq \min\{Dim(x), Dim(y)\}$.
4. $0 \leq mdim(x : y) \leq Mdim(x : y) \leq \min\{n, t\}$.
5. *If x and y are independently random, then $Mdim(x : y) = 0$.*
6. $mdim(x : y) = mdim(y : x)$, $Mdim(x : y) = Mdim(y : x)$.

5 Data Processing Inequalities and Applications

Our objectives in this section are to prove data processing inequalities for lower and upper mutual dimensions in Euclidean space and to use these inequalities to investigate how certain functions on Euclidean space preserve or predictably transform mutual dimensions.

The following result is our main theorem. The meaning and necessity of the Lipschitz hypothesis are explained in the introduction.

▶ **Theorem 5.1** (data processing inequality). *If $f : \mathbb{R}^n \to \mathbb{R}^t$ is computable and Lipschitz, then, for all $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^t$,*

$$mdim(f(x) : y) \leq mdim(x : y)$$

and

$$Mdim(f(x) : y) \leq Mdim(x : y).$$

We in fact prove a stronger result.

Definition. A *modulus (of uniform continuity)* for a function $f : \mathbb{R}^n \to \mathbb{R}^k$ is a nondecreasing function $m : \mathbb{N} \to \mathbb{N}$ such that, for all $x, y \in \mathbb{R}^n$ and $r \in \mathbb{N}$,

$$|x - y| \leq 2^{-m(r)} \Rightarrow |f(x) - f(y)| \leq 2^{-r}.$$

▶ **Lemma 5.2** (modulus processing lemma). *If $f : \mathbb{R}^n \to \mathbb{R}^k$ is a computable and uniformly continuous function, and m is a computable modulus for f, then for all $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^t$,*

$$mdim(f(x) : y) \leq mdim(x : y)\left(\limsup_{r\to\infty} \frac{m(r+1)}{r}\right)$$

and

$$Mdim(f(x) : y) \leq Mdim(x : y)\left(\limsup_{r\to\infty} \frac{m(r+1)}{r}\right).$$

Theorem 5.1 follows immediately from Lemma 5.2 and the following.

▶ **Observation 5.3.** *If a function $f : \mathbb{R}^n \to \mathbb{R}^k$ is Lipschitz, then there exists $s \in \mathbb{N}$ such that $m(r) = r + s$ is a modulus for f.*

In similar fashion, we can derive the following fact from the modulus processing lemma. (Recall the definition of Hölder functions given in the introduction.)

▶ **Corollary 5.4.** *If $f : \mathbb{R}^n \to \mathbb{R}^k$ is computable and Hölder with exponent α, then, for all $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^t$,*

$$mdim(f(x) : y) \leq \frac{1}{\alpha} mdim(x : y)$$

and

$$Mdim(f(x) : y) \leq \frac{1}{\alpha} Mdim(x : y).$$

We next develop reverse versions of the above inequalities.

Notation. Let $n \in \mathbb{Z}^+$.

1. $[n] = \{1, \cdots, n\}$.
2. For $S \subseteq [n]$, $x \in \mathbb{R}^{|S|}$, $y \in \mathbb{R}^{n-|S|}$, the string

$$x *_S y \in \mathbb{R}^n$$

is obtained by placing the components of x into the positions in S (in order) and the components of y into the positions in $[n] \setminus S$ (in order).

Definition. Let $f : \mathbb{R}^n \to \mathbb{R}^k$.

1. f is *co-Lipschitz* if there is a real number $c > 0$ such that for all $x, y \in \mathbb{R}^n$,

$$|f(x) - f(y)| \geq c|x - y|.$$

2. f is *bi-Lipschitz* if f is both Lipschitz and co-Lipschitz.
3. For $S \subseteq [n]$, f is *S-co-Lipschitz* if there is a real number $c > 0$ such that, for all $u, v \in \mathbb{R}^{|S|}$ and $y \in \mathbb{R}^{n-|S|}$,

$$|f(u *_S y) - f(v *_S y)| \geq c|u - v|.$$

4. For $i \in [n]$, f is *co-Lipschitz in its i^{th} argument* if f is $\{i\}$-co-Lipschitz.

Note that f is $[n]$-co-Lipschitz if and only if f is co-Lipschitz.

Example. The function $f : \mathbb{R}^n \to \mathbb{R}$ defined by

$$f(x_1, \cdots, x_n) = x_1 + \cdots + x_n$$

is S-co-Lipschitz if and only if $|S| \leq 1$. In particular, if $n \geq 2$, then f is co-Lipschitz in every argument, but f is not co-Lipschitz.

We next relate co-Lipschitz conditions to moduli.

Definition. Let $f : \mathbb{R}^n \to \mathbb{R}^k$.

1. An *inverse modulus* for f is a nondecreasing function $m : \mathbb{N} \to \mathbb{N}$ such that, for all $x, y \in \mathbb{R}^n$ and $r \in \mathbb{N}$,
$$|f(x) - f(y)| \le 2^{-m(r)} \Rightarrow |x - y| \le 2^{-r}.$$
2. Let $S \subseteq [n]$. An *S-inverse modulus* for f is a nondecreasing function $m : \mathbb{N} \to \mathbb{N}$ such that, for all $u, v \in \mathbb{R}^{|S|}$, all $y \in \mathbb{R}^{n-|S|}$, and all $r \in \mathbb{N}$,
$$|f(u *_S y) - f(v *_S y)| \le 2^{-m(r)} \Rightarrow |u - v| \le 2^{-r}.$$
3. Let $i \in [n]$. An *inverse modulus* for f *in its i^{th} argument* is an $\{i\}$-inverse modulus for f.

▶ **Observation 5.5.** *Let $f : \mathbb{R}^n \to \mathbb{R}^k$ and $S \subseteq [n]$.*

1. *If f is S-co-Lipschitz, then there is a positive constant $t \in \mathbb{N}$ such that $m'(r) = r + t$ is an S-inverse modulus of f.*
2. *If f is co-Lipschitz, then there is a positive constant $t \in \mathbb{N}$ such that $m'(r) = r + t$ is an inverse modulus of f.*

▶ **Lemma 5.6** (reverse modulus processing lemma)**.** *If $f : \mathbb{R}^n \to \mathbb{R}^k$ is a uniformly continuous function and m' is a computable S-inverse modulus for f, then, for all $S \subseteq [n]$, $x \in \mathbb{R}^{|S|}$, $y \in \mathbb{R}^t$, and $z \in \mathbb{R}^{n-|S|}$,*
$$mdim(x : y) \le mdim((f(x *_S z), z) : y)\left(\limsup_{r\to\infty} \frac{m'(r+1)}{r}\right)$$
and
$$Mdim(x : y) \le Mdim((f(x *_S z), z) : y)\left(\limsup_{r\to\infty} \frac{m'(r+1)}{r}\right).$$

By Observation 5.5 and Lemma 5.6, we have the following.

▶ **Theorem 5.7** (reverse data processing inequality)**.** *If $S \subseteq [n]$ and $f : \mathbb{R}^n \to \mathbb{R}^k$ is computable and S-co-Lipschitz, then, for all $x \in \mathbb{R}^{|S|}$, $y \in \mathbb{R}^t$, and $z \in \mathbb{R}^{n-|S|}$,*
$$mdim(x : y) \le mdim((f(x *_S z), z) : y)$$
and
$$Mdim(x : y) \le Mdim((f(x *_S z), z) : y).$$

Definition. Let $f : \mathbb{R}^n \to \mathbb{R}^k$ and $0 < \alpha \le 1$.

1. f is *co-Hölder with exponent α* if there is a real number $c > 0$ such that for all $x, y \in \mathbb{R}^n$,
$$|x - y| \le c|f(x) - f(y)|^\alpha.$$
2. For $S \subseteq [n]$, f is *S-co-Hölder with exponent α* if there is a real number $c > 0$ such that, for all $u, v \in \mathbb{R}^{|S|}$ and $y \in \mathbb{R}^{n-|S|}$,
$$|u - v| \le c|f(u *_S y) - f(v *_S y)|^\alpha.$$

▶ **Corollary 5.8.** *If $S \subseteq [n]$ and $f : \mathbb{R}^n \to \mathbb{R}^k$ is computable and S-co-Hölder with exponent α, then, for all $x \in \mathbb{R}^{|S|}$, $y \in \mathbb{R}^t$, and $z \in \mathbb{R}^{n-|S|}$,*
$$mdim(x : y) \le \frac{1}{\alpha} mdim((f(x *_S z), z) : y)$$
and
$$Mdim(x : y) \le \frac{1}{\alpha} Mdim((f(x *_S z), z) : y).$$

The rest of this section is devoted to applications of the data processing inequalities and their reverses.

▶ **Theorem 5.9** (mutual dimension conservation inequality). *If $f : \mathbb{R}^n \to \mathbb{R}^k$ and $g : \mathbb{R}^t \to \mathbb{R}^l$ are computable and Lipschitz, then, for all $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^t$,*

$$mdim(f(x) : g(y)) \leq mdim(x : y)$$

and

$$Mdim(f(x) : g(y)) \leq Mdim(x : y).$$

▶ **Theorem 5.10** (reverse mutual dimension conservation inequality). *Let $S_1 \subseteq [n]$ and $S_2 \subseteq [t]$. If $f : \mathbb{R}^n \to \mathbb{R}^k$ is computable and S_1-co-Lipschitz, and $g : \mathbb{R}^t \to \mathbb{R}^l$ is computable and S_2-co-Lipschitz, then, for all $x \in \mathbb{R}^{|S_1|}$, $y \in \mathbb{R}^{|S_2|}$, $w \in \mathbb{R}^{n-|S_1|}$, and $z \in \mathbb{R}^{t-|S_2|}$,*

$$mdim(x : y) \leq mdim((f(x *_S w), w) : (g(y *_S z), z))$$

and

$$Mdim(x : y) \leq Mdim((f(x *_S w), w) : (g(y *_S z), z))$$

▶ **Corollary 5.11** (preservation of mutual dimension). *If $f : \mathbb{R}^n \to \mathbb{R}^k$ and $g : \mathbb{R}^t \to \mathbb{R}^l$ are computable and bi-Lipschitz, then, for all $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^t$,*

$$mdim(f(x) : g(y)) = mdim(x : y)$$

and

$$Mdim(f(x) : g(y)) = Mdim(x : y).$$

▶ **Corollary 5.12.** *If $f : \mathbb{R}^n \to \mathbb{R}^k$ and $g : \mathbb{R}^t \to \mathbb{R}^l$ are computable and Hölder with exponents α and β, respectively, then, for all $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^t$,*

$$mdim(f(x) : g(y)) \leq \frac{1}{\alpha\beta} mdim(x : y)$$

and

$$Mdim(f(x) : g(y)) \leq \frac{1}{\alpha\beta} Mdim(x : y).$$

▶ **Corollary 5.13.** *Let $S_1 \subseteq [n]$ and $S_2 \subseteq [t]$. If $f : \mathbb{R}^n \to \mathbb{R}^k$ is computable and S_1-co-Hölder with exponent α, and $g : \mathbb{R}^t \to \mathbb{R}^l$ is computable and S_2-co-Hölder with exponent β, then, for all $x \in \mathbb{R}^{|S_1|}$, $y \in \mathbb{R}^{|S_2|}$, $w \in \mathbb{R}^{n-|S_1|}$, and $z \in \mathbb{R}^{t-|S_2|}$,*

$$mdim(x : y) \leq \frac{1}{\alpha\beta} mdim((f(x *_S w), w) : (g(y *_S z), z))$$

and

$$Mdim(x : y) \leq \frac{1}{\alpha\beta} Mdim((f(x *_S w), w) : (g(y *_S z), z)).$$

Acknowledgments. We thank Elvira Mayordomo, A. Pavan, Giora Slutzki, and Jim Lathrop for useful discussions.

References

1 Krishna B. Athreya, John M. Hitchcock, Jack H. Lutz, and Elvira Mayordomo. Effective strong dimension in algorithmic information and computational complexity. *SIAM Journal of Computing*, 37(3):671–705, 2007.

2 C.B. Bell. Mutual information and maximal correlation as measures of dependence. *Annals of Mathematical Statistics*, 33(2):587–595, 1962.

3 Mark Braverman and Stephen Cook. Computing over the reals: foundations for scientific computing. *Notices of the American Mathematical Society*, 53(3):318–329, 2006.

4 P. J. Couch, B. D. Daniel, and Timothy H. McNicholl. Computing space-filling curves. *Theory of Computing Systems*, 50(2):370–386, 2012.

5 Thomas R. Cover and Joy A. Thomas. *Elements of Information Theory.* John Wiley & Sons, Inc., second edition, 2006.

6 Randall Dougherty, Jack H. Lutz, Daniel R. Mauldin, and Jason Teutsch. Translating the Cantor set by a random real. *Transactions of the American Mathematical Society*, to appear.

7 Kenneth Falconer. *Fractal Geometry: Mathematical Foundations and Applications.* Wiley, second edition, 2003.

8 Xiaoyang Gu, Jack Lutz, and Elvira Mayordomo. Curves that must be retraced. *Information and Computation*, 209(6):992–1006, 2011.

9 Xiaoyang Gu, Jack H. Lutz, and Elvira Mayordomo. Points on computable curves. In *Foundations of Computer Science*, pages 469–474. IEEE Computer Society, 2006.

10 John M. Hitchcock. Correspondence principles for effective dimensions. *Theory of Computing Systems*, 38(5):559–571, 2005.

11 Bjørn Kjos-Hanssen and Anil Nerode. Effective dimension of points visited by Brownian motion. *Theoretical Computer Science*, 410(4-5):347–354, 2009.

12 Ker-I Ko. *Complexity Theory of Real Functions.* Birkhäuser, first edition, 1991.

13 Leonid A. Levin. On the notion of a random sequence. *Soviet Mathematics Doklady*, 14(5):1413–1416, 1973.

14 Leonid A. Levin. Laws of information conservation (nongrowth) and aspects of the foundation of probability theory. *Problemy Peredachi Informatsii*, 10(3):30–35, 1974.

15 Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications.* Springer, third edition, 2008.

16 Jack H. Lutz. The dimension of individual strings and sequences. *Information and Computation*, 187(1):49–79, 2003.

17 Jack H. Lutz and Elvira Mayordomo. Dimensions of points in self-similar fractals. *SIAM Journal on Computing*, 38(3):1080–1112, 2008.

18 Jack H. Lutz and Klaus Weihrauch. Connectivity properties of dimension level sets. *Mathematical Logic Quarterly*, 54(5):483–491, 2008.

19 Elvira Mayordomo. A Kolmogorov complexity characterization of constructive Hausdorff dimension. *Information Processing Letters*, 84(1):1–3, 2002.

20 Robert Rettinger and Xizhong Zheng. Points on computable curves of computable lengths. In *MFCS*, pages 736–743. Springer, 2009.

21 Alexander Shen and Nikolai K. Vereshchagin. Logical operations and Kolmogorov complexity. *Theoretical Computer Science*, 271(1-2):125–129, 2002.

22 Daniel Turetsky. Connectedness properties of dimension level sets. *Theoretical Computer Science*, 412(29):3598–3603, 2011.

23 Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. A correction. *Proceedings of the London Mathematical Society*, 43(2):544–546, 1937.

24 Klaus Weihrauch. *Computable Analysis: An Introduction.* Springer, first edition, 2000.

Exact and Approximation Algorithms for the Maximum Constraint Satisfaction Problem over the Point Algebra

Yoichi Iwata[1] **and Yuichi Yoshida**[2]

1 **University of Tokyo**
7-3-1, Hongo, Bunkyo-ku, Tokyo, Japan
`y.iwata@is.s.u-tokyo.ac.jp`

2 **National Institute of Informatics and Preferred Infrastructure, Inc.**
2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo, Japan
`yyoshida@nii.ac.jp`

Abstract

We study the constraint satisfaction problem over the point algebra. In this problem, an instance consists of a set of variables and a set of binary constraints of forms $(x < y), (x \leq y), (x \neq y)$ or $(x = y)$. Then, the objective is to assign integers to variables so as to satisfy as many constraints as possible. This problem contains many important problems such as Correlation Clustering, Maximum Acyclic Subgraph, and Feedback Arc Set.

We first give an exact algorithm that runs in $O^*(3^{\frac{\log 5}{\log 6}n})$ time, which improves the previous best $O^*(3^n)$ obtained by a standard dynamic programming. Our algorithm combines the dynamic programming with the split-and-list technique. The split-and-list technique involves matrix products and we make use of sparsity of matrices to speed up the computation.

As for approximation, we give a 0.4586-approximation algorithm when the objective is maximizing the number of satisfied constraints, and give an $O(\log n \log \log n)$-approximation algorithm when the objective is minimizing the number of unsatisfied constraints.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Constraint Satisfaction Problems, Point Algebra, Exact Algorithms, Approximation Algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.127

1 Introduction

Problems involving temporal constraints arise in various areas of computer science such as scheduling, program verification, and parallel computation. One of the most common frameworks to express temporal constraints is temporal constraint satisfaction problems (Temporal CSPs). In Temporal CSP, an instance consists of a set of variables and a set of constraints defined by first-order sentences with the predicate $(<)$, the strict total order of integers. Then, the objective is to assign integers to variables so as to satisfy all the constraints.[1]

One of most famous Temporal CSPs is the CSP over the point algebra, introduced by Vilain and Kautz [18]. In this problem, we have constraints of forms $(x < y), (x \leq y), (x \neq y)$

[1] We often choose the domain as the set of rational numbers and the predicate $(<)$ as the dense strict total order of rational numbers (e.g., [5]). However, we choose the domain as the set of integers to simplify our expositions.

and $(x = y)$. A considerably larger class of Temporal CSPs is the CSP over the Ord-Horn relations, introduced by Nebel and Bürkert [13]. CSPs over the point algebra and over Ord-Horn relations are known to be solvable in polynomial time [13, 18]. Also, Ordering CSPs introduced in [9] can be formulated as Temporal CSPs in which all variables must be assigned different integers.

Most existing works on Temporal CSPs are concerned with the problem of deciding whether all constraints are satisfied [1, 5, 12, 18]. However, it is natural to ask for an assignment that satisfies as many constraints as possible when all constraints are not satisfied simultaneously. In this paper, we are especially interested in the CSP over the point algebra as it is the most fundamental Temporal CSP. We can look at the problem in terms of maximizing the number of satisfied constraints (Max-PA), or in terms of minimizing the number of unsatisfied constraints (Min-PA). These two are equivalent at optimality but, as usual, differ from the point of view of approximation.

First, we give an exact algorithm for Max-PA (and hence for Min-PA) running in $O^*(3^{\frac{\log 5}{\log 6}n})$ time, where n is the number of variables.[2][3] This result improves the current best $O^*(3^n)$ obtained by a standard dynamic programming. Our algorithm is obtained by combining the dynamic programming with the split-and-list technique due to Ryan Williams [19]. That is, we reduce computation of the dynamic programming to computation of matrix products. The reduction is not trivial since the original split-and-list technique is only used to speed-up exhaustive search. Using the current fastest algorithm for multiplying general square matrices by Vassilevska Williams [20], we obtain an algorithm that runs in $O^*(3^{\frac{\omega}{\log 6}n})$ time, where $\omega < 2.3727 < \log 6$. However, one of the matrices generated by the reduction is sparse and has some recursive structure. To make use of this property, we modify the algorithm due to Bini et al. [3] and get an algorithm that runs in $O^*(3^{\frac{\log 5}{\log 6}n})$ time.

Next, we give a 0.4586-approximation algorithm for Max-PA. The idea of our algorithm is similar to [16]. We first solve a semidefinite relaxation and round the solution using three or four hyperplanes (we take the better one). If two variables are in the same side for every hyperplane, they will get the same value. Thus, we use at most 16 values. The ordering of values assigned to different clusters is chosen randomly.

If we only use constraints of the form $(x < y)$, then Max-PA coincides with Maximum Acyclic Subgraph, in which we want to find an ordering of vertices in a digraph so as to maximize the number of edges that go forward. It is NP-Hard to get a $(0.5+\epsilon)$-approximation for any $\epsilon > 0$ assuming Khot's unique games conjecture [10, 11]. The hardness suggests that it would be difficult to improve our approximation ratio significantly.

Finally, we give an $O(\log n \log\log n)$-approximation algorithm for Min-PA. If we only use constraints of the form $(x < y)$, then Min-PA coincides with Feedback Arc Set, in which we want to find an ordering of vertices in a digraph so as to minimize the number of edges that go backward. The best algorithm for Feedback Arc Set has an approximation ratio $O(\log n \log\log n)$ [8]. Thus, our algorithm can be seen as a generalization of the algorithm for Feedback Arc Set. The idea of our algorithm is reducing the problem to a variant of multicut problem, which we call the *symmetric multicut problem.* In this problem, we are given a digraph $G = (V, E)$, and a set of terminal pairs $T = \{(s_1, t_1), \ldots, (s_k, t_k)\}$. Then, we want to find an edge set $F \subseteq E$ of minimum cardinality such that, for any terminal pair $(s, t) \in T$, $G - F$ contains no path from s to t or no path from t to s. Then, we obtain an $O(\log n \log\log n)$-approximation algorithm for the symmetric multicut problem.

[2] $O^*(\cdot)$ hides a factor polynomial in n.
[3] We denote by log the logarithm to the base 2.

1.1 Related Works

By restricting types of constraints further, the CSP over the point algebra coincides with many other problems. If we use constraints of the form $(x = y)$ and $(x \neq y)$ only, then we get Correlation Clustering [2]. As far as we know, exact algorithms for Correlation Clustering is not studied in the literature. However, we can easily obtain an $O^*(2^n)$-time algorithm by a simple application of fast subset convolution by Björklund et al. [4] (see Section 3 for detail). If the underlying graph is a complete graph, then the maximization version admits PTAS and the minimization version can be approximated within a factor of 4 [6]. For general underlying graphs, the maximization version can be approximated within a factor of 0.7666 [16] and the minimization version can be approximated within a factor of $O(\log n)$ [6].

If we use constraints of the form $(x < y)$ only, then Max-PA coincides with Maximum Acyclic Subgraph and Min-PA coincides with Feedback Arc Set. The current fastest exact algorithm for Maximum Acyclic Subgraph (and hence Feedback Arc Set) is a simple $O^*(2^n)$-time dynamic programming. The current best approximation algorithm for Maximum Acyclic Subgraph is just the random assignment and its approximation ratio is 1/2. As we mentioned, Guruswami et al. [10] showed that it is NP-Hard to obtain $(1/2 + \epsilon)$-approximation for any $\epsilon > 0$ assuming Khot's unique games conjecture. As for Feedback Arc Set, there is an $O(\log n \log\log n)$-approximation algorithm [8]. It is known that obtaining 1.36-approximation is NP-Hard [7] and obtaining any constant approximation ratio is NP-Hard assuming Khot's unique games conjecture [10].

A Temporal CSP with a single predicate is called an Ordering CSP if all variables must be assigned different values. Guruswami et al. [9, 10] considered the maximization version of Ordering CSPs, and they showed that the random assignment always gives the best approximation ratio assuming Khot's unique games conjecture.

1.2 Organization

We give definitions used in this paper in Section 2. In Section 3, we show an $O^*(3^{\frac{\log 5}{\log 6}n})$-time exact algorithm for Max-PA. Sections 4 and 5 are devoted to show a 0.4586-approximation algorithm for Max-PA and an $O(\log n \log\log n)$-approximation algorithm for Min-PA, respectively.

2 Preliminaries

For an integer n, we denote by $[n]$ the set $\{1, \dots, n\}$. A *(d-ary) relation* over a domain $[k]$ is a subset of $[k]^d$, and a *(d-ary) constraint* is a pair of a tuple of d variables and a d-ary relation. A constraint $e = (\{x_1, \dots, x_t\}, R)$ is called *satisfied* by an assignment f if $(f(x_1), \dots, f(x_t)) \in R$. Now, we define two problems, Max-PA, and Min-PA.

Max-PA
Input: A set of n variables V and a set of m constraints C. Each variable $x \in V$ takes value from $[n]$, and each constraint is of the forms $(x < y), (x \leq y), (x \neq y)$ and $(x = y)$.
Output: An assignment $f : V \to [n]$ that maximizes the number of satisfied constraints.

Min-PA
Input: Same as Max-PA.
Output: An assignment $f : V \to [n]$ that minimizes the number of unsatisfied constraints.

Since the number of unsatisfied constraints is the number of constraints minus the number of satisfied constraints, the optimal assignments for the two problems coincide. Thus, for the exact algorithm, we deal with Max-PA only. In this paper, we only deal with unweighted instances, but our argument can be easily extended to weighted instances, for which we want to maximize (resp., minimize) the total weight of satisfied (resp., unsatisfied) constraints. If weights are integers between $-W$ and W, then the running time of our exact algorithm takes additional $O^*(W)$ factor. Running times of our approximation algorithms do not change.

3 Exact Algorithms

In this section, we give an exact algorithm for Max-PA and prove the following theorem.

▶ **Theorem 1.** *Max-PA can be solved in $O^*(3^{\frac{\min(\omega,\log 5)}{\log 6}n})$ time and $O^*(3^{\frac{2}{\log 6}n})$ space, where ω is the matrix product exponent.*

We cannot compare ω and $\log 5$ since we do not know the true value of ω. The current best bound on ω is 2.3727 by Vassilevska Williams [20], which is larger than $\log 5 < 2.3220$.

Before proving Theorem 1, we first show an $O^*(2^n)$-time algorithm for Correlation Clustering to see the difficulty of Max-PA. We formalize Correlation Clustering as a dynamic programming and solve it using fast subset convolution to achieve the time complexity $O^*(2^n)$. This can be done because of the simplicity of the recurrence in the dynamic programming.

Then, we introduce a standard $O^*(3^n)$-time dynamic programming algorithm for Max-PA. We will see that we cannot apply fast subset convolution to the recurrence. This is the point we become apart from Correlation Clustering, and we improve the running time of the algorithm to $O^*(3^{\frac{\omega}{\log 6}n})$ by applying the split-and-list technique to compute the recurrences. Finally, we further improve the running time by using structure of matrices involved when applying the split-and-list technique and give an $O^*(3^{\frac{\log 5}{\log 6}n})$-time algorithm.

3.1 Algorithm for Correlation Clustering

We explain an $O^*(2^n)$-time dynamic programming algorithm for Correlation Clustering. Recall that Correlation Clustering is a special case of Max-PA such that each constraint has the form $(x = y)$ and $(x \neq y)$ only. First, we reduce an instance of unweighted Correlation Clustering into an instance of weighted Correlation Clustering that has $(=)$-constraints only. This can be done by replacing each constraint of the form $(x \neq y)$ by a constraint $(x = y)$ with weight -1. If an assignment satisfies a removed constraint $(x \neq y)$, then it does not satisfy the added constraint $(x = y)$ and contributes to the objective value of the reduced instance by 0. Otherwise it satisfies the added constraint and contributes to the objective value by -1. Thus the difference of its contribution to the original instance and the reduced instance is always a fixed constant. Therefore, the optimal assignment does not change through the reduction.

Then, we solve the reduced instance by dynamic programming. For a subset $S \subseteq V$, we define $\mathrm{dp}_i(S)$ as the maximum total weight of satisfied constraints by assigning values from $[i]$ to S. When some variable in a constraint is not assigned any value, we simply regard that the constraint is not satisfied. We define $\mathrm{dp}_0(\emptyset) = 0$ and $\mathrm{dp}_0(S) = -\infty$ for any $S \neq \emptyset$. The optimal value can be obtained as $\mathrm{dp}_n(V)$. We can compute dp_{i+1} from dp_i by the following recurrence:

$$\mathrm{dp}_{i+1}(S) = \max_{T \subseteq S}\{\mathrm{dp}_i(T) + w(S \setminus T)\}, \tag{1}$$

where $w(S)$ is the total weight of the constraints of the form $(x = y)$ with $x \in S$ and $y \in S$. The running time of this dynamic programming is $O^*(\sum_{i=0}^{n} \binom{n}{i} 2^i) = O^*(3^n)$. Björklund et al. [4] showed that any recurrence of this form can be computed in $O^*(2^n)$ time by developing a technique called fast subset convolution. Thus, we can solve Correlation Clustering in $O^*(2^n)$ time.

3.2 Standard Dynamic Programming Algorithm

We explain an $O^*(3^n)$-time dynamic programming algorithm for Max-PA. First, we reduce an instance of unweighted Max-PA into an instance of weighted Max-PA that has $(<)$-constraints only. This can be done by the following reduction.

- For each constraint of the form $(x < y)$, we set its weight as 1.
- For each constraint of the form $(x \leq y)$, we replace it by a constraint $(x > y)$ with weight -1.
- For each constraint of the form $(x \neq y)$, we replace it by two constraints $(x < y)$ and $(x > y)$ with weight 1.
- For each constraint of the form $(x = y)$, we replace it by two constraints $(x < y)$ and $(x > y)$ with weight -1.

If an assignment satisfies a removed constraint $(x \leq y)$, then it does not satisfy the added constraint $(x > y)$ and contributes to the objective value of the reduced instance by 0. Otherwise it satisfies the added constraint and contributes to the objective value by -1. Thus the difference of its contribution to the original instance and the reduced instance is always a fixed constant. We have the same property for constraints $(x \neq y)$ and $(x = y)$. Therefore, the optimal assignment does not change through the reduction.

Then, we solve the reduced instance by dynamic programming. For a subset $S \subseteq V$, we define $\mathrm{dp}_i(S)$ as we did in the previous subsection. We can compute dp_{i+1} from dp_i by the following recurrence:

$$\mathrm{dp}_{i+1}(S) = \max_{T \subseteq S} \{\mathrm{dp}_i(T) + w(T, S \setminus T)\}, \tag{2}$$

where $w(A, B)$ is the total weight of constraints of the form $(x < y)$ with $x \in A$ and $y \in B$. The running time of this dynamic programming is $O^*(3^n)$ and it uses $O^*(2^n)$ space.

Compare to the recurrence (1) for Correlation Clustering, w in this recurrence does not have the form of $w(S \setminus T)$ but has the form of $w(T, S \setminus T)$ to which we cannot apply fast subset convolution. This is because, in order to determine whether a constraint is satisfied, we need to know not only how variables are partitioned into equal-valued sets, but also the ordering of those sets.

3.3 Split-and-List Algorithm for Max-PA

In this subsection, we improve the standard dynamic programming algorithm in the previous subsection by applying split-and-list technique. The original technique was developed to speed-up the exhaustive search for Max 2-SAT by Ryan Williams [19], and here, we demonstrate that it can be used to speed-up the computation of the recurrence in the dynamic programming. In the original technique for Max 2-SAT, we split the variable set into three equal-sized parts A, B, and C. Then we create two matrices X and Y from the number of satisfied clauses by each assignment on $A \cup B$ and $B \cup C$, respectively, so that we can obtain the maximum number of satisfied clauses from the product XY. In our application, we create two matrices X and Y from values of dp_i so that we can obtain values of dp_{i+1} from the product XY.

▶ **Lemma 2.** *Max-PA can be solved in* $O^*(3^{\frac{\omega}{\log 6}n})$ *time and* $O^*(3^{\frac{2}{\log 6}n})$ *space.*

Proof. We divide the variables into two parts V_1 and V_2 so that $\alpha|V_1| = |V_2|$, where $\alpha \geq 1$ is a parameter. Then, for $S_1 \subseteq V_1$ and $S_2 \subseteq V_2$, we can rewrite the recurrence of the dynamic programming (2) as follows.

$$\begin{aligned}
\mathrm{dp}_{i+1}(S_1 \cup S_2) &= \max_{T \subseteq S_1 \cup S_2} \{\mathrm{dp}_i(T) + w(T, (S_1 \cup S_2) \setminus T)\} \\
&= \max_{T_1 \subseteq S_1} \max_{T_2 \subseteq S_2} \{\mathrm{dp}_i(T_1 \cup T_2) + w(T_1 \cup T_2, (S_1 \cup S_2) \setminus (T_1 \cup T_2))\} \\
&= \max_{T_1 \subseteq S_1} \max_{T_2 \subseteq S_2} \{\mathrm{dp}_i(T_1 \cup T_2) + w(T_1, S_1 \cup S_2) + w(T_2, S_1 \cup S_2) \\
&\quad - w(T_1 \cup T_2, T_1 \cup T_2)\} \\
&= \max_{T_1 \subseteq S_1} \{w(T_1, S_1 \cup S_2) + \\
&\quad \max_{T_2 \subseteq S_2} \{\mathrm{dp}_i(T_1 \cup T_2) + w(T_2, S_1) - w(T_1 \cup T_2, T_1 \cup T_2) + w(T_2, S_2)\}\}.
\end{aligned}$$

We can reduce the computation of this recurrence into a product of the following two matrices X and Y with an indeterminate t.

$$\begin{aligned}
X_{(S_1,T_1),T_2} &= [T_1 \subseteq S_1]\, t^{\mathrm{dp}_i(T_1 \cup T_2) + w(T_2, S_1) - w(T_1 \cup T_2, T_1 \cup T_2)}, \\
Y_{T_2,S_2} &= [T_2 \subseteq S_2]\ t^{w(T_2,S_2)},
\end{aligned}$$

where $T_1, S_1 \subseteq V_1$ and $T_2, S_2 \subseteq V_2$. Also, $[\cdot]$ has value 1 if the condition inside is true and has value 0 otherwise. Indeed, the product XY can be expressed as

$$(XY)_{(S_1,T_1),S_2} = [T_1 \subseteq S_1] \sum_{T_2 \subseteq S_2} t^{\mathrm{dp}_i(T_1 \cup T_2) + w(T_2, S_1) - w(T_1 \cup T_2, T_1 \cup T_2) + w(T_2, S_2)}.$$

Thus, the value of $\mathrm{dp}_{i+1}(S_1 \cup S_2)$ coincides with the maximum degree of a non-zero term in

$$\begin{aligned}
&\sum_{T_1 \subseteq S_1} t^{w(T_1, S_1 \cup S_2)} (XY)_{(S_1,T_1),S_2} \\
&= \sum_{T_1 \subseteq S_1} t^{w(T_1, S_1 \cup S_2)} \sum_{T_2 \subseteq S_2} t^{\mathrm{dp}_i(T_1 \cup T_2) + w(T_2, S_1) - w(T_1 \cup T_2, T_1 \cup T_2) + w(T_2, S_2)} \\
&= \sum_{T \subseteq S_1 \cup S_2} t^{\mathrm{dp}_i(T) + w(T, S_1 \cup S_2 \setminus T)}.
\end{aligned}$$

The number of choices for S_1 and T_1 with $T_1 \subseteq S_1 \subseteq V_1$ is $3^{\frac{n}{1+\alpha}}$, and the number of choices for $T_2 \subseteq V_2$ is $2^{\frac{\alpha n}{1+\alpha}}$. By setting $\alpha = \log 3$, sizes of matrices X and Y become $3^{\frac{n}{\log 6}} \times 3^{\frac{n}{\log 6}}$, and we can multiply them in $O^*(3^{\frac{\omega}{\log 6}n})$ time. After the multiplication, we can compute the sum over T in $O^*(3^{\frac{2}{\log 6}n})$ time. In total, we can compute dp_{i+1} from dp_i in $O^*(3^{\frac{\omega}{\log 6}n})$ time and $O^*(3^{\frac{2}{\log 6}n})$ space, and thus we can compute $\mathrm{dp}_n(V)$ in the same time (up to a polynomial factor) and the same space. ◀

3.4 Utilizing Sparsity

Since Y_{T_2,S_2} in the previous subsection has non-zero value only when $T_2 \subseteq S_2$, the matrix Y has only $3^{|V_2|}$ non-zero entries. Moreover, by aligning indices of Y properly, we can see that Y is a *recursively partial matrix*, defined as follows.

▶ **Definition 3** (Recursively Partial Matrix)**.** Any matrix X with size 1×1 is a recursively partial matrix. A square matrix X with size 2^k is called recursively partial if when dividing it into four submatrices of size 2^{k-1}, the bottom left submatrix is a zero matrix and the other three submatrices are recursively partial matrices.

To exploit this structure when computing matrix products, we use the following theorem due to Bini et al. [3].

▶ **Theorem 4** ([3])**.** *We can compute an approximate product of a* 2×2 *matrix of the form* $\begin{bmatrix} a & b \\ 0 & c \end{bmatrix}$ *and any* 2×2 *matrix with 5 multiplications.*

Here, approximate product means that for any $\epsilon > 0$, we can compute the product with relative error depending on ϵ and it convergences to the exact value when $\epsilon \to 0$. Schönhage [14] observed that we can compute the exact matrix product by regarding ϵ as an indeterminate. This modification increases the running time by $O(\log n)$ factor. This type of matrix product, which computes the product of matrices containing zeros in a structured way, is called *partial matrix product*. The partial matrix product has been well studied for obtaining a faster algorithm for square matrix product, however, in our case, we can use the algorithm recursively to multiply a matrix X and a recursively partial matrix Y.

We give a detailed explanation of the algorithm. Let A be a $2^k \times 2^k$ matrix, B be a $2^k \times 2^k$ recursively partial matrix, and C be a product of A and B. We say that C' is a *d-approximate product* of C with an indeterminate ϵ if for each index (i, j), $C'_{i,j}$ can be written as $C'_{i,j} = C_{i,j} + \epsilon P_{i,j}(\epsilon)$ for some polynomial P of degree at most $d - 1$. Now, we compute the k-approximate product C' of C. We divide each matrix into four submatrices of size $2^{k-1} \times 2^{k-1}$. Then, the product of these two matrices is:

$$\begin{aligned} C_{1,1} &= A_{1,1}B_{1,1}, & C_{1,2} &= A_{1,1}B_{1,2} + A_{1,2}B_{2,2}, \\ C_{2,1} &= A_{2,1}B_{1,1}, & C_{2,2} &= A_{2,1}B_{1,2} + A_{2,2}B_{2,2}. \end{aligned}$$

We compute the $(k-1)$-approximate products of the followings by applying the algorithm recursively.

$$\begin{aligned} Z_1 &= (A_{2,1} + \epsilon A_{2,2})(B_{2,2} + \epsilon B_{1,2}), \\ Z_2 &= A_{1,1}(B_{1,1} + \epsilon B_{1,2}), \\ Z_3 &= A_{2,1}B_{2,2}, \\ Z_4 &= (A_{1,1} + A_{2,1} + \epsilon A_{1,2})B_{1,1}, \\ Z_5 &= (A_{2,1} + \epsilon A_{1,2})(B_{2,2} + B_{1,1}). \end{aligned}$$

Then, we can obtain k-approximate product C' from these products as follows:

$$\begin{aligned} C'_{1,1} &= Z_2, & C'_{1,2} &= \epsilon^{-1}(Z_2 - Z_3 - Z_4 + Z_5), \\ C'_{2,1} &= -Z_3 + Z_5, & C'_{2,2} &= \epsilon^{-1}(Z_1 - Z_3). \end{aligned}$$

Therefore, we can compute k-approximate product of two matrices in $O^*(5^k)$ time. Because the degree of each polynomial $P_{i,j}$ is at most $k - 1$, we can compute the exact product C by running the algorithm k times and using the interpolation.

By using this partial matrix product algorithm, we can compute the product XY in $O^*(5^{|V_2|}) = O^*(3^{\frac{\log 5}{\log 6}n})$ time. Recall that $|V_2| = \frac{\alpha n}{1+\alpha} = \frac{\log 3}{\log 6}n$ by our choice of $\alpha = \log 3$. As a result, we can solve Max-PA in $O^*(3^{\frac{\log 5}{\log 6}n})$ time and $O^*(3^{\frac{2}{\log 6}n})$ space.

4 Approximation Algorithms for Max-PA

In this section, we give a 0.4586-approximation algorithm for Max-PA. First, we reduce an instance of general Max-PA into an instance of Max-PA that has ($<$)-constraints and ($=$)-constraints only. This can be done by the following reduction.

- For each constraint of the form $(x \leq y)$, we replace it by two constraints $(x < y)$ and $(x = y)$.
- For each constraint of the form $(x \neq y)$, we replace it by two constraints $(x < y)$ and $(y < x)$.

The optimal value does not change through the reduction. This is because, for any assignment f, if the removed constraint is satisfied by f, then exactly one of the added two constraints is satisfied by f, and if the removed constraint is not satisfied by f, then none of the added two constraints is satisfied by f.

After applying the reduction, consider the following SDP:

$$\begin{aligned} \text{maximize} \quad & \sum_{(u=v)\in C} \langle x_u, x_v\rangle + \sum_{(u<v)\in C} (1 - \langle x_u, x_v\rangle), \\ \text{subject to} \quad & \|x_v\|^2 = 1 \quad (\forall v \in V), \\ & \langle x_u, x_v\rangle \geq 0 \quad (\forall u, v \in V), \end{aligned}$$

where x_v $(v \in V)$ is a real vector. To see that the SDP above is indeed a relaxation, let $f^* : V \to [n]$ be the optimal solution. Then, we set x_v be the vector whose i-th coordinate is 1 if $f^*(v) = i$ and 0 otherwise. Note that $\langle x_u, x_v\rangle = 1$ iff $f^*(u) = f^*(v)$. In particular, $1 - \langle x_u, x_v\rangle$ has value 1 when the constraint $(u < v)$ is satisfied. Therefore, the optimal SDP value gives the upper bound on the optimal value of the original Max-PA.

We can solve the SDP in polynomial time and let x^* be the optimal SDP solution. We create an assignment f by rounding x^* as follows. First, we generate k random n-dimensional unit vectors $y_1, \ldots, y_k$, where k is a parameter. Then, we partition the variable set V into 2^k groups according to the signs of k inner products $\langle x_v^*, y_i\rangle$. For each group, we assign a unique value to all variables in the group. Thus, we use at most 2^k different values in total. Finally, we introduce a random ordering among the values assigned to groups.

Now, we analyze the approximation ratio of this algorithm. A constraint $(u = v)$ is satisfied if u and v are in the same group. This event happens when they are in the same side of every hyperplane $\langle x, y_i\rangle = 0$. Thus, the probability that $(u = v)$ is satisfied is

$$\Pr[f(u) = f(v)] = \left(1 - \frac{\theta}{\pi}\right)^k,$$

where θ is the angle between two vectors x_u^* and x_v^*. In contrast, the constraint $(u = v)$ contributes to the SDP value by $\langle x_u^*, x_v^*\rangle = \cos\theta$.

A constraint $(u < v)$ is satisfied with probability $\frac{1}{2}$ if they are in different groups. Thus, the probability that the constraint is satisfied is

$$\frac{1}{2}\Pr[f(u) \neq f(v)] = \frac{1}{2}\left(1 - \left(1 - \frac{\theta}{\pi}\right)^k\right).$$

In contrast, the constraint $(u < v)$ contributes to the SDP value by $1 - \langle x_u^*, x_v^*\rangle = 1 - \cos\theta$.

The approximation ratio is a convex function of k and takes the maximum between $k = 3$ and $k = 4$. In order to take the balance, we run the algorithm by choosing $k = 3$ with probability α and $k = 4$ with probability $1 - \alpha$. Then, the approximation ratio is at least

$$\min\left\{\min_{\theta}\frac{\alpha(1-\frac{\theta}{\pi})^3+(1-\alpha)(1-\frac{\theta}{\pi})^4}{\cos\theta}, \min_{\theta}\frac{\alpha\left(1-(1-\frac{\theta}{\pi})^3\right)+(1-\alpha)\left(1-(1-\frac{\theta}{\pi})^4\right)}{2(1-\cos\theta)}\right\}.$$

By setting $\alpha = \frac{191}{593}$, the above value becomes $\frac{272}{593} > 0.4586$. The worst case is achieved when $\theta = \frac{\pi}{3}$ for $(=)$-constraints and $\theta = \frac{\pi}{2}$ for $(<)$-constraints.

5 Approximation Algorithms for Min-PA

In this section, we give an $O(\log n \log\log n)$-approximation algorithm for Min-PA. First, we reduce an instance of general Min-PA into an instance of Min-PA that has $(\leq)$-constraints and $(\neq)$-constraints only. This can be done by the following reduction.

- For each constraint of the form $(x = y)$, we replace it by two constraints $(x \leq y)$ and $(x \geq y)$.
- For each constraint of the form $(x < y)$, we replace it by two constraints $(x \leq y)$ and $(x \neq y)$.

The optimal value does not change through the reduction. This is because, for any assignment f, if the removed constraint is unsatisfied by f, then exactly one of the added two constraints is unsatisfied by f, and if the removed constraints is not unsatisfied, then none of the added two constraints is unsatisfied by f.

A sequence of variables $(v_1, \dots, v_k)$ is called a *$(\leq)$-path* from v_1 to v_k if a constraint $(v_i \leq v_{i+1})$ exists for every $i \in [k-1]$. Using the notion of $(\leq)$-path, we can obtain the following necessary and sufficient condition under which an instance is satisfiable.

▶ **Lemma 5** ([17]). *A CSP instance with $(\leq)$-constraints and $(\neq)$-constraints only is satisfiable if and only if, for any constraint $(x \neq y)$, we do not have a $(\leq)$-path from x to y and a $(\leq)$-path from y to x simultaneously.*

Due to Lemma 5, Min-PA with $(\leq)$-constraints and $(\neq)$-constraints only can be formulated as the following integer programming.

$$\begin{aligned} \text{minimize} \quad & \sum_{e\in C} x_e, \\ \text{subject to} \quad & x_e + \sum_{f\in P_{u,v}} x_f + \sum_{f\in P_{v,u}} x_f \geq 1 \quad \begin{pmatrix} \forall e = (u \neq v) \in C \\ \forall P_{u,v} : \text{a } (\leq)\text{-path from } u \text{ to } v \\ \forall P_{v,u} : \text{a } (\leq)\text{-path from } v \text{ to } u \end{pmatrix}, \qquad (3) \\ & x_e \in \{0,1\} \ (\forall e \in C). \end{aligned}$$

Here, $x_e = 1$ means that the constraint e is unsatisfied. Now we relax it to a linear programming by changing the last constraint to $x_e \geq 0$. The LP contains an exponential number of constraints. However, we can solve it in polynomial time by using the ellipsoid method since there is a polynomial-time separation oracle. More specifically, we construct a digraph as follows to check constraints (3). That is, For each constraint of the form $e = (u \leq v)$, we make an edge (u, v) of length x_e. Then, we check the corresponding constraints (3) is satisfied for each constraint of the form $e = (u = v)$. Here, $\sum_{f\in P_{u,v}} x_f$ (resp., $\sum_{f\in P_{u,v}} x_f$) can be computed as the length of the shortest path from u to v (resp., v to u) in the digraph.

Let x^* be the optimal solution to the LP. We create a new LP by using x^*. Let $e = (u \neq v)$ be a constraint, $P_{u,v}$ be a $(\leq)$-path from u to v and $P_{v,u}$ be a $(\leq)$-path from v to u. We first remove the corresponding constraint (3). If $x_e^* \geq \frac{1}{2}$, then we add a constraint $x_e \geq 1$ to the new LP. Otherwise and hence if $\sum_{f \in P_{u,v}} x_f^* + \sum_{f \in P_{v,u}} x_f^* \geq \frac{1}{2}$, then we add a constraint $\sum_{f \in P_{u,v}} x_f + \sum_{f \in P_{v,u}} x_f \geq 1$ to the new LP. The optimal value of the new LP is at most twice the optimal value of the original LP since $2x^*$ is a feasible solution to the new LP, and any feasible solution to the new LP is also a feasible solution to the original LP. Therefore, an integer solution to the new LP whose value is at most k times the optimal value of the new LP is a $2k$-approximation solution to the original problem.

The obtained problem can be considered as a variant of the directed multicut problem: given a digraph $G = (V, E)$ and a set of terminal pairs T, find an edge set $F \subseteq E$ of minimum cardinality such that for any terminal pair $(u, v) \in T$, $G - F$ contains no path from u to v or no path from v to u. We call this problem as the *symmetric multicut problem*. Also, we call a terminal pair (u, v) *separated* if there is no path from u to v or there is no path from v to u. We note that, in the standard multicut problem, a terminal pair (u, v) is also directed and we only care about deleting paths from u to v. We can obtain an instance of the symmetric multicut problem by setting $E = \{(u, v) \mid (u \leq v) \in C\}$, and $T = \{(u, v) \mid e = (u \neq v) \in C, x_e^* < \frac{1}{2}\}$.

To get approximation to the symmetric multicut problem, we consider the following LP relaxation:

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E} x_e, \\ \text{subject to} \quad & \sum_{f \in P_{u,v}} x_f + \sum_{f \in P_{v,u}} x_f \geq 1 \quad \begin{pmatrix} \forall (u, v) \in T \\ \forall P_{u,v} : \text{a path from } u \text{ to } v \\ \forall P_{v,u} : \text{a path from } v \text{ to } u \end{pmatrix}, \\ & x_e \geq 0 \ (\forall e \in E). \end{aligned} \tag{4}$$

Even et al. [8] showed an $O(\log n \log \log n)$-approximation algorithm for the (standard) multicut problem in some special kind of graphs, called *circular networks*. We use the same approach to solve the symmetric multicut problem. The following theorem immediately gives that the symmetric multicut problem and hence Min-PA can be approximated within a factor of $O(\log n \log \log n)$.

▶ **Theorem 6.** *The solution to the symmetric multicut problem whose value is at most* $O(\log n \log \log n)$ *times the optimal value of the LP* (4) *can be obtained in polynomial time.*

Proof. Let l be twice the optimal value of the LP (4). For an edge set $F \subseteq E$, we define $l(F) = \sum_{e \in F} l(e)$. We define the *length* of a path P as $l(P)$, and the *distance from u to v* as the length of the shortest path from u to v. For any terminal pair (s, t), the sum of distances from s to t and from t to s is at least 2.

We fix some terminal pair (s, t) and we assume that the distance from s to t is at least 1. Let $d(v)$ be the distance from s to v. For any $0 \leq x \leq 1$, we define edge sets $A(x)$, $L(x)$, and $B(x)$ as follows:

$$\begin{aligned} A(x) &= \{(u, v) \in E \mid d(u), d(v) \leq x\}, \\ L(x) &= \{(u, v) \in E \mid d(u) \leq x < d(v)\}, \\ B(x) &= \{(u, v) \in E \mid x < d(u), d(v)\}. \end{aligned}$$

Now we claim that there exists $0 \leq x \leq 1$ such that

$$|L(x)| \leq \mu(l(E)) - \mu(l(A(x))) - \mu(l(B(x))), \tag{5}$$

where $\mu(x) = 4x \ln(4x) \ln \log(4x)$. To show the claim, we use the following lemma by Seymour [15].

▶ **Lemma 7** ([15]). *Let $k > 0$ be a real number, let y be a real-valued monotone increasing function on $[0, 1]$ such that $y(0) \geq 0, y(1) \leq 1$ and for all $h \in [0, 1] - I$, where $I \subseteq [0, 1]$ is some finite subset of $[0, 1]$, y is differentiable and $\left.\frac{dy}{dx}\right|_{x=h} \geq \frac{1}{k}$. Then there exists h with $\frac{1}{4} < h < \frac{3}{4}, h \notin I$, such that*

$$k \left.\frac{dy}{dx}\right|_{x=h} \leq \mu(k) - \mu(ky(h)) - \mu(k(1 - y(h))).$$

This lemma is slightly different from the original one. In the original lemma, y is required to be contiguous, and this implies that y is monotone increasing because $\left.\frac{dy}{dx}\right|_{x=h} \geq \frac{1}{k}$. Actually, the same proof can be applied if y is not contiguous but monotone increasing.

We instantiate Lemma 7 with the following function y:

$$y(x) = \frac{1}{l(E)} \left(l(A(x)) + \sum_{e=(u,v) \in L(x)} (x - d(u)) \right).$$

Note that the function $y(x)$ is not contiguous at $x = d(v)$ for $v \in V$. An edge $e = (u, v)$ contributes to y by $\frac{1}{l(E)}(x-d(u))$ if $e \in L(x)$, and by $\frac{1}{l(E)} l(e)$ if $e \in A(x)$. Thus its contribution changes from $\frac{1}{l(E)}(d(v) - d(u))$ to $\frac{1}{l(E)} l(e)$ at $x = d(v)$. Because $d(v) - d(u) \leq l(e)$ holds for any edge $e = (u, v)$, its contribution to y never decreases. Therefore, the function y is monotone increasing. For any differentiable point $0 < x < 1$, the value of $\frac{dy}{dx}$ is $\frac{|L(x)|}{l(E)}$, which is at least $\frac{1}{l(E)}$ because $L(x) \neq \emptyset$. Therefore, there exists h such that:

$$\begin{aligned} |L(h)| &= l(E) \left.\frac{dy}{dx}\right|_{x=h} \\ &\leq \mu(l(E)) - \mu(l(E)y(h)) - \mu(l(E)(1 - y(h))) \\ &\leq \mu(l(E)) - \mu(l(A(h))) - \mu(l(B(h))), \end{aligned}$$

and the claim holds. Because there are essentially at most n choices for h, we can find it in polynomial time.

To obtain a good cut, we just find a real number x satisfying the condition (5) and then we remove the edge set $L(x)$. Let V_1 be the set of vertices whose distances from s are at most x and V_2 be the set of vertices whose distances from s are greater than x. Then, $L(x)$ contains only edges from V_1 to V_2 and does not contain any edge from V_2 to V_1. Nonetheless, since all paths from V_1 to V_2 are cut, every terminal pair (u, v) with $u \in V_1$ and $v \in V_2$ become separated. In particular, the terminal pair (s, t) is separated.

Thus, we can consider two graphs $G_1 = (V_1, A(x))$ and $G_2 = (V_2, B(x))$ separately, and we can recursively solve the symmetric multicut problem on G_1 and G_2 independently. We can show that the total number of removed edges in recursive steps is at most $\mu(l(E))$ by induction on $|E|$ because $|L(x)| + \mu(l(A(x))) + \mu(l(B(x))) \leq \mu(l(E))$. On the other hand, since l is twice the optimal LP solution, the optimal LP value is $\frac{1}{2}l(E)$. Thus we can obtain a solution whose value is at most $O(\log n \log \log n)$ times the optimal value of the LP (4). ◀

Acknowledgments

The authors are grateful to François Le Gall for valuable discussions about the usage of sparse matrix product for the exact algorithm.

References

1 J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

2 N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56(1):89–113, 2004.

3 D. Bini, M. Capovani, F. Romani, and G. Lotti. $O(n^{2.7799})$ complexity for $n*n$ approximate matrix multiplication. *Inf. Process. Lett.*, 8(5):234–235, 1979.

4 A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets Möbius: fast subset convolution. In *Proc. 39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 67–74, 2007.

5 M. Bodirsky and J. Kára. The complexity of temporal constraint satisfaction problems. *Journal of the ACM*, 57(2):1–41, 2010.

6 M. Charikar, V. Guruswami, and A. Wirth. Clustering with qualitative information. *Journal of Computer and System Sciences*, 71(3):360–383, 2005.

7 I. Dinur and S. Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, pages 439–485, 2005.

8 G. Even, J. Naor, B. Schieber, and M. Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.

9 V. Guruswami, J. Håstad, R. Manokaran, P. Raghavendra, and M. Charikar. Beating the random ordering is hard: Every ordering CSP is approximation resistant. *SIAM Journal on Computing*, 40(3):878–914, 2011.

10 V. Guruswami, R. Manokaran, and P. Raghavendra. Beating the random ordering is hard: Inapproximability of maximum acyclic subgraph. In *Proc. 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 573–582, 2008.

11 S. Khot. On the power of unique 2-prover 1-round games. In *Proc. 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 767–775, 2002.

12 J. Malik and T. Binford. Reasoning in time and space. In *Proc. 8th International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 343–345, 1983.

13 B. Nebel and H. Bürckert. Reasoning about temporal relations: a maximal tractable subclass of Allen's interval algebra. *Journal of the ACM*, 42(1):43–66, 1995.

14 A. Schönhage. Partial and total matrix multiplication. *SIAM J. Comput.*, 10(3):434–455, 1981.

15 P. D. Seymour. Packing directed circuits fractionally. *Combinatorica*, 15(2):281–288, 1995.

16 C. Swamy. Correlation clustering: maximizing agreements via semidefinite programming. In *Proc. 15th Annual ACM-SIAM Symposium on Discrete algorithms (SODA)*, pages 526–527, 2004.

17 P. van Beek and R. Cohen. Exact and approximate reasoning about temporal relations. *Computational Intelligence*, 6:132–144, 1990.

18 M. Vilain and H. Kautz. Constraint propagation algorithms for temporal reasoning. In *Proc. 5th National Conference on Artificial Intelligence*, pages 377–382, 1986.

19 R. Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005.

20 V. V. Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proc. 44th Annual ACM Symposium on Theory of Computing (STOC)*, pages 887–898, 2012.

Local Search is Better than Random Assignment for Bounded Occurrence Ordering k-CSPs

Konstantin Makarychev[1]

1 **Microsoft Research, Redmond, WA 98052**

Abstract

We prove that the Bounded Occurrence Ordering k-CSP Problem is not approximation resistant. We give a very simple local search algorithm that always performs better than the random assignment algorithm (unless, the number of satisfied constraints does not depend on the ordering). Specifically, the expected value of the solution returned by the algorithm is at least

$$\textsc{Alg} \geq \textsc{Avg} + \alpha(B,k)(\textsc{Opt} - \textsc{Avg}),$$

where $\textsc{Opt}$ is the value of the optimal solution; $\textsc{Avg}$ is the expected value of the random solution; and $\alpha(B,k) = \Omega_k(B^{-(k+O(1))})$ is a parameter depending only on k (the arity of the CSP) and B (the maximum number of times each variable is used in constraints).

The question whether bounded occurrence ordering k-CSPs are approximation resistant was raised by Håstad [6], who recently showed that bounded occurrence 3-CSPs and "monotone" k-CSPs admit a non-trivial approximation.

1998 ACM Subject Classification F.2.2 Computations on discrete structures

Keywords and phrases approximation algorithms, approximation resistance, ordering CSPs

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.139

1 Introduction

Overview. In this work, we give a very simple local search algorithm for ordering constraints satisfaction problems that works better than the random assignment for those instances of the ordering k-CSP problem, where each variable is used only a bounded number of times. To motivate the study of the problem, we first overview some known results for regular constraint satisfaction problems.

An instance of a constraint satisfaction problem consists of a set of variables $V = \{x_1, \dots, x_n\}$ taking values in a domain D and a set of constraints $\mathcal{C}$. Each constraint $C \in \mathcal{C}$ is a function from D^k to $\mathbb{R}^+$ applied to k variables from V. Given an instance of a CSP, our goal is to assign values to the variables to maximize the total payoff of all constraints:

$$\max_{x_1,\dots,x_n \in D^n} \sum_{C \in \mathcal{C}} C(x_1, \dots, x_n).$$

Note, that we write $C(x_1, \dots, x_n)$ just to simplify the notation. In fact, C may depend on at most k variables. The parameter k is called the arity of the CSP. In specific CSP problems, constraints C come from a specific family of constraints. For example, in Max Cut, the domain is $D = \{-1, 1\}$, and all constraints have the form $C(x_1, \dots, x_n) = \mathbf{1}(x_i \neq x_j)$; in Max 3LIN-2, the domain $D = \{0, 1\}$, and all constraints have the form $C(x_1, \dots, x_n) = \mathbf{1}(x_i \oplus x_j \oplus x_l = 0)$ or $C(x_1, \dots, x_n) = \mathbf{1}(x_i \oplus x_j \oplus x_l = 1)$.

Various approximation algorithms have been designed for CSPs. The most basic among them, the "trivial" probabilistic algorithm simply assigns random values to the variables.

It turns out, however, that in some cases this algorithm is essentially optimal. Håstad [8] showed that for some CSPs e.g., 3LIN-2 and E3-SAT, beating the approximation ratio of the random assignment algorithm (by any positive constant ε) is NP-hard. Such problems are called approximation resistant. That is, a constraint satisfaction problem is approximation resistant, if for every positive $\varepsilon > 0$, it is NP-hard to find a $(A_{trivial} + \varepsilon)$ approximation, where $A_{trivial}$ is the approximation ratio of the random assignment algorithm. If there exists an algorithm with the approximation ratio $(A_{trivial} + \varepsilon)$ for some positive ε, we say that the problem *admits a non-trivial approximation.* It is still not known which constraint satisfaction problems are approximation resistant and which admit a non-trivial approximation. This is an active research question in approximation algorithms.

Suppose now that in our instance of k-CSP, each variable is used by at most B constraints. (For example, for Max Cut, this means that the maximum degree of the graph is bounded by B.) Håstad [9] proved that such instances (which we call *B-bounded occurrence k*-CSPs) admit a non-trivial approximation. Let Opt denote the value of the optimal solution; Avg denote the expected value of the random assignment; and Alg denote the expected value returned by the algorithm. Håstad [9] showed that there exists an approximation algorithm such that[1]

$$\textsc{Alg} \geq \textsc{Avg} + \frac{\textsc{Opt} - \textsc{Avg}}{O_k(B)}.$$

Here the hidden constant in $O_k(\cdot)$ may depend on k. Trevisan [12] showed a hardness of approximation lower bound of $\textsc{Avg} + (\textsc{Opt} - \textsc{Avg})/(\Omega_k(\sqrt{B}))$.

In this work, we study *ordering* constraints satisfaction problems. A classical example of an ordering k-CSP is the Maximum Acyclic Subgraph problem. Given a directed graph $G = (V, E)$, the goal is to find an ordering of the vertices $\pi : V \to \{1, \dots, n\}$ (π is a bijection; $n = |V|$), so as to maximize the number of forward edges. In this case, the edges of the graph are constraints on the ordering π. An edge (u, v) corresponds to the constraint $\pi(u) < \pi(v)$. Another example is the Betweenness problem. We are given a set of vertices V and a set of constraints $\{C_{u,v,w}\}$. Each $C_{u,v,w}$ is defined as follows: $C_{u,v,w}(\pi) = 1$, if $u < v < w$ or $w < v < u$, and $C_{u,v,w}(\pi) = 0$, otherwise. The goal again is to find an ordering satisfying the maximum number of constraints.

More generally, in an ordering k-CSP, each constraint C is a function of the ordering that depends only on the relative order of k vertices. The goal is given a set of vertices V and a set of constraints $\mathcal{C}$, to find an ordering $\pi : V \to [n]$ to maximize the total value of all constraints:

$$\max_{\pi: V \to [n]} \sum_{C \in \mathcal{C}} C(\pi).$$

Here π is a bijection and $n = |V|$. If all constraints take values $\{0, 1\}$, then the objective is simply to maximize the number of satisfied constraints. Note, that an *ordering* k-CSP is not a k-CSP.

Surprisingly, we know more about ordering CSPs than about regular CSPs. Guruswami, Håstad, Manokaran, Raghavendra, and Charikar [4] showed that every ordering CSP problem is approximation resistant assuming the Unique Games Conjecture (special cases of this

[1] The quantity ("value of the solution" − Avg) is called the *advantage over random.* The algorithm of Håstad [9] is $O_k(B)$ approximation algorithm for the advantage over random:

$$\textsc{Alg} - \textsc{Avg} \geq \frac{\textsc{Opt} - \textsc{Avg}}{O_k(B)}.$$

result were obtained by Guruswami et al. [5] and Charikar et al. [2]). On the positive side, Berger and Shor [1] showed that bounded degree Maximum Acyclic Subgraph, and thus every bounded occurrence ordering 2CSP, admits a non-trivial approximation. Their result implies that $\textsc{Alg} \geq \textsc{Avg} + (\textsc{Opt} - \textsc{Avg})/O(\sqrt{B})$. Charikar, Makarychev, and Makarychev [3] showed that a slight advantage over the random assignment algorithm can be also achieved for instances of Maximum Acyclic Subgraph ($\textsc{Alg} \geq \textsc{Avg} + (\textsc{Opt} - \textsc{Avg})/O(\log n)$) whose maximum degree is not bounded. Gutin, van Iersel, Mnich, and Yeo [7] showed that the "advantage over the random assignment" for ordering 3CSPs is *fixed–parameter tractable* (we refer the reader to the paper for definitions and more details). Finally, Guruswami and Zhou [6] proved that all bounded occurrence ordering 3CSPs admit a non-trivial approximation ($\textsc{Alg} \geq \textsc{Avg} + (\textsc{Opt} - \textsc{Avg})/O_k(B)$). They also proved that there exists an approximation algorithm for *monotone* k-CSP (i.e., ordering CSPs, where all constraints are of the form $\pi(u_{i_1}) < \pi(u_{i_2}) < \cdots < \pi(u_{i_k})$) with approximation ratio $1/k! + 1/O_k(B)$.

Our results. We show that a very simple randomized local search algorithm finds a solution of expected value:

$$\textsc{Alg} \geq \textsc{Avg} + \frac{\textsc{Opt} - \textsc{Avg}}{O_k(B^{k+2})}. \tag{1}$$

This algorithm works for every bounded occurrence ordering k-CSP. Consequently, all bounded occurrence ordering k-CSPs admit a non-trivial approximation. The running time of the algorithm is $O(n \log n)$. We do not know whether the dependence on B is optimal. However, the result of Trevisan [12] implies a hardness of approximation upper bound of $\textsc{Alg} + (\textsc{Opt} - \textsc{Avg})/\Omega_k(\sqrt{B})^2$.

Techniques. Our algorithm works as follows: first, it permutes all vertices in a random order. Then, n times, it picks a random vertex and moves it to the optimal position without changing the positions of other vertices. We give an elementary proof that this algorithm performs better than the random assignment. However, the bound we get is exponentially small in B.

Then, we improve this bound. Roughly speaking, instead of the original problem we consider the "D-ordering" problem, where the algorithm puts vertices in $D \approx Bk$ buckets (possibly, in a clever way), then it randomly permutes vertices in each of the buckets, and finally outputs vertices in the first bucket, second bucket, etc. This idea was previously used by Charikar, Makarychev, and Makarychev [3], Guruswami, Håstad, Manokaran, Raghavendra, and Charikar [4], Gutin, van Iersel, Mnich, and Yeo [7] and Guruswami and Zhou [6]. The transition to "D-orderings" allows us to represent the payoff function as a Fourier series with relatively few terms. We prove that the L_1 weight of all coefficients of the payoff function is at least $\textsc{Avg} + \Omega_k(\textsc{Opt} - \textsc{Avg})$ (Note, that the optimal value of the "D-ordering" problem may be less than $\textsc{Opt}$). Then we show that (a) for each vertex we can find one "heavy" Fourier coefficient $\hat{f}_S$; and (b) when the original local search algorithm moves a vertex it increases the value of the solution in expectation by at least $\Omega_k(\hat{f}_S/B)$. This concludes the proof.

Correction. In the preliminary version of the paper that appeared at arXiv, we proved the main result of the paper, Theorem 1. We also gave an alternative, more complicated algorithm in the Appendix. We erroneously claimed that the performance guarantee of the alternative algorithm is slightly better than (1). This is not the case. So the best bound known to the author is (1).

[2] Every k-CSP can be encoded by an ordering $2k$-CSP by replacing every boolean variable x with two variables $u_x^{\leftarrow}$ and $u_x^{\rightarrow}$, and letting $x = 1$ if and only if $\pi(u_x^{\leftarrow}) < \pi(u_x^{\rightarrow})$.

2 Preliminaries

An instance of an ordering k-CSP problem $(V, \mathcal{C})$ consists of a set of vertices V of size n, and a set of constraints $\mathcal{C}$. An ordering of vertices $\pi : V \to \{1, \dots, n\}$ is a bijection from V to $\{1, \dots, n\}$. Each constraint $C \in \mathcal{C}$ is a function from the set of all ordering $\mathfrak{S}_V = \{\pi : V \to \{1, \dots, n\}\}$ to $\mathbb{R}^+$ that depends on the relative order of at most k vertices. That is, for every C there exists a set $T_C \subset V$ of size at most k such that if for two orderings π_1 and π_2, $\pi_1(u) < \pi_1(v) \Leftrightarrow \pi_2(u) < \pi_2(v)$ for all $u, v \in T_C$, then $C(\pi_1) = C(\pi_2)$. The value of an ordering π equals

$$\text{value}(\pi, \mathcal{C}) = \sum_{C \in \mathcal{C}} C(\pi).$$

We will sometimes write value$((u_1, \dots u_n), \mathcal{C})$ to denote the value$(\pi, \mathcal{C})$ for $\pi : u_i \mapsto i$. We denote the optimal value of the problem by $\textsc{Opt}(V, \mathcal{C}) \equiv \max_{\pi \in \mathfrak{S}_V} \text{value}(\pi, \mathcal{C})$, the average value — the value returned by the random assignment algorithm — by $\textsc{Avg}(V, \mathcal{C}) = 1/n! \sum_{\pi \in \mathfrak{S}_V} \text{value}(\pi, \mathcal{C})$.

3 Algorithm

We now present the algorithm.

Randomized Local Search Algorithm

Input: a set of vertices V, and a set of constraints $\mathcal{C}$.
Output: an ordering of vertices $(v_1, \dots, v_n)$.

```
1. Randomly permute all vertices.
2. Repeat $n$ times:
   - Pick a random vertex $u$ in $V$.
   - Remove $u$ from the ordering and insert it at a new location to maximize the payoff.
     I.e., if $v_1, \dots, v_{n-1}$ is the current ordering of all vertices but the vertex $u$, then find a
     location $i$ that maximizes the value$(v_1, \dots, v_{i-1}, u, v_{i+1}, \dots v_{n-1}, \mathcal{C})$, and put $u$ in the
     $i$-th position.
3. Return the obtained ordering.
```

▶ **Theorem 1.** *Given an instance $(V, \mathcal{C})$ of a B-bounded occurrence ordering k-CSP problem, the Randomized Local Search Algorithm returns a solution π_{ALG} of expected value*

$$\mathbb{E}\, \text{value}(\pi_{\text{ALG}}, \mathcal{C}) \geq \textsc{Avg}(V, \mathcal{C}) + \frac{\textsc{Opt}(V, \mathcal{C}) - \textsc{Avg}(V, \mathcal{C})}{O_k(B^{k+2})}. \tag{2}$$

▶ Remark. In fact, our proof implies a slightly stronger bound: The second term on the right hand side of the inequality (2) can be replaced with $(\textsc{Opt}(V, \mathcal{C}) - \textsc{Worst}(V, \mathcal{C}))/O_k(B^{k+2})$, where $\textsc{Worst}(V, \mathcal{C})$ is the value of the worst possible solution.

Proof. I. We first show using an elementary argument that

$$\mathbb{E}\, \text{value}(\pi_{\text{ALG}}, \mathcal{C}) \geq \textsc{Avg}(V, \mathcal{C}) + \alpha(B, k)(\textsc{Opt}(V, \mathcal{C}) - \textsc{Avg}(V, \mathcal{C})),$$

for some function $\alpha(B,k)$ depending only on B and k. This immediately implies that every bounded occurrence ordering k-CSP admits a non-trivial approximation. Then, using a slightly more involved argument we prove the bound (2).

Observe, that the expected value of the solution after step 1 is exactly equal to $\textsc{Avg}(V,\mathcal{C})$. So we need to estimate how much local moves at step 2 improve the solution. Let Δ_u be the maximum possible increase in the value of an ordering π, when we move u to another position. In other words, $\Delta_u = \max_{\pi^+,\pi^-}(\text{value}(\pi^+,\mathcal{C}) - \text{value}(\pi^-,\mathcal{C}))$, where the orderings π^+ and π^- differ only in the position of the vertex u. Let π^* be the optimal ordering, and π_* be the worst possible ordering. We can transition from π^* to π_* by moving every vertex u at most once. Thus,

$$\sum_{u\in V}\Delta_u \geq \text{value}(\pi^*,\mathcal{C}) - \text{value}(\pi_*,\mathcal{C}) = \textsc{Opt}(V,\mathcal{C}) - \textsc{Worst}(V,\mathcal{C}) \geq \textsc{Opt}(V,\mathcal{C}) - \textsc{Avg}(V,\mathcal{C}).$$

Now, our goal is to show that when the algorithm moves a vertex u, the value of the solution increases in expectation by at least $\alpha(B,k)\Delta_u$ for some function α depending only on B and k.

Fix a vertex u. Let π^+ and π^- be the orderings that differ only in the position of the vertex u such that $\Delta_u = \text{value}(\pi^+,\mathcal{C}) - \text{value}(\pi^-,\mathcal{C})$. It may happen that the random permutation chosen by the algorithm at step 1 is π^-, and u is chosen first among all vertices in V at step 2. In this case, the algorithm can obtain the permutation π^+ by moving u, and thus it can increase the value of the solution by Δ_u. However, the probability of such event is negligible. It is $1/n \cdot 1/n!$. The main observation is that the increase in the value of the ordering, when we move u, depends only on the order of the *neighbors* of u i.e., those vertices that share at least one common constraint $C \in \mathcal{C}$ with u (including u itself). We denote the set of neighbors by $N(u)$. Since each vertex participates in at most B constraints, and every constraint depends on at most k variables, $|N(u)| \leq kB$.

Consider an execution of the algorithm. We say that u is *fresh* if u was chosen at least once in the "repeat" loop of the algorithm, and none of the neighbors were chosen before u was chosen the first time. The probability that a variable u is fresh is at least $1/2|N(u)|^{-1}$. Indeed, the probability that at least one vertex in $N(u)$ is chosen is $1-(1-|N(u)|/n)^n > 1 - 1/e$; the probability that the first vertex chosen in $N(u)$ is u is $1/|N(u)|$ (since all vertices in $N(u)$ have the same probability of being chosen first).

If u is fresh, then when it is chosen, its neighbors are located in a random order (since none of them was moved by the algorithm). Thus, with probability $1/N(u)! \geq 1/(kB)!$, the order of neighbors of u is the same as in π^-. Then, by moving u we can increase the value of the ordering by Δ_u.

Therefore, when the algorithm moves the vertex u, the value of the ordering increases in expectation by at least

$$\Pr(u \text{ is fresh}) \cdot \Pr(N(u) \text{ is ordered as } \pi^- \text{ after step 1}) \cdot \Delta_u = \frac{\Delta_u}{2|N(u)|\,|N(u)|!} \geq \frac{\Delta_u}{kB\,(kB)!}.$$

This finishes the elementary proof that a positive $\alpha(B,k)$ exists.

II. We now improve the lower bound on $\alpha(B,k)$. We show that for a fresh variable u, the value of the ordering increases in expectation by at least $\Omega_k(B^{-(k+1)})\Delta_u$, and thus (2) holds. Let $L = \lceil \log_2(|N(u)|+1)\rceil$ and $D = 2^L$. Consider D buckets $[D] = \{1,\ldots,D\}$. For every mapping x of the vertices to the buckets $v \mapsto x_v \in [D]$, we define a distribution $\mathcal{U}_x$ on orderings of V. A random ordering from $\mathcal{U}_x$ is generated as follows: put each vertex v in the bucket x_v; then randomly and uniformly permute vertices in each bucket; and finally output vertices in the first bucket, second bucket, etc (according to their order in those buckets). Let $\mathcal{C}_u$ be the set of constraints that depend on the vertex u. Since every variable participates in at most B constraints, $|\mathcal{C}_u| \leq B$. Let $f(x)$ be the expected total value of constraints in $\mathcal{C}_u$ on a random ordering π sampled from the distribution $\mathcal{U}_x$:

$$f(x) = \mathbb{E}_{\pi \sim \mathcal{U}_x} \Big[\sum_{C \in \mathcal{C}_u} C(\pi) \Big].$$

Since the number of buckets D is greater than or equals to $|N(u)| + 1$, we may put every vertex in $N(u)$ in its own bucket and keep one bucket empty. Let π^+ and π^- be the orderings as in part I of the proof: π^+ and π^- differ only in the position of the vertex u, and $\text{value}(\pi^+, \mathcal{C}) - \text{value}(\pi^-, \mathcal{C}) = \Delta_u$. Consider mappings $x^+ : V \to [D]$ and $x^- : V \to [D]$ that put only one vertex from $N(u)$ in every bucket and such that $x^+_v = x^-_v$ for every $v \neq u$, and x^+ orders vertices in $N(u)$ according to π^+, x^- orders vertices in $N(u)$ according to π^-. For example, if π^+ arranges vertices in the order (a, b, u, c), and π^- arranges vertices in the order (a, u, b, c), then $x^+ = (a \mapsto 1, *, b \mapsto 3, u \mapsto 4, c \mapsto 5)$ and $x^- = (a \mapsto 1, u \mapsto 2, b \mapsto 3, *, c \mapsto 5)$. Since the order of all vertices in $N(u)$ is fixed by x^+ and x^-, we have $f(x^+) = \text{value}(\pi^+, \mathcal{C}_u)$ and $f(x^-) = \text{value}(\pi^-, \mathcal{C}_u)$. Then

$$\begin{aligned} f(x^+) - f(x^-) &= \text{value}(\pi^+, \mathcal{C}_u) - \text{value}(\pi^-, \mathcal{C}_u) \\ &= \text{value}(\pi^+, \mathcal{C}) - \text{value}(\pi^-, \mathcal{C}) = \Delta_u. \end{aligned}$$

We now use Theorem 2, which we prove in Section 4. Let X_v (for $v \in V$) be independent random variables uniformly distributed in $[D]$. By Theorem 2,

$$\begin{aligned} \mathbb{E}[\max_{x_u \in D} f(x_u, \{X_v\}_{v \neq u}) - f(X_u, \{X_v\}_{v \neq u})] &\geq \Omega_k(B^{-1} D^{-k})(f(x^+) - f(x^-)) \\ &= \Omega_k(B^{-(k+1)}) \Delta_u. \end{aligned}$$

Here, $(x_u, \{X_v\}_{v \neq u})$ denotes the mapping $u \mapsto x_u$ and $v \mapsto X_v$ for $v \neq u$; and $(X_u, \{X_v\}_{v \neq u})$ denotes the mapping $v \mapsto X_v$ for all v.

Observe, that when we sample random variables X_v, and then sample π according to $\mathcal{U}_X$, we get a random uniform ordering π of all vertices in V. Thus,

$$\mathbb{E}[f(X_u, \{X_v\}_{v \neq u})] = \mathbb{E}_{\pi \in \mathfrak{S}_V} \Big[\sum_{C \in \mathcal{C}_u} C(\pi) \Big] = \mathbb{E}_\pi[\text{value}(\pi, \mathcal{C}_u)].$$

Similarly, when we sample random variables X_v, set $x_u = \text{argmax}_{x_u \in D} f(x_u, \{X_v\}_{v \neq u})$, and then sample π' according to $\mathcal{U}_{(x_u, \{X_v\}_{v \neq u})}$, we get a random uniform ordering of all vertices except for the vertex u. Denote by $LS(\pi, u)$ the ordering obtained from the ordering π by moving the vertex u to the optimal position. It is easy to see that if π is a random uniform ordering, then $LS(\pi, u)$ has the same distribution as $LS(\pi', u)$, since the new optimal position of u depends only on the relative order of other vertices v, and not on the old position of u. Hence,

$$\begin{aligned} \mathbb{E}[\max_{x_u \in D} f(x_u, \{X_v\}_{v \neq u})] &\equiv \mathbb{E}_{\pi'}[\text{value}(\pi', \mathcal{C}_u)] \\ &\leq \mathbb{E}_{\pi'}[\text{value}(LS(\pi', u), \mathcal{C}_u)] \\ &= \mathbb{E}_\pi[\text{value}(LS(\pi, u), \mathcal{C}_u)]. \end{aligned}$$

Hence,

$$\begin{aligned} \mathbb{E}_\pi[\text{value}(LS(\pi, u), \mathcal{C}) - \text{value}(\pi, u, \mathcal{C})] &= \mathbb{E}_\pi[\text{value}(LS(\pi, u), \mathcal{C}_u) - \text{value}(\pi, u, \mathcal{C}_u)] \\ &\geq \Omega_k(B^{-(k+1)}) \Delta_u. \end{aligned}$$

◀

4 Theorem 2

▶ **Theorem 2.** *Let D be a set of size 2^L (for some L). Consider a function $f : D^{n+1} \to \mathbb{R}$ that can be represented as a sum of T functions $f_t : D^{n+1} \to \mathbb{R}$:*

$$f(x_0, x_1, \dots, x_n) = \sum_{t=1}^{T} f_t(x_0, x_1, \dots, x_n)$$

such that each function f_t depends on at most k variables x_u. Here, $x_0, \dots, x_n \in D$. Then, the following inequality holds for random variables $X_0, \dots, X_n$ uniformly and independently distributed in D:

$$\mathbb{E}[\max_{x \in D} f(x, X_1, \dots, X_n) - f(X_0, X_1, \dots, X_n)] \geq$$
$$\Omega_k(T^{-1}|D|^{-k}) \max_{x^+, x^-, x_1, \dots, x_n \in D} (f(x^+, x_1, \dots, x_n) - f(x^-, x_1, \dots, x_n)).$$

▶ Remark. The variable x_0 corresponds to x_u from the proof of Theorem 1. The functions $f_t(x)$ are equal to $\mathbb{E}_{\pi \sim \mathcal{U}_x} C_t(\pi)$, where C_t is the t-th constraint from $\mathcal{C}_u$.

Proof. Without loss of generality we assume that elements of D are vertices of the boolean cube $\{-1, 1\}^L$. We denote the i-th coordinate of $x \in D$ by $x(i)$. We now treat f as a function of $(n+1)L$ boolean variables $x_u(i)$. We write the Fourier series of the function f. The Fourier basis consists of functions

$$\chi_S(x_0, \dots, x_n) = \prod_{(u,i) \in S} x_u(i),$$

which are called *characters*. Each index $S \subset \{0, 1, \dots, n\} \times \{1, \dots, L\}$ corresponds to the set of boolean variables $\{x_u(i) : (u, i) \in S\}$. Note, that $\chi_\varnothing(x_0, \dots, x_n) = 1$. The Fourier coefficients of f equal

$$\hat{f}_S = \mathbb{E}[f(X_0, \dots, X_n)\, \chi_S(X_0, \dots, X_n)],$$

and the function f equals

$$f(x_0, \dots, x_n) = \sum_S \hat{f}_S\, \chi_S(x_0, \dots, x_n).$$

▶ Remark. In the proof, we only use the very basic facts about the Fourier transform. The main property we need is that the characters form an orthonormal basis, that is,

$$\mathbb{E}[\chi_{S_1}(X_0, \dots, X_n)\chi_{S_2}(X_0, \dots, X_n)] = \begin{cases} 1, & \text{if } S_1 = S_2; \\ 0, & \text{if } S_1 \neq S_2. \end{cases}$$

Particularly, for $S \neq \varnothing$,

$$\mathbb{E}[\chi_S(X_0, \dots, X_n)] = \mathbb{E}[\chi_S(X_0, \dots, X_n)\chi_\varnothing(X_0, \dots, X_n)] = 0.$$

We will also need the following property: if f does not depend on the variable $x_u(i)$, then all Fourier coefficients $\hat{f}_S$ with $(u, i) \in S$ are equal to 0.

Here is a brief overview of the proof: We will show that the L_1 weight of Fourier coefficients of f is at least $f(x^+, x_1, \dots, x_n) - f(x^-, x_1, \dots, x_n)$, and the weight of one of the coefficients $\hat{f}_{S^*}$ is at least $\Omega(T^{-1}|D|^{-k}(f(x^+, x_1, \dots, x_n) - f(x^-, x_1, \dots, x_n)))$. Consequently, if we flip a single bit $X_0(i^*)$ in X_0 to make the term $\hat{f}_{S^*}\chi_{S^*}(X_0', X_1, \dots, X_n)$ positive, we will increase the expected value of f by $|\hat{f}_{S^*}|$.

Observe, that since each function f_t depends on at most kL boolean variables, it has at most $2^{kL} = |D|^k$ nonzero Fourier coefficients. Thus, f has at most $T\,|D|^k$ nonzero Fourier coefficients $\hat{f}_S$.

Pick $x^+, x^-, x_1^*, \dots, x_n^*$ that maximize $f(x^+, x_1^*, \dots, x_n^*) - f(x^-, x_1^*, \dots, x_n^*)$. We have

$$f(x^+, x_1^*, \dots, x_n^*) - f(x^-, x_1^*, \dots, x_n^*) = \sum_S \hat{f}_S(\chi_S(x^+, x_1^*, \dots, x_n^*) - \chi_S(x^-, x_1^*, \dots, x_n^*)).$$

If S does not contain pairs $(0, i)$ corresponding to the bits of the variable x_0, then the character $\chi_S(x_0, x_1, \dots, x_n)$ does not depend on x_0, and $\chi_S(x^+, x_1^*, \dots, x_n^*) - \chi_S(x^-, x_1^*, \dots, x_n^*) = 0$, hence

$$\begin{aligned} &f(x^+, x_1^*, \dots, x_n^*) - f(x^-, x_1^*, \dots, x_n^*) = \\ &= \sum_{S:\exists i \text{ s.t. } (0,i)\in S} \hat{f}_S \cdot (\chi_S(x^+, x_1^*, \dots, x_n^*) - \chi_S(x^-, x_1^*, \dots, x_n^*)) \leq 2 \sum_{S:\exists i \text{ s.t. } (0,i)\in S} |\hat{f}_S|. \end{aligned}$$

Pick a character $\hat{f}_{S^*}$ with maximum absolute value and pick one of the elements $(0, i^*) \in S^*$. Since the number of nonzero characters $\hat{f}_S$ is at most $T\,|D|^k$,

$$|\hat{f}_{S^*}| \geq \frac{f(x^+, x_1^*, \dots, x_n^*) - f(x^-, x_1^*, \dots, x_n^*)}{2T\,|D|^k}.$$

Let $\sigma = \operatorname{sgn}(\hat{f}_{S^*})$. Define a new random variable X_0' on the same probability space as $X_0, \dots, X_n$,

$$X_0'(i) = \begin{cases} X_0(i), & \text{for } i \neq i^*; \\ \sigma\,\chi_{S^*}(X_0, \dots, X_n)\,X_0(i), & \text{for } i = i^*. \end{cases}$$

Consider a character χ_S. If $(0, i^*) \notin S$, then χ_S does not depend on the bit $x_0(i^*)$, hence $\mathbb{E}[\chi_S(X_0', X_1, \dots, X_n)] = \mathbb{E}[\chi_S(X_0, X_1, \dots, X_n)]$. On the other hand, if $(0, i^*) \in S$, then

$$\begin{aligned} \mathbb{E}[\chi_S(X_0', X_1, \dots, X_n)] &= \mathbb{E}\Big[\frac{X_0'(i^*)}{X_0(i^*)}\,\chi_S(X_0, X_1, \dots, X_n)\Big] = \\ \mathbb{E}[\sigma\chi_{S^*}(X_0, X_1, \dots, X_n)\chi_S(X_0, X_1, \dots, X_n)] &= \begin{cases} 0, & \text{if } S \neq S^*; \\ \sigma, & \text{if } S = S^*. \end{cases} \end{aligned}$$

The last equality holds because characters χ_S form an orthonormal basis. Therefore,

$$\mathbb{E}[f(X_0', X_1, \dots, X_n) - f(X_0, X_1, \dots, X_n)] = \sigma\hat{f}_{S^*} = |\hat{f}_{S^*}|.$$

We get

$$\begin{aligned} &\mathbb{E}[\max_{x\in D} f(x, X_1, \dots, X_n) - f(X_0, X_1, \dots, X_n)] \geq \mathbb{E}[f(X_0', X_1, \dots, X_n) - f(X_0, X_1, \dots, X_n)] \\ &= |\hat{f}_{S^*}| \geq \Omega_k(T^{-1}|D|^{-k}) \max_{x^*, x_*, x_1, \dots, x_n \in D} (f(x^+, x_1, \dots, x_n) - f(x^-, x_1, \dots, x_n)). \end{aligned}$$

◀

5 Concluding remarks

We can guarantee that the algorithm finds a solution of value (2) with high probability by repeating the algorithm $\Theta_k(B^{k+2})$ times (since the maxim possible value of the solution is Opt).

We note that our local search algorithm works not only for ordering k-CSPs, but also for (regular) k-CSPs. The algorithm first assigns random values to all variable x_i, and then, n times, picks a random $i \in \{1, \dots, n\}$, and changes the value of the variable x_i to the optimal value for fixed other variables. The approximation guarantee of the algorithm is $\textsc{Avg}(V, \mathcal{C}) + (\textsc{Opt}(V, \mathcal{C}) - \textsc{Avg}(V, \mathcal{C}))/O_{k,D}(B)$, here k is the arity, and D is the domain size of the CSP. The approximation guarantee has the same dependence on B as the approximation guarantee of Håstad's [9] original algorithm. The analysis relies on Theorem 2.

References

1 B. Berger and P. Shor. Approximation Algorithms for the Maximum Acyclic Subgraph Problem. *SODA* 1990.

2 M. Charikar, V. Guruswami, and R. Manokaran. Every Permutation CSP of arity 3 is Approximation Resistant. *IEEE Conference on Computational Complexity* 2009.

3 M. Charikar, K. Makarychev, and Y. Makarychev. On the Advantage over Random for Maximum Acyclic Subgraph. *FOCS* 2007.

4 V. Guruswami, J. Håstad, R. Manokaran, P. Raghavendra, and M. Charikar. Beating the Random Ordering Is Hard: Every Ordering CSP Is Approximation Resistant. *SIAM J. Comput. 40(3)* 2011.

5 V. Guruswami, R. Manokaran, and P. Raghavendra. Beating the Random Ordering is Hard: Inapproximability of Maximum Acyclic Subgraph. *FOCS* 2008.

6 V. Guruswami and Y. Zhou. Approximating Bounded Occurrence Ordering CSPs. *APPROX-RANDOM* 2012.

7 G. Gutin, L. van Iersel, M. Mnich and A. Yeo. Every ternary permutation constraint satisfaction problem parameterized above average has a kernel with a quadratic number of variables. *J. Comput. Syst. Sci., vol 78 (1)* 2012.

8 J. Håstad. Some optimal inapproximability results. *STOC* 1997.

9 J. Håstad. On bounded occurrence constraint satisfaction. *Inf. Process. Lett. (IPL) 74(1-2):1-6* 2000.

10 J. Håstad. Every 2-CSP allows nontrivial approximation. *STOC* 2005.

11 A. Newman. The Maximum Acyclic Subgraph Problem and Degree-3 Graphs. *RANDOM-APPROX* 2001.

12 L. Trevisan. Non-approximability results for optimization problems on bounded degree instances. *STOC* 2001.

The complexity of approximating conservative counting CSPs*

Xi Chen[1], Martin Dyer[2], Leslie Ann Goldberg[3], Mark Jerrum[4], Pinyan Lu[5], Colin McQuillan[3], and David Richerby[3]

1 **Dept. of Comp. Sci., Columbia University, 450 Comp. Sci. Building, 1214 Amsterdam Avenue, Mailcode: 0401, New York, NY 10027-7003, USA.**
2 **School of Computing, University of Leeds, Leeds, LS2 9JT, UK.**
3 **Department of Computer Science, University of Liverpool, Liverpool, L69 3BX, UK.**
4 **School of Mathematical Sciences, Queen Mary, University of London, Mile End Road, London, E1 4NS, UK.**
5 **Microsoft Research Asia, Microsoft Shanghai Technology Park, No 999, Zixing Road, Minhang District, Shanghai, 200241, China.**

Abstract

We study the complexity of approximation for a weighted counting constraint satisfaction problem #CSP($\mathcal{F}$). In the conservative case, where $\mathcal{F}$ contains all unary functions, a classification is known for the Boolean domain. We give a classification for problems with general finite domain. We define *weak log-modularity* and *weak log-supermodularity*, and show that #CSP($\mathcal{F}$) is in FP if $\mathcal{F}$ is weakly log-modular. Otherwise, it is at least as hard to approximate as #BIS, counting independent sets in bipartite graphs, which is believed to be intractable. We further sub-divide the #BIS-hard case. If $\mathcal{F}$ is weakly log-supermodular, we show that #CSP($\mathcal{F}$) is as easy as Boolean log-supermodular weighted #CSP. Otherwise, it is NP-hard to approximate. Finally, we give a trichotomy for the arity-2 case. Then, #CSP($\mathcal{F}$) is in FP, is #BIS-equivalent, or is equivalent to #SAT, the problem of approximately counting satisfying assignments of a CNF Boolean formula.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases counting constraint satisfaction problem, approximation, complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.148

1 Introduction

A weighted counting constraint satisfaction problem has a fixed finite domain D and a fixed finite "weighted constraint language" $\mathcal{F}$, a set of functions. Every $F \in \mathcal{F}$ maps a tuple of domain elements to a value called a "weight". In the computational problem #CSP($\mathcal{F}$), an instance consists of a set $V = \{v_1, \dots, v_n\}$ of variables and a set of "weighted constraints". A weighted constraint applies a function from $\mathcal{F}$ to an appropriate-sized tuple of variables.

For example, with Boolean domain $D = \{0, 1\}$, $\mathcal{F}$ might contain a single binary (arity-2) function F defined by $F(0,0) = F(0,1) = F(1,0) = 1$ and $F(1,1) = 2$. An instance might have variables v_1, v_2, v_3 and weighted constraints $F(v_1, v_2)$, $F(v_2, v_3)$. If $\mathbf{x} = (x_1, x_2, x_3)$ is an assignment of domain elements to the variables, the total weight associated with $\mathbf{x}$ is the

* This work was supported by the EPSRC Research Grant "Computational Counting". Xi Chen was also supported by NSF Grant CCF-1139915 and Columbia University start-up funds. Full version: [11].

product of the weighted constraints, evaluated at $\mathbf{x}$. The computational problem is to evaluate the sum of weights of all assignments. In our example, this is $\sum_{\mathbf{x} \in \{0,1\}^3} F(x_1, x_2)F(x_2, x_3) = 1+1+1+2+1+1+2+4 = 13$.

There has been a lot of work on classifying the computational difficulty of exactly solving $\#\mathrm{CSP}(\mathcal{F})$. For some $\mathcal{F}$, this task is computationally easy; for others it is intractable. We briefly summarise what is known: see the surveys of Chen [10] and Lu [21] for more detail.

First, suppose the domain D is Boolean (that is, $D = \{0, 1\}$). Creignou and Hermann [14] showed that, when the weights also lie in $\{0, 1\}$, $\#\mathrm{CSP}(\mathcal{F})$ is in FP (functions computable in polynomial time) if all $F \in \mathcal{F}$ are affine but, otherwise, it is #P-complete. Dyer, Goldberg, and Jerrum [16] extended this dichotomy to non-negative rational weights, showing that the problem is polynomial-time solvable if (1) every $F \in \mathcal{F}$ is expressible as a product of unary functions, equalities and disequalities, or (2) every $F \in \mathcal{F}$ is a constant multiple of an affine function. Otherwise, the problem is complete in the complexity class $\mathrm{FP}^{\#\mathrm{P}}$. We do not deal with negative weights but the above results have been extended to these [4] and also to complex [9] weights. A dichotomy exists for the related Holant* problem [8] (see also [21]).

Next, consider arbitrary finite D. For $\{0, 1\}$ weights, Bulatov's breakthrough [2] showed that $\#\mathrm{CSP}(\mathcal{F})$ is always either in FP or #P-hard. This was simplified by Dyer and Richerby [18], using a new criterion called "strong balance", and extended to non-negative rational weights by Bulatov et al. [3] and to all non-negative algebraic weights by Cai, Chen and Lu [7], using a generalised notion of balance that we use here. Finally, Cai and Chen [6] extended the dichotomy to complex algebraic weights. The criteria for the above dichotomies are known to be decidable [7, 18], except for the complex case, which remains open.

Less is known about the complexity of approximation for $\#\mathrm{CSP}(\mathcal{F})$. The complexity of approximate counting within #P was studied by Dyer, Goldberg, Greenhill and Jerrum [15], who identified three complexity classes for approximation problems within #P:

1. the class of problems with a fully-polynomial randomised approximation scheme (FPRAS),
2. a logically-defined complexity class called $\#\mathrm{RH}\Pi_1$, and
3. a class of problems for which approximation is NP-hard.

A typical complete problem in $\#\mathrm{RH}\Pi_1$ is #BIS, approximately counting independent sets in bipartite graphs. Either all complete problems in $\#\mathrm{RH}\Pi_1$ have an FPRAS, or none do; it is conjectured that none do [19]. #SAT, counting satisfying assignments of CNF Boolean formulas, is complete in class 3. Another important concept in the classification of approximate counting CSPs is log-supermodularity [5]. A function with Boolean domain is log-supermodular if its logarithm is supermodular; we give a formal definition later.

Over the Boolean domain, Dyer, Goldberg and Jerrum [17] gave a trichotomy for the complexity of approximately solving #CSP for $\{0, 1\}$ weights. If every $F \in \mathcal{F}$ is affine, then $\#\mathrm{CSP}(\mathcal{F})$ is in FP. Otherwise, it is at least as hard to approximate as #BIS. The hard approximation problems can be divided into #BIS-equivalent cases and #SAT-equivalent cases. When the domain D is Boolean, but arbitrary non-negative weights are allowed, no complete classification is known. However, Bulatov *et al.* [5] gave a classification for the so-called "conservative" case, where $\mathcal{F}$ contains all unary functions. Informally,

- if every function in $\mathcal{F}$ can be expressed in a certain simple way using disequality and unary functions, then, for any finite $\mathcal{G} \subset \mathcal{F}$, $\#\mathrm{CSP}(\mathcal{G})$ has an FPRAS;
- otherwise,
  - for some finite $\mathcal{G} \subset \mathcal{F}$, $\#\mathrm{CSP}(\mathcal{G})$ is at least as hard to approximate as #BIS and,
  - if $\mathcal{F}$ contains any function that is not log-supermodular, then there is a finite $\mathcal{G} \subset \mathcal{F}$ such that $\#\mathrm{CSP}(\mathcal{G})$ is as hard to approximate as #SAT.

Yamakami [26] gave an approximation dichotomy when further unary functions are in $\mathcal{F}$ (including those with negative values). The negative weights cause more constraint languages to become intractable [26]. Here, we allow only non-negative weights, since more subtle complexity classifications arise.

Prior to this paper, no complexity classification was known for approximation of #CSP($\mathcal{F}$) when the domain is not Boolean. This is the problem that we address. Our main result, Theorem 2, is a classification for the conservative case (where all unary functions are in $\mathcal{F}$). An informal description is given below. The technical concepts are defined in §1.1.

- If $\mathcal{F}$ is "weakly log-modular" then, for any finite $\mathcal{G} \subset \mathcal{F}$, #CSP($\mathcal{G}$) is in FP.
- Otherwise, for some finite $\mathcal{G} \subset \mathcal{F}$, #CSP($\mathcal{G}$) is at least as hard to approximate as #BIS, and
  - if $\mathcal{F}$ is "weakly log-supermodular" then, for any finite $\mathcal{G} \subset \mathcal{F}$, there is a finite set $\mathcal{G}'$ of log-supermodular functions on the Boolean domain such that #CSP($\mathcal{G}$) is at least as easy to approximate as #CSP($\mathcal{G}'$);
  - otherwise, #CSP($\mathcal{G}$) is as hard to approximate as #SAT.

Our hardness results build on the approximation classification in the Boolean case [5] and on the key role played by log-supermodular functions. The easiness results use the classification of the exact evaluation of #CSP($\mathcal{F}$) in the general case [7] and balance, in particular. The results concerning weak log-supermodularity build on key studies of the complexity of optimisation CSPs by Takhanov [24], Cohen, Cooper and Jeavons [12] and Kolmogorov and Živný [20]. We use arguments and ideas from these papers, and not merely their results. Thus, we must delve into them in some detail.

Our final result is a trichotomy for the binary case, where all $F \in \mathcal{F}$ have arity at most 2. This additionally uses work of Rudolf and Woeginger [23] on decomposing Monge matrices.

- If $\mathcal{F}$ is weakly log-modular then, for any finite $\mathcal{G} \subset \mathcal{F}$, #CSP($\mathcal{G}$) is in FP.
- Otherwise, if $\mathcal{F}$ is weakly log-supermodular, then
  - for every finite $\mathcal{G} \subset \mathcal{F}$, #CSP($\mathcal{G}$) is at least as easy to approximate as #BIS and
  - for some finite $\mathcal{G} \subset \mathcal{F}$, #CSP($\mathcal{G}$) is at least as hard to approximate as #BIS.
- Otherwise, for some finite $\mathcal{G} \subset \mathcal{F}$, #CSP($\mathcal{G}$) is as hard to approximate as #SAT.

1.1 Preliminaries and statement of results

If D is a finite domain with $|D| \geq 2$, we denote the set of functions $D^k \to R$, for some codomain R, by $\mathrm{Func}_k(D,R)$ and then $\mathrm{Func}(D,R) = \bigcup_{k=0}^{\infty} \mathrm{Func}_k(D,R)$. Let EQ be the binary equality function defined by $\mathrm{EQ}(x,x) = 1$ and $\mathrm{EQ}(x,y) = 0$ for $x \neq y$; let $\mathrm{NEQ}(x,y) = 1 - \mathrm{EQ}(x,y)$.

We use the following definitions from [5]. Let $\mathcal{F} \subseteq \mathrm{Func}(D,R)$ and let $V = \{v_1, \ldots, v_n\}$ be a set of variables. Atomic formulas have the form $\varphi = G(v_{i_1}, \ldots, v_{i_a})$ where $G \in \mathcal{F}$, $a = a(G)$ is the arity of G, and $(v_{i_1}, v_{i_2}, \ldots, v_{i_a}) \in V^a$ is the "scope". (The v_{i_j} need not be distinct.) The associated function is $F_\varphi \colon D^n \to R$, where $F_\varphi(\mathbf{x}) = G(\mathbf{x}(v_{i_1}), \ldots, \mathbf{x}(v_{i_a}))$ and $\mathbf{x} \colon V \to D$ is an assignment. To simplify notation, we write $x_j = \mathbf{x}(v_j)$ so $F_\varphi(\mathbf{x}) = G(x_{i_1}, \ldots, x_{i_a})$.

A pps-formula ("primitive product summation formula") is a sum of products of atomic formulas. A pps-formula ψ over $\mathcal{F}$ in variables $V' = \{v_1, \ldots, v_{n+k}\}$ and its associated function $F_\psi \colon D^n \to R$ are defined as follows.

$$\psi = \sum_{v_{n+1},\ldots,v_{n+k}} \prod_{j=1}^{m} \varphi_j, \qquad F_\psi(\mathbf{x}) = \sum_{\mathbf{y} \in D^k} \prod_{j=1}^{m} F_{\varphi_j}(\mathbf{x},\mathbf{y})\,. \tag{1}$$

Here φ_j are atomic formulas over $\mathcal{F}$ in the variables V'. (The variables in $V = \{v_1, \dots, v_n\}$ are *free* and those in $V' \setminus V$ are *bound*). The vectors $\mathbf{x}$ and $\mathbf{y}$ are assignments $\mathbf{x}\colon V \to D$ and $\mathbf{y}\colon V' \setminus V \to D$. The *functional clone $\langle \mathcal{F} \rangle_\#$ generated by* $\mathcal{F}$ is the set of functions representable by pps-formulas over $\mathcal{F} \cup \{\mathrm{EQ}\}$. Note that $\langle \langle \mathcal{F} \rangle_\# \rangle_\# = \langle \mathcal{F} \rangle_\#$ by [5, Lemma 1]. We will rely on this transitivity property implicitly.

▶ **Definition 1.**

1. A *weighted constraint language* $\mathcal{F}$ is a subset of $\mathrm{Func}(D, \mathbb{Q}_{\geq 0})$. Functions in $\mathcal{F}$ are called *weight functions.*
2. A weighted constraint language $\mathcal{F}$ is *conservative* if $\mathcal{U}_D \subseteq \mathcal{F}$, where $\mathcal{U}_D = \mathrm{Func}_1(D, \mathbb{Q}_{\geq 0})$.
3. A weighted constraint language $\mathcal{F}$ is *weakly log-modular* if, for all binary functions $F \in \langle \mathcal{F} \rangle_\#$ and all elements $a, b \in D$, $F(a,a)F(b,b) = F(a,b)F(b,a)$, or $F(a,a) = F(b,b) = 0$, or $F(a,b) = F(b,a) = 0$.
4. $\mathcal{F}$ is *weakly log-supermodular* if, for all binary functions $F \in \langle \mathcal{F} \rangle_\#$ and elements $a, b \in D$, $F(a,a)F(b,b) \geq F(a,b)F(b,a)$ or $F(a,a) = F(b,b) = 0$.
5. A function $F \in \mathrm{Func}_k(\{0,1\}, \mathbb{Q}_{\geq 0})$ is *log-supermodular* if $F(\mathbf{x} \vee \mathbf{y})F(\mathbf{x} \wedge \mathbf{y}) \geq F(\mathbf{x})F(\mathbf{y})$, for all $\mathbf{x}, \mathbf{y} \in \{0,1\}^k$, where $\wedge$ (min) and $\vee$ (max) are applied component-wise. LSM is the set of all log-supermodular functions in $\mathrm{Func}(\{0,1\}, \mathbb{Q}_{\geq 0})$. $\langle \mathsf{LSM} \rangle_\# = \mathsf{LSM}$ [5, Lemma 7].

Note that, in §4, we introduce *valued constraint languages* and *cost functions*, for optimisation CSPs. These should be distinguished from the weighted version, for counting.

The counting problem $\#\mathrm{CSP}(\mathcal{F})$ over a finite, weighted constraint language $\mathcal{F}$ is:

Instance. A pps-formula ψ, consisting of a product of m atomic $\mathcal{F}$-formulas over n free variables $\mathbf{x}$. (Thus, ψ has no bound variables.)
Output. The value $\sum_{\mathbf{x} \in D^n} F_\psi(\mathbf{x})$, where F_ψ is the function defined by ψ.

Where convenient, we write $\#\mathrm{CSP}(F)$ to mean $\#\mathrm{CSP}(\{F\})$ and write $\#\mathrm{CSP}(\mathcal{F}, \mathcal{F}')$ to mean $\#\mathrm{CSP}(\mathcal{F} \cup \mathcal{F}')$. As in [5] and other works, we take the size of a $\#\mathrm{CSP}(\mathcal{F})$ instance to be $n+m$, where n is the number of (free) variables and m is the number of weighted constraints (atomic formulas). In contrast to the unweighted case, the multiplicity of constraints matters, so we cannot bound m in terms of n. We typically denote an instance of $\#\mathrm{CSP}(\mathcal{F})$ by I and the output by $Z(I)$, which is often called the "partition function".

A counting problem, for our purposes, is any function from instances, encoded as words over a finite alphabet Σ, to $\mathbb{Q}_{\geq 0}$. A *randomised approximation scheme* for a counting problem $\#X$ is a randomised algorithm that takes an instance w and returns an approximation Y to $\#X(w)$, where a parameter $\varepsilon \in (0,1)$ specifies the error tolerance. Since the algorithm is randomised, the output Y is a random variable depending on the "coin tosses" made by the algorithm. We require that, for every instance w and every $\varepsilon \in (0,1)$,

$$\Pr\left[e^{-\varepsilon}\#X(w) \leq Y \leq e^{\varepsilon}\#X(w)\right] \geq 3/4\,.$$

The randomised approximation scheme is said to be a *fully polynomial randomised approximation scheme*, or *FPRAS*, if it runs in time bounded by a polynomial in $|w|$ (the length of the word w) and ε^{-1}. See Mitzenmacher and Upfal [22, Definition 10.2].

If $\#X$ and $\#Y$ are counting problems, an "approximation-preserving reduction" [15] (AP-reduction) turns an FPRAS for $\#Y$ into an FPRAS for $\#X$. Specifically, an *AP-reduction from $\#X$ to $\#Y$* is a randomised algorithm $\mathcal{A}$ for computing $\#X$ using an oracle for $\#Y$. $\mathcal{A}$'s input is a pair $(w, \varepsilon) \in \Sigma^* \times (0,1)$, and: (i) oracle calls made by $\mathcal{A}$ are of the form (v, δ), where $v \in \Sigma^*$ is an instance of $\#Y$, and $0 < \delta < 1$ is an error bound with $\delta^{-1} \leq \mathrm{poly}(|w|, \varepsilon^{-1})$;

(ii) $\mathcal{A}$ is a randomised approximation scheme for $\#X$ whenever the oracle is a randomised approximation scheme for $\#Y$; and (iii) the run-time of $\mathcal{A}$ is polynomial in $|w|$ and ε^{-1}. If an AP-reduction from $\#X$ to $\#Y$ exists, we write $\#X \leq_{\mathrm{AP}} \#Y$. A counting problem $\#X$ is $\#Y$-easy if $\#X \leq_{\mathrm{AP}} \#Y$ and it is $\#Y$-hard if $\#Y \leq_{\mathrm{AP}} \#X$. A problem $\#X$ is LSM-*easy* if there is a finite, weighted constraint language $\mathcal{F} \subset \mathsf{LSM}$ such that $\#X \leq_{\mathrm{AP}} \#\mathrm{CSP}(\mathcal{F})$.

The notion of pps-definability is closely related to AP-reductions. In particular, [5, Lemma 17] shows that $G \in \langle \mathcal{F} \rangle_{\#}$ implies that $\#\mathrm{CSP}(\mathcal{F}, G) \leq_{\mathrm{AP}} \#\mathrm{CSP}(\mathcal{F})$. We will use this fact without comment. Note that, subsequent to [15], the notation $\leq_{\mathrm{AP}}$ has been used for a different approximation-preserving reduction which applies to optimisation problems. Since our emphasis is on counting problems, this should not cause confusion.

We now state our main theorem. Note that we have only defined $\#\mathrm{CSP}(\mathcal{F})$ for finite $\mathcal{F}$.

▶ **Theorem 2.** *Let $\mathcal{F}$ be a conservative weighted constraint language taking values in $\mathbb{Q}_{\geq 0}$.*

1. *If $\mathcal{F}$ is weakly log-modular then $\#\mathrm{CSP}(\mathcal{G})$ is in* FP *for every finite $\mathcal{G} \subset \mathcal{F}$.*
2. *If $\mathcal{F}$ is weakly log-supermodular but not weakly log-modular, then $\#\mathrm{CSP}(\mathcal{G})$ is* LSM*-easy for every finite $\mathcal{G} \subset \mathcal{F}$ and #BIS-hard for some such $\mathcal{G}$.*
3. *If $\mathcal{F}$ is weakly log-supermodular but not weakly log-modular and consists of functions of arity at most two, then $\#\mathrm{CSP}(\mathcal{G})$ is #BIS-easy for every finite $\mathcal{G} \subset \mathcal{F}$ and #BIS-equivalent for some such $\mathcal{G}$.*
4. *If $\mathcal{F}$ is not weakly log-supermodular, then $\#\mathrm{CSP}(\mathcal{G})$ is #SAT-easy for every finite $\mathcal{G} \subset \mathcal{F}$ and #SAT-equivalent for some such $\mathcal{G}$.*

In particular, among conservative #CSPs, there are no new complexity classes below #BIS or above LSM; furthermore there is a trichotomy for conservative weighted constraint languages with no functions of arity greater than two.

#BIS-hardness and relationships with #SAT are proved in §2, restated as Theorem 4. Membership in FP is Theorem 7 in §3. LSM-easiness and #BIS-easiness are established by Theorem 29 in §6. Further, there is an algorithm that determines which case of Theorem 2 holds for $\mathcal{H} \cup \mathcal{U}_D$ where $\mathcal{H}$ is finite, as shown in §7. Full proofs are in [11].

2 Hardness results

Our hardness results use the following special case of [5, Theorem 18]. That result is expressed in terms of efficiently computable reals and we restrict to rationals for simplicity. The statement of [5, Theorem 18] does not imply our lemma but the proof does.

▶ **Lemma 3.** *Let F be a function in $\mathrm{Func}_2(\{0,1\}, \mathbb{Q}_{\geq 0})$.*

- *If $F \notin \langle \mathrm{NEQ}, \mathcal{U}_{\{0,1\}} \rangle_{\#}$, then $\#\mathrm{CSP}(\mathcal{G})$ is #BIS-hard for some finite $\mathcal{G} \subset \{F\} \cup \mathcal{U}_{\{0,1\}}$.*
- *If $F \notin \langle \mathrm{NEQ}, \mathcal{U}_{\{0,1\}} \rangle_{\#} \cup \mathsf{LSM}$, then $\#\mathrm{CSP}(\mathcal{G})$ is #SAT-hard for some finite $\mathcal{G} \subset \{F\} \cup \mathcal{U}_{\{0,1\}}$.*

▶ **Theorem 4.** *Let $\mathcal{F}$ be a conservative weighted constraint language taking values in $\mathbb{Q}_{\geq 0}$.*

- *If $\mathcal{F}$ is not weakly log-modular, $\#\mathrm{CSP}(\mathcal{G})$ is #BIS-hard for some finite $\mathcal{G} \subset \mathcal{F}$.*
- *If $\mathcal{F}$ is not weakly log-supermodular, $\#\mathrm{CSP}(\mathcal{G})$ is #SAT-hard for some finite $\mathcal{G} \subset \mathcal{F}$.*
- *For all finite $\mathcal{G} \subset \mathcal{F}$, $\#\mathrm{CSP}(\mathcal{G})$ is #SAT-easy.*

Proof (sketch). Functions in $\langle \mathrm{NEQ}, \mathcal{U}_{\{0,1\}} \rangle_{\#}$ can have only certain forms. If $\mathcal{F}$ is not weakly log-modular, we construct $F \in \langle \mathcal{F} \rangle_{\#}$ that does not have any of these forms and #BIS-hardness follows from Lemma 3. The proof of #SAT-hardness is similar. #SAT-easiness follows from the reduction to the unweighted case [3], which is #SAT-easy [15]. ◀

3 Balance and weak log-modularity

In this section we use the notion of balance to show that weak log-modularity implies tractability. We may associate a matrix $\mathbf{M}$ with an undirected bipartite graph $G_{\mathbf{M}}$ whose vertex partition consists of the set of rows R and columns C of $\mathbf{M}$. A pair $(r, c) \in R \times C$ is an edge of $G_{\mathbf{M}}$ if, and only if, $\mathbf{M}_{rc} \neq 0$. A *block* of $\mathbf{M}$ is a submatrix whose rows and columns form a connected component in $G_{\mathbf{M}}$. $\mathbf{M}$ has block-rank 1 if all its blocks have rank 1.

A weighted constraint language $\mathcal{F}$ is *balanced* [7, 18] if, for every function $F(x_1, \dots, x_n) \in \langle\mathcal{F}\rangle_{\#}$ with arity $n \geq 2$, and every k with $0 < k < n$, the $|D|^k \times |D|^{n-k}$ matrix $F((x_1, \dots, x_k), (x_{k+1}, \dots, x_n))$ has block-rank 1.

A function $F \colon \{0,1\}^n \to \mathbb{R}$ has *rank 1* if it has the form $F(x_1, \dots, x_k) = U_1(x_1) \cdots U_k(x_k)$. A function $F \colon D^n \to \mathbb{Q}_{\geq 0}$ is *essentially pseudo-Boolean* if its support is a subset of $D_1 \times \cdots \times D_n$ with $|D_1|, \dots, |D_n| \leq 2$. The *projection* of a relation $R \subseteq D^n$ onto indices $1 \leq i < j \leq n$ is the set of $(a, b) \in D^2$ such that there exists $\mathbf{x} \in R$ with $x_i = a$ and $x_j = b$. A *generalised* NEQ is a relation $\{(x_i, x_j), (y_i, y_j)\} \subset D^2$ for some $x_i \neq y_i$ and $x_j \neq y_j$.

▶ **Lemma 5.** *Let $F \colon D^n \to \mathbb{Q}_{\geq 0}$ be an essentially pseudo-Boolean function which is not of rank 1. If $\{F\} \cup \mathcal{U}_D$ is weakly log-modular, then some binary projection of the support of F is a generalised* NEQ.

▶ **Lemma 6.** *Every conservative weakly log-modular weighted constraint language is balanced.*

Proof (sketch). If $\mathcal{F}$ is not balanced, there is a function $F \in \langle\mathcal{F}\rangle_{\#}$ and a partition of its variables for which the corresponding matrix $\mathbf{M}$ is not block-rank-1. We show that $\mathbf{M}$ has a 2×2 submatrix that is not block-rank-1 and use this to construct an essentially pseudo-Boolean function G that is not of rank 1 and none of whose binary projections is a generalised NEQ. Now, use Lemma 5. ◀

This, along with the dichotomy of Cai, Chen and Lu [7] for the complexity of exact evaluation of #CSP, gives us the tractable case of Theorem 2, since balanced problems can be solved exactly, in polynomial time.

▶ **Theorem 7.** *Let $\mathcal{F}$ be a conservative weighted constraint language taking values in $\mathbb{Q}_{\geq 0}$. If $\mathcal{F}$ is weakly log-modular then, for any finite $\mathcal{G} \subset \mathcal{F}$, $\#\mathrm{CSP}(\mathcal{G}) \in \mathrm{FP}$.*

4 Valued clones, valued CSPs and relational clones

To define valued clones, we use the same set-up as §1.1 except that summation is replaced by minimisation and product is replaced by sum. Let D be a finite domain with $|D| \geq 2$ and let R be a codomain with $\{0, \infty\} \subseteq R$, where ∞ obeys the following rules for all $x \in R$: $x + \infty = \infty$, $x \leq \infty$ and $\min\{x, \infty\} = x$. Let Φ be a subset of $\mathrm{Func}(D, R)$. Let $V = \{v_1, \dots, v_n\}$ be a set of variables. For each atomic formula $\varphi = G(v_{i_1}, \dots, v_{i_a})$ we use the notation f_φ to denote the function represented by φ, so $f_\varphi(\mathbf{x}) = G(x_{i_1}, \dots, x_{i_a})$.

A psm-formula ("primitive sum minimisation formula") is a minimisation of a sum of atomic formulas. A psm-formula ψ over Φ in variables $V' = \{v_1, \dots, v_{n+k}\}$, and its associated function f_ψ are defined, analogously to (1), by

$$\psi = \min_{v_{n+1}, \dots, v_{n+k}} \sum_{j=1}^{m} \varphi_j, \qquad f_\psi(\mathbf{x}) = \min_{\mathbf{y} \in D^k} \sum_{j=1}^{m} f_{\varphi_j}(\mathbf{x}, \mathbf{y}),$$

where the φ_j are atomic formulas and $\mathbf{x} \colon \{v_1, \dots, v_n\} \to D$, $\mathbf{y} \colon \{v_{n+1}, \dots, v_{n+k}\} \to D$ are assignments.

The *valued clone* $\langle\Phi\rangle_V$ *generated by* Φ is the set of all functions that can be represented by a psm-formula over $\Phi \cup \{\text{eq}\}$, where eq is the binary equality function on D given by $\text{eq}(x,x) = 0$ and $\text{eq}(x,y) = \infty$ for $x \neq y$.

We next introduce valued constraint satisfaction problems (VCSPs), which are optimisation problems. In the work of Kolmogorov and Živný [20], the codomain is $R = \mathbb{Q}_{\geq 0} \cup \{\infty\}$. It will be useful to extend the codomain to include irrational numbers. This causes no problems since, apart from Theorem 23, we use only calculations from their papers, not complexity results. For Theorem 23, we avoid irrational numbers and, in fact, restrict to cost functions taking values in $\{0, \infty\} \subset R$. Furthermore, all the real numbers we use are either rationals, or their logarithms, so are efficiently computable.

Let $\overline{\mathbb{R}}_{\geq 0} = \mathbb{R}_{\geq 0} \cup \{\infty\}$ be the set of non-negative real numbers together with ∞.

▶ **Definition 8.** A *cost function* is a function $D^k \to \overline{\mathbb{R}}_{\geq 0}$. A *valued constraint language* is a set of cost functions $\Phi \subseteq \text{Func}(D, \overline{\mathbb{R}}_{\geq 0})$.

For a valued constraint language Φ, a problem in VCSP(Φ) has instance ψ, a psm-formula which is a sum of m atomic Φ-formulas in n free variables $\mathbf{x}$, and computes the value

$$\text{minCost}(\psi) = \min_{\mathbf{x} \in D^n} f_\psi(\mathbf{x}), \quad \text{where } f_\psi \text{ is the function defined by } \psi.$$

We typically use the notation of Kolmogorov and Živný. An instance is usually denoted by the letter I. In this case, we use f_I to denote the function specified by the psm-formula corresponding to instance I, so the value of the instance is denoted by minCost(I). The psm-formula corresponding to I is a sum of atomic formulas (since all of the variables are free variables). We refer to each of these atomic formulas as a *valued constraint* and we represent these by the multiset T of all valued constraints in the instance I. For each valued constraint $t \in T$ we use k_t to denote its arity, f_t to denote the function represented by the corresponding atomic formula, and σ_t to denote its scope, which is given as a tuple $(i(t,1), \ldots, i(t,k_t)) \in \{1, \ldots, n\}^{k_t}$ containing the indices of the variables in the scope. Thus,

$$f_I(\mathbf{x}) = \sum_{t \in T} f_t(x_{i(t,1)}, \ldots, x_{i(t,k_t)}). \tag{2}$$

We will use $\mathbf{x}[\sigma_t]$ as an abbreviation for the tuple $(x_{i(t,1)}, \ldots, x_{i(t,k_t)})$. In this abbreviated notation, the function defined by instance I may be written as $f_I(\mathbf{x}) = \sum_{t \in T} f_t(\mathbf{x}[\sigma_t])$.

Now, let $[0,1]_\mathbb{Q} = [0,1] \cap \mathbb{Q}$. For reasons which will be clear below, it will be useful to work with weight functions in Func($D, [0,1]_\mathbb{Q}$). For such a weight function F, let the cost function $\ell(F) \in \text{Func}(D, \overline{\mathbb{R}}_{\geq 0})$ be the function defined by

$$(\ell(F))(\mathbf{x}) = \begin{cases} -\ln F(\mathbf{x}) & \text{if } F(\mathbf{x}) > 0 \\ \infty & \text{if } F(\mathbf{x}) = 0. \end{cases}$$

For example, $\ell(\text{EQ}) = \text{eq}$. For a weighted constraint language $\mathcal{F} \subseteq \text{Func}(D, [0,1]_\mathbb{Q})$, let $\ell(\mathcal{F})$ be the valued constraint language defined by $\ell(\mathcal{F}) = \{\ell(F) \mid F \in \mathcal{F}\}$.

There is a bijection between instances of #CSP($\mathcal{F}$) and VCSP($\ell(\mathcal{F})$), given by replacing each function F_t in the former by the function $f_t = \ell(F_t)$ in the latter, with scopes unchanged. Note that $f_I(\mathbf{x}) = -\ln F_I(\mathbf{x})$, for any assignment $\mathbf{x}$, with the convention $-\ln 0 = \infty$.

▶ **Definition 9.** A valued constraint language is *conservative* if it contains all arity-1 cost functions $D \to \overline{\mathbb{R}}_{\geq 0}$.

The mapping $F \mapsto \ell(F)$ from Func($D, [0,1]_\mathbb{Q}$) to Func($D, \overline{\mathbb{R}}_{\geq 0}$) is not surjective since not all real numbers are logarithms of rationals. Similarly, for any weighted constraint language

$\mathcal{F}$, the valued constraint language $\ell(\mathcal{F})$ is not conservative. Finally, note that we define $\ell(F)$ only for $F \in \mathrm{Func}(D, [0,1]_{\mathbb{Q}})$. The extension to $F \in \mathrm{Func}(D, \mathbb{Q}_{\geq 0})$ produces negative-valued cost functions. We wish to avoid this since Kolmogorov and Živný [20] do not allow it.

▶ **Definition 10.** A cost function is *crisp* [13] if $f(\mathbf{x}) \in \{0, \infty\}$ for all $\mathbf{x}$.

▶ **Definition 11.** For a cost function f, let $\mathrm{Feas}(f)$ be the relation $\mathrm{Feas}(f) = \{\mathbf{x} \mid f(\mathbf{x}) < \infty\}$.

Thus, any cost function f can be associated with its underlying relation. Similarly, we can represent any relation by a crisp cost function f for which $f(\mathbf{x}) = 0$ if and only if $\mathbf{x}$ is in the relation. A *crisp constraint language* is a set of relations, which we always represent as crisp cost functions, not as functions with codomain $\{0, 1\}$. For a valued constraint language Φ, the crisp constraint language $\mathrm{Feas}(\Phi)$ is given by $\mathrm{Feas}(\Phi) = \{\mathrm{Feas}(f) \mid f \in \Phi\}$. A *relational clone* is simply a crisp constraint language $\mathrm{Feas}(\langle\Phi\rangle_V)$ for a valued constraint language Φ.

▶ **Lemma 12.** *Suppose* $\Phi \subseteq \mathrm{Func}(D, \overline{\mathbb{R}}_{\geq 0})$. *Then* $\langle \mathrm{Feas}(\Phi)\rangle_V = \mathrm{Feas}(\langle\Phi\rangle_V)$.

Proof. The mapping $\rho\colon \overline{\mathbb{R}}_{\geq 0} \to \{0, \infty\}$ defined by $\rho(\infty) = \infty$ and $\rho(x) = 0$, for all $x < \infty$, is a homomorphism of semirings, from $(\overline{\mathbb{R}}_{\geq 0}, \min, +)$ to $(\{0, \infty\}, \min, +)$. ◀

▶ **Definition 13.** A crisp constraint language is *conservative* if it includes all arity-1 relations.

5 STP/MJN multimorphisms and weak log-supermodularity

In [20, Corollary 12], Kolmogorov and Živný showed that the VCSP associated with a conservative valued constraint language Φ is tractable iff Φ has an STP/MJN multimorphism.

We define STP/MJN multimorphisms below. In this section, we show that, if a weighted constraint language $\mathcal{F} \in \mathrm{Func}(D, [0,1]_{\mathbb{Q}})$ is weakly log-supermodular, the corresponding valued constraint language $\ell(\mathcal{F})$ has an STP/MJN multimorphism. In §6, this will enable us to use such a multimorphism (via the work of Kolmogorov and Živný [20] and Cohen, Cooper and Jeavons [12]) to prove #BIS-easiness and LSM-easiness of the weighted counting CSP.

Our proofs rely on work of Kolmogorov and Živný [20] and Takhanov [24]. We start with some general definitions. Note that some of these definitions in [20] differ from those in [12].

▶ **Definition 14.**

1. A *k-ary operation on D* is a function from D^k to D. An *operation on D* is a k-ary operation, for some k. We omit "on D" when the domain D is clear from the context.
2. A k-tuple $\langle\rho_1, \dots, \rho_k\rangle$ of k-ary operations $\rho_1, \dots, \rho_k$ is *conservative* if the multisets $\{\{x_1, \dots, x_k\}\}$ and $\{\{\rho_1(\mathbf{x}), \dots, \rho_k(\mathbf{x})\}\}$ are equal for all $\mathbf{x} = (x_1, \dots, x_k) \in D^k$.
3. $\langle\rho_1, \dots, \rho_k\rangle$ is a *multimorphism* of an arity-r cost function f if:
$$\sum_{i=1}^{k} f(\rho_i(x_1^1, \dots, x_1^k), \dots, \rho_i(x_r^1, \dots, x_r^k)) \leq \sum_{i=1}^{k} f(\mathbf{x}^i) \quad \text{for all } \mathbf{x}^1, \dots, \mathbf{x}^k \in D^r.$$
4. $\langle\rho_1, \dots, \rho_k\rangle$ is a multimorphism of a valued constraint language Φ if it is a multimorphism of every $f \in \Phi$.

Note that we have defined "conservative" for operations and constraint languages (weighted, valued and crisp). These notions are connected, but we do not need that here.

▶ **Observation 15.** If $\langle\rho_1, \dots, \rho_k\rangle$ is conservative, it is a multimorphism of every unary f.

▶ **Definition 16.**

1. Suppose $M \subseteq D^2$. A pair $\langle \sqcap, \sqcup \rangle$ of binary operations is a *symmetric tournament pair* (STP) on M if it is conservative and both operations are commutative on M. We say that it is an STP if it is an STP on D^2.
2. Suppose $M \subseteq D^2$. A triple $\langle \mathtt{Mj1}, \mathtt{Mj2}, \mathtt{Mn3} \rangle$ of ternary operations is an *MJN on* M if it is conservative and, for all triples $(a, b, c) \in D^3$ with $\{\{a, b, c\}\} = \{\{x, x, y\}\}$ where $x \neq y$ and $(x, y) \in M$, we have $\mathtt{Mj1}(a, b, c) = \mathtt{Mj2}(a, b, c) = x$ and $\mathtt{Mn3}(a, b, c) = y$.
3. An *STP/MJN multimorphism* of a valued constraint language Φ consists of a pair of operations $\langle \sqcap, \sqcup \rangle$ and a triple of operations $\langle \mathtt{Mj1}, \mathtt{Mj2}, \mathtt{Mn3} \rangle$, both of which are multimorphisms of Φ, for which, for some symmetric subset M of D^2, $\langle \sqcap, \sqcup \rangle$ is an STP on M and $\langle \mathtt{Mj1}, \mathtt{Mj2}, \mathtt{Mn3} \rangle$ is an MJN on $\{(a, b) \in D^2 \setminus M \mid a \neq b\}$.
4. $\Phi \subseteq \mathrm{Func}(D, \overline{\mathbb{R}}_{\geq 0})$ is *weakly submodular* if, for all binary $f \in \langle \Phi \rangle_V$ and $a, b \in D$, $f(a, a) + f(b, b) \leq f(a, b) + f(b, a)$ or $f(a, a) = f(b, b) = \infty$.

Our definition of weak submodularity for cost functions restates Kolmogorov and Živný's "Assumption 3". It is nontrivial that weak log-supermodularity for $\mathcal{F}$ is related to weak submodularity for $\ell(\mathcal{F})$. In particular, we cannot expect $\langle \ell(\mathcal{F}) \rangle_V = \ell(\langle \mathcal{F} \rangle_\#)$ to hold in general. However, the following is suitable for our purposes.

▶ **Lemma 17.** *Suppose* $\mathcal{F} \subseteq \mathrm{Func}(D, [0, 1]_\mathbb{Q})$ *and let* $\Phi = \ell(\mathcal{F})$. *If* $\mathcal{F}$ *is weakly log-supermodular then* Φ *is weakly submodular.*

Proof (sketch). For the contrapositive, take a binary $f \in \langle \Phi \rangle_V$ that is not weakly submodular and let $F^{(k)} \in \langle \mathcal{F} \rangle_\#$ be the function defined by replacing every atomic formula $g(\mathbf{x}, \mathbf{y})$ in the definition of f with $G(\mathbf{x}, \mathbf{y})^k$, where $g = \ell(G)$. For k large enough, $F^{(k)}(\mathbf{x})^{1/k}$ is close enough to the maximum value taken by any $G(\mathbf{x}, \mathbf{y})$ in the definition of F. But this maximum value is $\exp(-f(\mathbf{x}))$ and we can now show that $F^{(k)}$ is not weakly log-supermodular. ◀

Let Γ be a crisp constraint language. A *majority polymorphism* of Γ is a ternary operation ρ such that $\rho(a, a, b) = \rho(a, b, a) = \rho(b, a, a) = a$ for all $a, b \in D$, and for all k-ary relations $R \in \Gamma$, $\mathbf{x}, \mathbf{y}, \mathbf{z} \in R$ implies that $(\rho(x_1, y_1, z_1), \ldots, \rho(x_k, y_k, z_k)) \in R$.

The formulation of the following theorem is essentially [24, Theorem 9.1] (but note that Takhanov uses the term "functional clone" in a different way to us).

▶ **Theorem 18** (Takhanov). *Let* $N(a, b, c, d)$ *be the relation* $\{(a, c), (b, c), (a, d)\}$, *and let* Γ *be a conservative relational clone with domain* D. *At least one of the following holds.*

- *There are distinct* $a, b \in D$ *such that* $N(a, b, a, b) \in \Gamma$.
- *There are distinct* $a, b \in D$ *such that* $\{(a, a, a), (a, b, b), (b, a, b), (b, b, a)\} \in \Gamma$.
- *For some* $k \geq 1$, *there are* $a_0, \ldots, a_{2k}, b_0, \ldots, b_{2k} \in D$ *such that, for each* $0 \leq i \leq 2k$, $a_i \neq b_i$ *and, for each* $0 \leq i \leq 2k - 1$, $N(a_i, b_i, a_{i+1}, b_{i+1}) \in \Gamma$ *and* $N(a_{2k}, b_{2k}, a_0, b_0) \in \Gamma$.
- Γ *has a majority polymorphism.*

To prove the following lemma, we show that the first three bullets of Takhanov's theorem cannot hold, so the fourth must.

▶ **Lemma 19.** *If* $\Phi \subseteq \mathrm{Func}(D, \overline{\mathbb{R}}_{\geq 0})$ *is conservative and weakly submodular then* $\Gamma = \langle \mathrm{Feas}(\Phi) \rangle_V$ *has a majority polymorphism.*

The main theorem of this section now follows from Lemmas 17 and 19. The final construction of the STP/MJN multimorphism comes from [20, §§6.1–6.4].

▶ **Theorem 20.** *Let* $\mathcal{F}$ *be a weighted constraint language with* $\mathrm{Func}_1(D, [0, 1]_\mathbb{Q}) \subseteq \mathcal{F} \subseteq \mathrm{Func}(D, [0, 1]_\mathbb{Q})$. *If* $\mathcal{F}$ *is weakly log-supermodular then* $\ell(\mathcal{F})$ *has an STP/MJN multimorphism.*

6 LSM-**easiness and** #BIS-**easiness**

Our aim is to show that $\mathcal{F}$ is LSM-easy if $\ell(\mathcal{F})$ has an STP/MJN multimorphism. This will use arguments from [12] and [20], but we try, as far as possible, to avoid going into the details of their proofs. We start by generalising the notion of an STP multimorphism.

▶ **Definition 21.** Let f be an arity-k cost function. A *generalised STP multimorphism of f* is a pair $\langle \sqcap, \sqcup \rangle$, defined as follows. For $1 \leq i \leq k$, $\sqcap_i$ and $\sqcup_i$ are operations on the set $D_i = \{a \in D \mid \exists \mathbf{x} : x_i = a \text{ and } f(\mathbf{x}) < \infty\}$, and $\langle \sqcap_i, \sqcup_i \rangle$ is an STP of $\{f\}$.

The operation $\sqcap$ is the binary operation on $D_1 \times \cdots \times D_k$ defined by applying $\sqcap_1, \ldots, \sqcap_k$ component-wise. Similarly, $\sqcup$ is defined by applying $\sqcup_1, \ldots, \sqcup_k$ component-wise. We require that, for all $\mathbf{x}, \mathbf{y} \in D^k$, $f(\sqcup(\mathbf{x}, \mathbf{y})) + f(\sqcap(\mathbf{x}, \mathbf{y})) \leq f(\mathbf{x}) + f(\mathbf{y})$. Equivalently, we require that $f(\sqcup_1(x_1, y_1), \ldots, \sqcup_k(x_k, y_k)) + f(\sqcap_1(x_1, y_1), \ldots, \sqcap_k(x_k, y_k)) \leq f(\mathbf{x}) + f(\mathbf{y})$.

Theorem 22 is closely related to [20, Theorem 11]. Its proof is the same but we stop when [20, Lemma 35] has been proved. We also need an algorithmic consequence.

▶ **Theorem 22** (Kolmogorov and Živný)**.** *Suppose Φ_0 is a finite, valued constraint language which has an STP/MJN multimorphism. Then there is a polynomial-time algorithm that takes an instance I of* VCSP(Φ_0) *and returns a generalised STP multimorphism $\langle \sqcap, \sqcup \rangle$ of f_I. The pair $\langle \sqcap, \sqcup \rangle$ depends only on the STP/MJN multimorphism of Φ_0 and on the relation* Feas(f_I) *underlying f_I. It does not depend in any other way on I.*

▶ **Theorem 23** (Kolmogorov and Živný)**.** *If Φ_0 is a finite, crisp constraint language that has an STP/MJN multimorphism, then there is a polynomial-time algorithm for* VCSP(Φ_0).

For our eventual construction, we would like $\langle \sqcap, \sqcup \rangle$ to induce a generalised STP multimorphism of f_t for each individual valued constraint t in the instance. We do not know whether this is true of the generalised STP multimorphism provided by Kolmogorov and Živný's algorithm, but something sufficiently close to this is true.

▶ **Definition 24.** For an instance I, a valued constraint t and a length-k_t vector $\mathbf{a}$, define $R_{I,t}(\mathbf{a}) = 0$ if there exists $\mathbf{x}$ with $\mathbf{x}[\sigma_t] = \mathbf{a}$ and $f_I(\mathbf{x}) < \infty$ and $R_{I,t}(\mathbf{a}) = \infty$, otherwise. Define $f'_t = f_t + R_{I,t}$. Thus, f'_t is a "trimmed" version of f_t whose domain is precisely the k_t-tuples of values that can actually arise in feasible solutions to instance I.

We will see that, if the scope σ_t contains variables with indices $i(t,1), \ldots, i(t,k_t)$, then $\big\langle \sqcap[\sigma_t], \sqcup[\sigma_t] \big\rangle = \big\langle (\sqcap_{i(t,1)}, \ldots, \sqcap_{i(t,k_t)}), (\sqcup_{i(t,1)}, \ldots, \sqcup_{i(t,k_t)}) \big\rangle$ is a generalised STP multimorphism of f'_t, even though it might not necessarily be a generalised STP multimorphism of f_t. Note that Theorem 23 has the following consequence.

▶ **Corollary 25.** *Let Φ_0 be a finite, valued constraint language that has an STP/MJN multimorphism. There is a polynomial-time algorithm that takes an instance I of* VCSP(Φ_0)*, a valued constraint t and a length-k_t vector* $\mathbf{a}$ *and returns a (finite) truth table for f'_t.*

▶ **Theorem 26** (Extends Theorem 22)**.** *Suppose Φ_0 is a finite, valued constraint language which has an STP/MJN multimorphism. Consider the algorithm from Theorem 22 which takes an instance I of* VCSP(Φ_0) *(in the form (2)) and returns a generalised STP multimorphism $\langle \sqcap, \sqcup \rangle$ of f_I. Then, for all $t \in T$, $\big\langle \sqcap[\sigma_t], \sqcup[\sigma_t] \big\rangle$ is a generalised STP multimorphism of f'_t.*

We say that two n-variable instances I and I' of VCSP(Φ) are *equivalent* if $f_I(\mathbf{x}) = f_{I'}(\mathbf{x})$ for all $\mathbf{x} \in D^n$. Given a finite, valued constraint language $\Phi_0 \subset \text{Func}(D, \overline{\mathbb{R}}_{\geq 0})$, let Φ'_0 be the set of functions of the form $f + R$, for $f \in \Phi_0 \cap \text{Func}_k(D, \overline{\mathbb{R}}_{\geq 0})$, $R \in \text{Func}_k(D, \{0, \infty\})$ and $k \in \mathbb{N}$. Note that Φ'_0 is finite because $\text{Func}_k(D, \{0, \infty\})$ is finite for any finite k. These definitions, along with Corollary 25 and Theorem 26 allow us to prove the following.

▶ **Lemma 27.** *Suppose Φ_0 is a finite, valued constraint language which has an STP/MJN multimorphism. Consider an instance I of* VCSP(Φ_0)*. There is an equivalent instance I' of* VCSP(Φ_0') *and a generalised STP multimorphism $\langle \sqcap, \sqcup \rangle$ of $f_{I'}$ which induces a generalised STP multimorphism of f_t for each valued constraint t of I'. Both I' and $\langle \sqcap, \sqcup \rangle$ are polynomial-time computable (given I). Moreover, each operation $\sqcap_i$ and $\sqcup_i$ induces a total order.*

The proof of the next lemma uses a reduction from the given #CSP problem over domain D to a Boolean problem. A variable x ranging over D is simulated by Boolean variables $\{x_d \mid d \in D\}$ and an assignment $x = a$ is simulated by setting $x_b = 1$ for all $b < a$ and $x_b = 0$, otherwise, where "$b < a$" is determined by the appropriate total order from the multimorphism. The constraints needed to prevent assignments to the x_d that do not correspond to an assignment to x can be represented by LSM functions, as can the corresponding translation of a weight function $f\colon D^k \to [0,1]_{\mathbb{Q}}$ to $f'\colon \{0,1\}^{kd} \to [0,1]_{\mathbb{Q}}$. In the case where all functions in $\mathcal{F}$ have arity at most 2, it can be shown that all of the necessary constraints can be implemented using Boolean implication and unary weights.

▶ **Lemma 28.** *If $\mathcal{F} \subseteq \mathrm{Func}(D, [0,1]_{\mathbb{Q}})$ and $\ell(\mathcal{F})$ has an STP/MJN multimorphism, then, for every finite $\mathcal{G} \subset \mathcal{F}$, #CSP($\mathcal{G}$) is* LSM*-easy, and* #BIS*-easy if every weight function has arity at most 2.*

Our main theorem, Theorem 2, now follows from Theorems 4 and 7 and the following.

▶ **Theorem 29.** *Let $\mathcal{F}$ be a weakly log-supermodular, conservative weighted constraint language taking values in $\mathbb{Q}_{\geq 0}$.*

- *For any finite $\mathcal{G} \subset \mathcal{F}$, there is a finite $\mathcal{G}' \subset$ LSM such that #CSP($\mathcal{G}$) $\leq_{\mathrm{AP}}$ #CSP($\mathcal{G}'$).*
- *If all $F \in \mathcal{F}$ have arity at most two, then #CSP($\mathcal{G}$) is #BIS-easy for any finite $\mathcal{G} \subset \mathcal{F}$.*

Proof (sketch). Scale $\mathcal{F}$ to obtain a weakly log-supermodular weighted constraint language $\mathcal{F}'$ where all weights are in $[0,1]_{\mathbb{Q}}$. By Theorem 20, $\ell(\mathcal{F}')$ has an STP/MJN multimorphism. The result follows from Lemma 28 and the fact that the scaling does not affect complexity. ◀

7 Algorithmic aspects

Finally, we show that there is an algorithm that determines the complexity of approximating #CSP with constraints from the language $\mathcal{H} \cup \mathcal{U}_D$ for finite $\mathcal{H}$. The proof is by reduction to determining whether a certain finite subset of $\mathcal{H} \cup \mathcal{U}_D$ is balanced, whether $\ell(\mathcal{H})$ has an STP/MJN multimorphism and whether $\mathcal{H}$'s weight functions have arity at most 2. Balance is decidable by [7], the existence of an STP/MJN multimorphism can be checked by brute force, since $\mathcal{H}$ is finite, and we can clearly check the maximum arity of the weight functions.

▶ **Theorem 30.** *There is an algorithm that, given a finite, weighted constraint language $\mathcal{H}$ taking values in $\mathbb{Q}_{\geq 0}$, correctly makes one of the following deductions, where $\mathcal{F} = \mathcal{H} \cup \mathcal{U}_D$:*

1. *#CSP($\mathcal{G}$) is in* FP *for every finite $\mathcal{G} \subset \mathcal{F}$;*
2. *#CSP($\mathcal{G}$) is* LSM*-easy for every finite $\mathcal{G} \subset \mathcal{F}$ and #BIS-hard for some such $\mathcal{G}$;*
3. *#CSP($\mathcal{G}$) is #BIS-easy for all finite $\mathcal{G} \subset \mathcal{F}$ and #BIS-equivalent for some such $\mathcal{G}$;*
4. *#CSP($\mathcal{G}$) is #SAT-easy for all finite $\mathcal{G} \subset \mathcal{F}$ and #SAT-equivalent for some such $\mathcal{G}$.*

If every function in $\mathcal{H}$ has arity at most 2, the output is not deduction 2.

References

1 E. Boros and P. Hammer. Pseudo-Boolean optimization. *Discrete Appl. Math.*, 123:155–225, 2002.

2 A. Bulatov. The complexity of the counting constraint satisfaction problem. In *Proc. 35th ICALP (Part I)*, LNCS 5125, pp. 646–661. Springer, 2008.

3 A. Bulatov, M. Dyer, L.A. Goldberg, M. Jalsenius, M. Jerrum and D. Richerby. The complexity of weighted and unweighted #CSP. *J. Comput. Syst. Sci.*, 78:681–688, 2012.

4 A. Bulatov, M. Dyer, L.A. Goldberg, M. Jalsenius and D. Richerby. The complexity of weighted Boolean #CSP with mixed signs. *Theor. Comput. Sci.*, 410:3949–3961, 2009.

5 A. Bulatov, M. Dyer, L.A. Goldberg, M. Jerrum and C. McQuillan. The expressibility of functions on the Boolean domain, with applications to counting CSPs. CoRR eprint, arXiv:abs/1108.5288, 2011.

6 J.-Y. Cai and X. Chen. Complexity of counting CSP with complex weights. In *Proc. 44th STOC*, pp. 909–920. ACM, 2012.

7 J.-Y. Cai, X. Chen, and P. Lu. Non-negatively weighted #CSP: An effective complexity dichotomy. In *Proc. 26th CCC*, pp. 45–54. IEEE, 2011.

8 J.-Y. Cai, P. Lu, and M. Xia. Dichotomy for Holant* problems of Boolean domain. In *Proc. 22nd SODA*, pp. 1714–1728. ACM–SIAM, 2011.

9 J.-Y. Cai, P. Lu and M. Xia. Holant problems and counting CSP. In *Proc. 41st STOC*, pp. 715–724. ACM, 2009.

10 X. Chen. Complexity dichotomies of counting problems. *SIGACT News*, 42:54–76, 2011.

11 X. Chen, M. Dyer, L. A. Goldberg, M. Jerrum, P. Lu, C. McQuillan and D. Richerby. The complexity of approximating conservative counting CSPs. CoRR eprint, arXiv:abs/1208.1783, 2012.

12 D. Cohen, M. Cooper and P. Jeavons. Generalising submodularity and Horn clauses: Tractable optimization problems defined by tournament pair multimorphisms. *Theor. Comput. Sci.*, 401:36–51, 2008.

13 D. Cohen, M. Cooper, P. Jeavons and A. Krokhin. Soft constraints: Complexity and multimorphisms. In *Proc. 9th CP*, LNCS 2833, pp. 244–258. Springer, 2003.

14 N. Creignou and M. Hermann. Complexity of generalized satisfiability counting problems. *Inform. Comput.*, 125:1–12, 1996.

15 M. Dyer, L. A. Goldberg, C. Greenhill and M. Jerrum. The relative complexity of approximate counting problems. *Algorithmica*, 38:471–500, 2004.

16 M. Dyer, L. A. Goldberg and M. Jerrum. The complexity of weighted Boolean CSP. *SIAM J. Comput.*, 38:1970–1986, 2009.

17 M. Dyer, L. A. Goldberg and M. Jerrum. An approximation trichotomy for Boolean #CSP. *J. Comput. Syst. Sci.*, 76:267–277, 2010.

18 M. Dyer and D. Richerby. An effective dichotomy for the counting constraint satisfaction problem. *SIAM J. Comput.*, to appear. (Conference version in *STOC 2010*.)

19 L. A. Goldberg and M. Jerrum. Approximating the partition function of the ferromagnetic Potts model. *J. Assoc. Comput. Mach.*, to appear. (Conference version in *ICALP 2010*.)

20 V. Kolmogorov and S. Živný. The complexity of conservative valued CSPs. CoRR eprint, arXiv:abs/1110.2809, 2011. (Conference version in *SODA 2012*.)

21 P. Lu. Complexity dichotomies of counting problems. ECCC eprint, 18:93, 2011.

22 M. Mitzenmacher and E. Upfal. *Probability and Computing.* CUP, 2005.

23 R. Rudolf and G. J. Woeginger. The cone of Monge matrices: extremal rays and applications. *Math. Method Oper. Res.*, 42:161–168, 1995.

24 R. Takhanov. A dichotomy theorem for the general minimum cost homomorphism problem. CoRR eprint, arXiv:abs/0708.3226, 2007. (Conference version in *STACS 2010*.)

25 D. Topkis. Minimizing a submodular function on a lattice. *Oper. Res.*, 26:305–321, 1978.

26 T. Yamakami. Approximate counting for complex-weighted Boolean constraint satisfaction problems. *Inform. Comput.*, 219:17–38, 2012.

Lossy Chains and Fractional Secret Sharing

Yuval Ishai[*1], **Eyal Kushilevitz**[†2], **and Omer Strulovich**[‡3]

1 **Technion, Haifa, Israel**
`yuvali@cs.technion.ac.il`
2 **Technion, Haifa, Israel**
`eyalk@cs.technion.ac.il`
3 **Technion, Haifa, Israel**
`omers@cs.technion.ac.il`

Abstract

Motivated by the goal of controlling the amount of work required to access a shared resource or to solve a cryptographic puzzle, we introduce and study the related notions of *lossy chains* and *fractional secret sharing.*

Fractional secret sharing generalizes traditional secret sharing by allowing a fine-grained control over the amount of uncertainty about the secret. More concretely, a fractional secret sharing scheme realizes a fractional access structure $f : 2^{[n]} \to \{0, \ldots, m-1\}$ by guaranteeing that from the point of view of each set $T \subseteq [n]$ of parties, the secret is *uniformly* distributed over a set of $f(T) + 1$ potential secrets. We show that every (monotone) fractional access structure can be realized. For *symmetric* structures, in which $f(T)$ depends only on the size of T, we give an efficient construction with share size $\mathrm{poly}(n, \log m)$.

Our construction of fractional secret sharing schemes is based on the new notion of *lossy chains* which may be of independent interest. A lossy chain is a Markov chain $(X_0, \ldots, X_n)$ which starts with a random secret X_0 and gradually loses information about it at a rate which is specified by a *loss function* g. Concretely, in every step t, the distribution of X_0 conditioned on the value of X_t should always be uniformly distributed over a set of size $g(t)$. We show how to construct such lossy chains efficiently for any possible loss function g, and prove that our construction achieves an optimal asymptotic information rate.

1998 ACM Subject Classification E.3 Data Encryption

Keywords and phrases Cryptography, secret sharing, Markov chains

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.160

1 Introduction

In this work, we introduce and study two related notions: *lossy chains* and *fractional secret sharing.* We start by describing the latter.

Fractional secret sharing. Suppose that we wish to share a secret password between several parties, such that the largest subset of cooperating parties will be the first to guess the correct password. (Think of the password as a key which locks vault, where the number of guessing attempts measures the amount of work required for unlocking the vault.)

[*] Supported by the European Research Council as part of the ERC project CaC (grant 259426), ISF grant 1361/10, and BSF grant 2008411.
[†] Supported by ISF grant 1361/10 and BSF grant 2008411.
[‡] Supported by ERC grant 259426.

A simple solution that comes to mind is the following. If the password is a binary string of length k and we have $n \leq k$ parties, we can give each party one or more bits of the password. In this solution, a larger cooperating subset of parties will need a smaller expected number of attempts to guess the password than a smaller one. This solution achieves our goal, but is limited to specific parameters. For example, we cannot easily extend this method to $n > k$ parties, nor can we have finer control over the relative amount of expected work required by different subsets of parties.

Our goal is to find a solution which gives maximal control over the amount of information about the password revealed to each subset of parties. This motivates the notion of *fractional secret sharing*. In traditional secret sharing [9, 3, 6], each subset of n parties either has full information about the secret or has no information about the secret. Fractional secret sharing generalizes this notion by allowing a fine-grained control over the amount of uncertainty of each subset about a uniformly random secret. The uncertainty is specified by a *fractional access structure* $f : 2^{[n]} \to \{0, \dots, m-1\}$. A fractional secret sharing scheme realizing f should have the property that from the point of view of each set $T \subseteq [n]$ of parties, the secret is always *uniformly* distributed over a set of $f(T) + 1$ potential secrets. Since adding a party to a subset cannot reduce the amount of available information, we assume f to be *monotone* in the sense that if $T \subseteq T'$ then $f(T') \leq f(T)$. This raises the following questions:

Can every (monotone) fractional access structure be realized? If so, how efficiently?

How to gradually forget. Motivated in part by the problem of fractional secret sharing, we introduce the related notion of *lossy chains*. A lossy chain is a Markov chain $(X_0, \dots, X_n)$ which starts with a random secret X_0 and gradually loses information about it at a rate which is specified by a predefined loss function $g : [n] \to [m]$. More concretely, we require that for any $1 \leq i \leq n$ and any possible value x_i in the support of X_i, the distribution of the secret X_0, conditioned on the event that $X_i = x_i$, is uniform[1] over a set of size $g(i)$. (The identity of this set may depend on x_i.) In a similar manner to fractional access structures, we require that the loss function g be monotone, in the sense that for $i < j$ we have $g(i) < g(j)$. This raises the following questions:

Can every (monotone) loss function be realized? If so, how efficiently?

The Markov property of the chain (namely, the requirement that X_{i+1} be independent of $X_0, \dots, X_{i-1}$ given X_i) is important for our motivating applications, as it rules out the possibility of combining several values X_i in order to learn more information than that implied by the "best" value X_i. Jumping ahead, this property will turn out to be crucial for the construction of fractional secret sharing from lossy chains.

Why uniform? An important aspect of our notions of fractional secret sharing and lossy chains is that they require each conditional distribution to always be *uniform* over a subset of potential secrets having a specified size. One could instead consider alternative definitions which only specify some measure of *entropy*, such as conditional Shannon entropy [10], or min-entropy [8], without further restricting the distribution. Insisting on a uniform distribution has several important advantages. First, a crude measure of uncertainty such as entropy is not informative enough to capture all relevant properties of a distribution.

[1] The uniformity requirement rules out simple solutions that are based on gradually adding independent random noise to the initial secret (cf. [4]), see further discussion below.

For instance, min-entropy determines the best probability of guessing the secret in the first attempt, but says little about the expected number of attempts until the secret is correctly guessed. Second, using the uniform distribution does not only give control over the expected number of attempts in an optimal guessing strategy, but it also minimizes the *variance* of the number of such attempts under the expectation constraint. (See full version for a proof that the uniform distribution beats any other distribution in this respect.) Finally, in some scenarios it may be desirable to spread the point in time in which the secret is correctly guessed as evenly as possible (think of a password controlling a shared resource). This too is achieved optimally by the uniform distribution. We note that one could relax the requirement of uniformity to being statistically close to uniform. This is addressed in the full version.

1.1 Our Results

We obtain several positive and negative results about lossy chains and fractional secret sharing.

- We show that any monotone loss function $g : [n] \to [m]$ can be efficiently realized by a lossy chain $(X_0, \ldots, X_n)$ in which the bit-length of each X_i is at most $n \cdot \lceil \log m \rceil$. Moreover, we show this bound to be asymptotically tight by demonstrating the existence of a family of loss functions $g_{n,m} : [n] \to [m]$ for which some X_i must be $\Omega(n \log m)$ bits long. This asymptotic lower bound still holds even if we allow the conditional distributions to have negligible statistical distance from uniform. Settling for a constant statistical distance, the bit-length of each X_i can be $O(\log^2 m)$, independently of n.
- We show a general reduction of fractional secret sharing to lossy chains, which implies that every monotone fractional access structure $f : 2^{[n]} \to \{0, \ldots, m-1\}$ can be realized. For the important case of *symmetric* structures, in which $f(T)$ depends only on the size of T, we get an efficient construction in which the share size of each party is at most $n \cdot \lceil \log \max\{n, m\} \rceil$.

1.2 Overview of Techniques

Recall that a lossy chain is a Markov chain $(X_0, \ldots, X_n)$, where X_0 is a random secret, and each step loses additional information about the secret. This loss is specified by a loss function $g : [n] \to [m]$, such that for each $1 \leq i \leq n$ and x_i in the support of X_i, the distribution of X_0 conditioned on $X_i = x_i$ is uniform over a set of size $g(i)$. (See Section 3.1 for a formal definition.)

As a simple warmup example, let X_0 be uniform over $\{0,1\}^n$, and let X_i include the first $n - i$ bits of X_0. In this case, X_0 conditioned on $X_i = x_i$ is distributed uniformly over a set of size 2^i. Thus, this lossy chain realizes the loss function $g(i) = 2^i$. This simple approach only works for a loss function g which is increasing exponentially, and can be generalized only to loss functions g such that $g(i)$ divides $g(i+1)$.

The following alternative approach works for any monotone loss function $g : [n] \to [m]$, where without loss of generality $g(n) = m$:

1. Pick x_0 uniformly from $[m]$.
2. For $i = 1, \ldots, n$, pick (a set) x_i uniformly at random from all subsets of $[m]$ of size $g(i)$ containing x_{i-1}.
3. Output $(x_0, x_1, \ldots, x_n)$

Intuitively, this method starts from a set $\{x_0\}$ containing only the correct secret, and in each step adds $g(i) - g(i-1)$ new random "distractors" from $[m]$. This allows us to realize a lossy chain for any loss function. However, storing or sending the values of such a chain may be infeasible when m is large (e.g., think of m as the number of possible passwords). It is therefore desirable to get a solution in which the bit-length of each X_i grows logarithmically with m instead of linearly with m.

A natural approach is to limit the subsets represented by X_i to only be discrete intervals of the form $[j, k] = \{j, j+1, \ldots, k\}$, where $1 \leq j \leq k \leq m$. Unfortunately, this simple modification of the previous construction fails to satisfy the uniform conditional distribution property. More concretely, given an interval $[j, k]$ for X_i, the probability of the secret X_0 being in the middle of the interval will be higher than in the edges of the interval. To avoid this problem, we employ *cyclic* intervals. Intuitively, given an arbitrary ordered set, a cyclic interval can "cycle" through the end back to the start of the set. Using recursive nesting of such cyclic intervals, we construct a lossy chain for any loss function while keeping the support of each X_i small. We describe our results for lossy-chains in Section 3. We present the above construction in Section 3.2, and we establish the optimality of this construction in Section 3.3 by using some basic linear algebraic properties of the probability vectors associated with a lossy chain. Positive and negative results for the statistical relaxation of lossy chains are given in the full version.

Finally, in Section 4 we describe the reduction of fractional secret sharing to lossy chains. Recall that a fractional secret sharing scheme realizes a fractional access structure $f : 2^{[n]} \to \{0, \ldots, m-1\}$ by ensuring that from the point of view of each set $T \subseteq [n]$ of parties, the secret is *uniformly* distributed over a set of $f(T)+1$ potential secrets. (See Section 4.1 for a formal definition.) In the case of a symmetric structure f, where $f(T)$ depends only on the size of T, we can use the following natural construction: let $g(i) = f([n-i]) + 1$ and let $(X_0, \ldots, X_n)$ be a lossy chain realizing g. A fractional secret sharing scheme realizing f can be obtained by using a threshold secret sharing scheme (such as Shamir's scheme [9]) to distribute the value of each X_i between the n parties with reconstruction threshold $n-i$. Any set T of t parties will be able to reconstruct the values $X_{n-t}, \ldots, X_n$, which by the Markov property contain the same information about the secret X_0 as X_{n-t}. By the definition of g, the distribution of X_0 conditioned on the value of X_{n-t} is uniform over a set of size $f(T) + 1$, as required. The above construction can be generalized to arbitrary fractional access structures. However, similarly to traditional secret sharing, the complexity of the general construction may be exponential in the number of parties.

Related work. The notion of fractional secret sharing can be viewed as a restricted instance of *non-perfect* secret sharing (also referred to as *ramp secret sharing*). While in standard (perfect) secret sharing schemes each set of players should either be able to fully reconstruct the secret or alternatively should learn nothing about it, in a non-perfect secret sharing there is also a third kind of sets that may learn partial information about the secret. Non-perfect schemes were proposed mainly for the reason of improving the efficiency of secret sharing by reducing the size of the shares. Unlike fractional secret sharing, in non-perfect secret sharing there is no requirement on the type of partial information available to the third kind of subsets. For works on non-perfect secret sharing, see [2, 11, 7, 5] and references therein.

2 Preliminaries

Notation. We let $[n]$ denote the set of integers $\{1, 2, \ldots, n\}$. We use $\log n$ to denote $\log_2 n$. For a random variable X, we let $\mathsf{supp}(X)$ denote the support set of X, that is, the set of values which X may take with nonzero probability. The support set of a real-valued vector is the set of coordinates in which it takes nonzero values.

Markov chains. A Markov chain is a sequence of random variables such that the distribution of each variable in the sequence depends only on the value of the previous variable. Formally:

▶ **Definition 1. (Markov chain)** Let $\bar{X} = (X_0, X_1, \ldots, X_n)$ be a sequence of jointly distributed random variables. We say that $\bar{X}$ is a *Markov chain* if for every $i \in [n]$ and for any sequence of values $x_0 \in \mathsf{supp}(X_0), \ldots, x_i \in \mathsf{supp}(X_i)$,

$$\Pr\left[X_i = x_i | X_{i-1} = x_{i-1}\right] = \Pr\left[X_i = x_i | X_{i-1} = x_{i-1}, \ldots, X_0 = x_0\right].$$

In general, Markov chains can be defined as infinite sequences of random variables with infinite support size. However, in this work we will only consider finite Markov chains.

The above definition is equivalent to requiring that for any i and x_i in the support of X_i, the random variables $(X_1, \ldots, X_{i-1})$ and $(X_{i+1}, \ldots, X_n)$ are independent conditioned on $X_i = x_i$. The symmetry of the above conditional independence requirement implies the following "reversibility" property of Markov chains:[2]

▶ **Fact 1.** *If $\bar{X} = (X_0, X_1, \ldots, X_n)$ is a Markov chain, then so is $\bar{X}^R = (X_n, X_{n-1}, \ldots, X_0)$.*

3 Lossy Chains

In this section we define our new notion of a lossy chain (Section 3.1), present an efficient construction of lossy chains (Section 3.2), and prove a lower bound on their efficiency (Section 3.3).

3.1 Definitions and Basic Properties

A lossy chain is a Markov chain in which the information loss about the initial value is fully specified by a *loss function*. We start by defining the latter.

▶ **Definition 2. (Loss function)** A *loss function* is a monotone increasing function $g : [n] \to [m]$. That is, for every $1 \leq i < j \leq n$ we have $g(i) < g(j)$.

We now turn to define lossy chains.

▶ **Definition 3. (Lossy chain)** Let $g : [n] \to [m]$ be a loss function, and let $\bar{X} = (X_0, X_1, \ldots, X_n)$ be a sequence of random variables. We say that $\bar{X}$ is a *lossy chain realizing* g if the following conditions hold:

- $\bar{X}$ is a Markov chain, and
- for every $i \in [n]$ and every x_i in the support of X_i, the distribution of X_0 conditioned on $X_i = x_i$ is uniform over a set of size $g\left(i\right)$.

Our goal is to construct lossy chains in which each value can be succinctly described. To this end we use the following measure of efficiency.

▶ **Definition 4. (Information rate)** Let $\bar{X} = (X_0, X_1, \ldots, X_n)$ be a lossy chain. The *information rate* of $\bar{X}$ is defined as

$$\rho(\bar{X}) = \min_{0 \leq i \leq n} \frac{\log g(n)}{\log |\mathsf{supp}\left(X_i\right)|}$$

[2] For a formal proof see [1, p. 215].

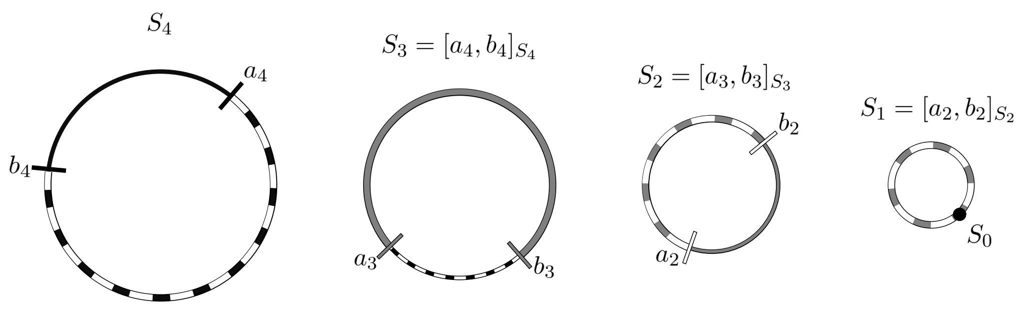


Figure 1 The cyclic interval from a_4 to b_4 is taken from a set S_4 with a cyclic order to create S_3. Then S_2 is created as a subset of S_3 by taking another cyclic interval. This goes on, until S_0, the starting value is chosen from S_1.

It will be convenient to assume that in a lossy chain realizing $g : [n] \to [m]$, the initial value X_0 is uniformly distributed over a set of size $g(n)$, and X_n has support set of size 1. The following claim shows that this assumption is without loss of generality: any lossy chain realizing g can be converted into a canonical form that has this property and has the same or better information rate.

▶ **Claim 1. (Canonical lossy chain)** *Let $g : [n] \to [m]$ be a loss function and let $\bar{X} = (X_0, X_1, \dots, X_n)$ be a lossy chain realizing g. Let x_n be an arbitrary element in the support of X_n. Let $\bar{X}' = (X'_0, X'_1, \dots, X'_n)$ be the joint distribution defined by*

$$\Pr[\bar{X}' = (x'_0, x'_1, \dots, x'_n)] = \Pr[\bar{X} = (x'_0, x'_1, \dots, x'_n) \mid X_n = x_n].$$

Then $\bar{X}'$ is a lossy chain realizing g. Moreover, X'_0 is uniform over a set of size $g(n)$ and $\mathsf{supp}(X'_i) \subseteq \mathsf{supp}(X_i)$ for $0 \le i \le n$.

3.2 An Efficient Construction

In the Introduction, we have seen a simple general construction of a lossy chain realizing $g : [n] \to [m]$ whose information rate is $\tilde{\Theta}(1/m)$. This construction may be infeasible for large values of m. In this section, we show how the rate can be improved to $1/n$.

We first recall the scheme described in the Introduction. Given $g : [n] \to [m]$ where (without loss of generality) $g(n) = m$, the lossy chain is computed as follows.

1. Pick x_0 uniformly from $[m]$.
2. For $i = 1, \dots, n$, pick a set x_i uniformly at random from all subsets of $[m]$ of size $g(i)$ containing x_{i-1}.
3. Output $(x_0, x_1, \dots, x_n)$.

This chain is inefficient in that it requires to store arbitrary subsets of $[m]$. In order to obtain a more efficient variant of this construction, we restrict these subsets to be nested *cyclic intervals*.

▶ **Definition 5. (Cyclic interval)** Let $S = \{e_0, \dots, e_{m-1}\}$ be a linearly ordered set, where $e_0 < e_1 < \dots < e_{m-1}$. For any two integers $a, b \in \{0, \dots, m-1\}$, the *cyclic interval from a to b over S*, denoted $[a, b]_S$, is defined by:

```
"Cyclic Intervals" lossy chain

Input: A loss function $g : [n] \to [m]$.

Algorithm:
1. $S_n \leftarrow [g(n)]$
2. For $i = n-1, ..., 1$

    a. Pick $a_i \in \{0, ..., g(i) - 1\}$ uniformly at random
    b. $b_i \leftarrow (a_i + g(i) - 1) \bmod g(i+1)$
    c. set $S_i = [a_i, b_i]_{S_{i+1}}$

3. Pick $x_0$ uniformly at random from $S_1$

Output: $\bar{x} = (x_0, S_1, S_2, \ldots, S_n)$
```

Figure 2 Lossy chain obtained via nested cyclic intervals

$$[a,b]_S = \begin{cases} \{e_a, \ldots, e_b\} & a \le b \\ \{e_a, \ldots, e_{m-1}\} \cup \{e_0, \ldots e_b\} & a > b \end{cases}$$

Note that for a given size k, there are exactly $|S|$ distinct nested intervals of size k in S, one for each starting point a. The algorithm for generating the lossy chain iteratively generates subsets S_i of size $g(i)$ for every $i \in [n]$ in *decreasing order*, where each subset S_i is a random cyclic interval in S_{i+1}. See Figure 1 for a visual illustration. A precise description of the algorithm is given in Figure 2.

We now prove that the output of the "Cyclic Intervals" algorithm from Figure 2 forms a lossy chain realizing g. In the following, we denote by $\bar{X} = (X_0, \ldots, X_n)$ the joint distribution of the output. We start by showing that the output indeed forms a Markov chain.

▶ **Lemma 1.** *The output distribution $(X_0, \ldots, X_n)$ forms a Markov chain.*

Proof. For $1 \le i \le n$, the output X_{i-1} is sampled based on X_i alone. This implies that $(X_n, \ldots, X_0)$ is a Markov chain and, by Fact 1, we have that $(X_0, \ldots, X_n)$ is also a Markov chain. ◀

▶ **Lemma 2.** *The chain $\bar{X}$ realizes the loss function g.*

Proof. We prove that for any $1 \le i \le n$ and any $S_i \in \mathsf{supp}(X_i)$, the distribution of X_0 conditioned on the event $X_i = S_i$ is distributed uniformly over S_i. Since $|S_i| = g(i)$ the lemma will follow.

The above claim intuitively follows by symmetry. We formally prove it by induction on i. The case $i = 1$ follows directly from the algorithm's description. Suppose the claim holds for i, and let S_{i+1} be in the support of X_{i+1}. We need to prove that X_0 conditioned on $X_{i+1} = S_{i+1}$ is uniformly distributed over S_{i+1}. Indeed, when $X_{i+1} = S_{i+1}$ the choice of the output x_0 can be viewed as resulting from the following two step process:

1. Pick S_i as a random cyclic interval in S_{i+1} of size $g(i)$.
2. Pick x_0 from the distribution of X_0 conditioned on $X_i = S_i$.

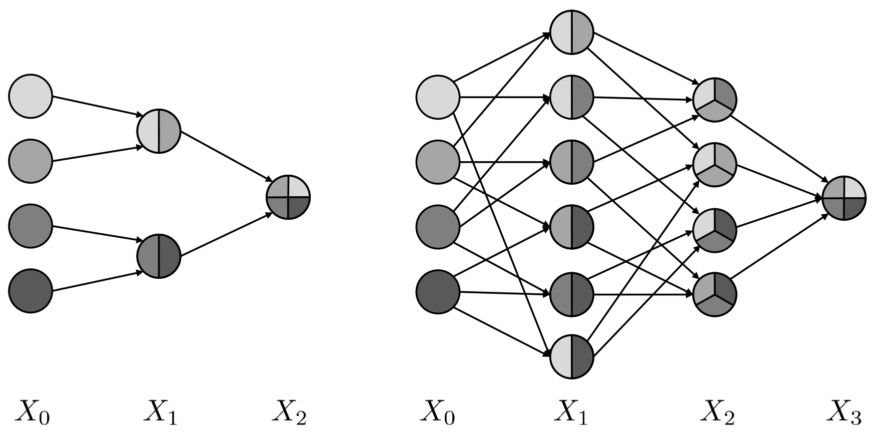


Figure 3 On the left, a states graph for a simple lossy chain realizing $g\left(i\right) = 2^i$ with 4 possible starting values. On the right, a lossy chain realizing $g\left(i\right) = i + 1$, also with 4 possible starting values.

The choice of S_i in the first step guarantees that each $x \in S_{i+1}$ has an equal probability to be in S_i. By the induction's hypothesis, the second step picks x_0 uniformly at random from S_i. Combining the two steps, x_0 is uniformly distributed over S_{i+1}, as required. ◀

Using the above two lemmas, we obtain the main theorem of this section.

▶ **Theorem 3.** *For any loss function $g : [n] \to [m]$, there is a lossy chain realizing g whose information rate is at least $\frac{1}{n-1}$.*

▶ Remark. **(On computational efficiency)** *The description in Figure 2 does not address the question of how the sets S_i are represented and how one can efficiently enumerate the elements of S_i or sample from S_i. To this end, we note that if we modify the algorithm such that X_i contains the representation of S_i by the sequence $(a_{n-1}, \ldots, a_i)$, the resulting chain still realizes g (namely, the additional information provided by this sequence does not change the distribution of X_0 conditioned on S_i). Moreover, the information rate of this (slightly redundant) representation of the sets S_i is still lower bounded by $1/(n-1)$.* See full version for efficient algorithms supporting this representation.

3.3 A Negative Result

In this section, we establish a limitation on the information rate of lossy chains, showing that the cyclic intervals construction cannot be asymptotically improved in the worst case. Specifically, we show a family of loss functions $g : [n] \to [m]$ for which the support size of each X_i is at least $\binom{m}{m-n+i}$. For proving this result, it is convenient to use the following notion of a *states graph* of a Markov chain.

▶ **Definition 6. (States graph)** Let $\bar{X} = (X_0, \ldots, X_n)$ be a Markov chain, and let V_i denote the support set of X_i. Assume, without loss of generality, that the sets V_i are pairwise disjoint. The *states graph* of $\bar{X}$ is a weighted directed graph (G, f) where

- $G = (V, E)$ is a layered graph in which V_i is the set of i-th level nodes and E contains the edges (u, v) such that, for some i, we have $u \in V_i$, $v \in V_{i+1}$ and v is in the support of X_{i+1} conditioned on $X_i = u$.
- For any $u \in V_i$ and $v \in V_{i+1}$, we have $f(u, v) = \Pr\left[X_{i+1} = v | X_i = u\right]$.

An example for a states graph of a simple lossy chain appears in Figure 3.

We define, for each node $v \in V\backslash V_0$, a probabilities vector which contains the probability of each starting value given that X_j was chosen to be v.

▶ **Definition 7.** **(Probabilities vector)** Let $g : [n] \to [m]$ be a loss function such that $g(n) = m$. Let $\bar{X} = (X_0, \dots, X_n)$ be a lossy chain realizing g with $|\mathsf{supp}(X_0)| = m$, and let G be its states graph. Let $v \in V_j$ be a node in layer j of G. The *probabilities vector* of v is a vector $\bar{v} \in \mathbb{R}^m$ such that $\bar{v}[i] = \Pr[X_0 = e_i | X_j = v]$, where e_i is the element with index i in V_0, and $\bar{v}[i]$ is the ith coordinate of $\bar{v}$. We say that a vector $\bar{u} \in \mathbb{R}^m$ *fits* layer j of G if $\bar{u}$ has $g(j)$ entries of value $\frac{1}{g(j)}$ and the other entries are 0.

Note that if $\bar{v}$ is the probabilities vector of a node $v \in V_j$, then $\bar{v}$ necessarily fits layer j. However, the converse is not necessarily true.

Our negative result relies on the fact that the probabilities vector of any node in the states graph is a convex linear combination of the probabilities vectors of its parents (that is, a linear combination with positive coefficients that add up to 1).

▶ **Lemma 4.** *Let $\bar{X} = (X_0 \dots, X_n)$ be a lossy chain with states graph $G = (V, E)$. For any $1 \le j \le n$, let $v \in V_j$ be a node of G and $u_1, \dots, u_k \in V_{j-1}$ be all the nodes such that $(u_i, v) \in E$. Then $\bar{v}$, the probabilities vector of v, is a convex linear combination of $\bar{u}_1, \dots, \bar{u}_k$, the probabilities vectors of $u_1, \dots, u_k$.*

The main theorem of this section shows a tight negative result on the efficiency of a lossy chain realizing a concrete family of loss functions.

▶ **Theorem 5.** *Let m, n be positive integers such that $m \ge n$ and let $g_{m,n} : [n] \to [m]$ be the loss function defined by $g_{m,n}(i) = m - n + i$. Let $(X_0, \dots, X_n)$ be a lossy chain realizing $g_{m,n}$. Then, for any $0 < i \le n$, it holds that $|\mathsf{supp}(X_i)| \ge \binom{m}{m-n+i}$.*

The theorem relies on the following technical lemma, which will imply a lower bound on the number of probabilities vectors from level i required to span a probabilities vector from level $i + 1$.

▶ **Lemma 6.** *Let $\bar{v} \in \mathbb{R}^n$ be a 0-1 vector of Hamming weight k. Let $\bar{u}_1, \dots, \bar{u}_m$ be 0-1 vectors of Hamming weight $k - 1$. If $\bar{v}$ is a linear combination of $\bar{u}_1, \dots, \bar{u}_m$ with positive coefficients, then $m \ge k$.*

We are now ready to prove Theorem 5.

Proof. Let $\bar{X} = (X_0, \dots, X_n)$ be a lossy chain realizing $g_{m,n}$. By Claim 1, we may assume without loss of generality that X_0 is uniform over a set of size $g_{m,n}(n) = m$ and X_n has support of size 1.

Let $V_0, \dots, V_n$ be the layers in the states graph of $\bar{X}$. We prove by induction that, for any $i \in [n]$ and for any of the $\binom{m}{m-n+i}$ probabilities vectors $\bar{v}$ which fit layer i, there is a node $v \in V_i$ such that $\bar{v}$ is the probabilities vector of v. The base case is $i = n$. In this case, the probabilities vector of the (single) node in V_n is $(1/m, \dots, 1/m)$, which is the only vector which fits level n.

We now assume that the claim holds for layer $i + 1$ and prove it for layer i. Let $\bar{u}$ be a vector which fits layer i. By the induction hypothesis, we know that for any vector $\bar{v}$ which fits layer $i + 1$ there is a corresponding node $v \in V_{i+1}$. Let $v \in V_{i+1}$ be such a node for which the support set of $\bar{v}$ contains that of $\bar{u}$. By Lemma 4, $\bar{v}$ is a convex linear combination of the probability vectors of its parents u_i. Note that each probabilities vector of a parent node u_i

is a scalar multiple of a 0-1 vector of weight $g_{m,n}(i)$ whereas $\bar{v}$ is a scalar multiple of a 0-1 vector of weight $g_{m,n}(i+1)$. By Lemma 6 and the fact that $g_{m,n}(i+1) = g_{m,n}(i) + 1$, the probability vectors $\bar{u}_i$ of the parent nodes u_i have support sets that cover *all* $m - n + i + 1$ subsets of size $m - n + i$ of the support set of $\bar{v}$. In particular, one of the $\bar{u}_i$ must coincide with $\bar{u}$. Since the above holds for any $\bar{u}$ which fits layer i, this concludes the proof of the claim and the theorem. ◀

Using loss functions of the form $g\left(i\right) = m - n + i$, we get the following corollary.

▶ **Corollary 7.** *For every $\varepsilon > 0$ there is an infinite family of loss functions $g_n : [n] \to [m(n)]$, where $m(n) = \lceil n^{1+\varepsilon} \rceil$, such that the information rate of any lossy chain realizing g_n is $O\left(\frac{1}{n}\right)$.*

4 Fractional Secret Sharing

In this section, we define the notion of *fractional secret sharing* and show how to realize it via the use of lossy chains.

4.1 Definitions

An instance of the fractional secret sharing problem is specified by a *fractional access structure*. Recall that a traditional access structure specifies which subsets of parties can reconstruct the secret, where the remaining sets of parties should learn nothing about the secret. A fractional access structure generalizes this by allowing full control on the amount of information learned by each set of parties.

▶ **Definition 8. (Fractional access structure)** Let $P = \{p_1, \ldots, p_n\}$ be a finite set of parties and let m be an integer. A function $f : 2^P \to \{0, \ldots, m-1\}$ is *monotone* if $B \subseteq C$ implies that $f(B) \geq f(C)$. A *fractional access structure* is a monotone function $f : 2^P \to \{0, \ldots, m-1\}$, with $f\left(\emptyset\right) = m - 1$. We say that f is *symmetric* if $f(B)$ depends only on $|B|$.

We note that if we limit the range of f to $\{0, m-1\}$ then f corresponds to a traditional access structure. We now formally define our notion of fractional secret sharing.

▶ **Definition 9. (Fractional secret sharing scheme)** Let $f : 2^P \to \{0, \ldots, m-1\}$ be a fractional access structure and let S be a finite secret-domain. Let D be a randomized algorithm which outputs a uniformly random $s \in S$ together with an n-tuple of shares $(s_1, \ldots, s_n)$. We say that D is a *fractional secret-sharing scheme realizing f with secret-domain S* if there exists a positive integer k such that the following holds: For every $Q \subseteq P$, and any possible share vector s_Q of parties in Q, the distribution of s conditioned on the event that parties in Q receive the shares s_Q is uniform over a subset of S of size $f\left(Q\right) \cdot k + 1$. If the above holds with $k = 1$, we say that D *strictly realizes f*.

Note that our default notion of realizing a fractional access structure views the structure as only specifying a kind of *ratio* between the amount of uncertainty of different sets, without specifying the absolute amount of uncertainty or the size of the secret-domain. This relaxation is needed in order to capture standard access structures as a special case. Also note that the above definition generates a random secret along with the shares, unlike most traditional definitions of secret sharing which do not refer to any particular distribution over the secret domain. As in the case of traditional secret sharing, we measure the complexity by comparing the size of the biggest share-domain to the size of the secret-domain.

4.2 Fractional Secret Sharing from Lossy Chains

We now apply the positive results from Section 3.2 towards realizing any fractional access structure.

▶ **Theorem 8.** *For any fractional access structure $f : 2^P \to \{0, \ldots, m-1\}$, there exists a fractional secret sharing scheme which strictly realizes f.*

Proof. Without loss of generality, assume that $f(P) = 0$ and $f(\emptyset) = m - 1$. We shall use $S = [m]$ as the secret-domain. Let $\alpha_0, ..., \alpha_l$ be all the different values in the range of f in increasing order; that is, $\alpha_0 < ... < \alpha_l$. By our assumptions, we have $\alpha_0 = 0$ and $\alpha_l = m - 1$. Define a loss function $g : [l] \to [m]$ such that $g(i) = \alpha_i + 1$ and let $\bar{X}$ be a lossy chain realizing g. The share generation algorithm D can now proceed as follows:

1. Sample values $(x_0, \ldots, x_l)$ from $\bar{X}$ and let $s = x_0$;
2. For every subset of parties $Q \subseteq P$, let $f(Q) = \alpha_j + 1$. Use a traditional $|Q|$-out-of-$|Q|$ secret sharing scheme to share x_j into $s_{Q,1}, \ldots, s_{Q,|Q|}$ (e.g., using additive secret sharing) and give the j-th party in Q the share $s_{Q,j}$.

We now show that D is a fractional secret sharing scheme strictly realizing f. Let $Q \subseteq P$ be a subset of parties. By the properties of the underlying $|Q|$-out-of-$|Q|$ scheme, the information available to parties in Q is equivalent to learning all values x_j such that $f(Q') = \alpha_j + 1$ for some $Q' \subseteq Q$. By the monotonicity of f this means the parties in Q learn x_i, where i is the index such that $f(Q) = \alpha_i + 1$, and possibly additional values x_j for $j > i$. By the Markov property of a lossy chain, the distribution of the secret s conditioned on the above values x_i and x_j is uniform over a set of size $g(i) = \alpha_i + 1 = f(Q)$, as required. ◀

We remark that if $f(P) \neq 0$, we can add another party p' to the set of parties and set $f(Q)$ to 0 for every subset Q containing p'. We can then execute the proposed algorithm and "throw away" all the shares of p'.

Similarly to traditional secret sharing, the size of the shares produced by the above algorithm can be exponential in the number of parties. This can be avoided in the case of *symmetric* fractional access structures.

▶ **Theorem 9.** *Let $f : 2^P \to \{0, \ldots, m-1\}$ be a symmetric fractional access structure with $f(\emptyset) = m - 1$. Then there exists a fractional secret sharing scheme D which (strictly) realizes f with secret-domain $[m]$, where the bit-length of each share is at most $n \cdot \lceil \log \max \{n, m\} \rceil$.*

Proof. As before, let $\alpha_1, \ldots, \alpha_l$ be all the different values in the range of f in increasing order and define $g : [l] \to [m]$ such that $g(i) = \alpha_i + 1$. We now define D as follows:

1. Generate values $\bar{x} = (x_0, \ldots, x_l)$ for the cyclic intervals lossy chain realizing g, and let $s = x_0$. Furthermore, let $a_1, \ldots, a_{l-1}$ be the starting values of the cyclic intervals defining $\bar{x}$ (see Remark 3.2).
2. For every $i \in [n]$, let α_j be the value such that for any subset of parties $Q \subseteq P$ of size i we have $f(Q) = \alpha_j + 1$. Use Shamir's i-out-of-n threshold secret sharing scheme to create shares of a_j and give one share to each of the parties in P.

We now show that D is a fractional secret sharing scheme. For every subset of parties $Q \subseteq P$, the parties can reconstruct all the values out of $x_0, \ldots, x_n$ that were shared in a threshold scheme requiring $|Q|$ or less parties. This means that if $f(Q) = \alpha_j + 1$, the parties of Q can reconstruct $a_j, \ldots, a_l$. By the definition of the cyclic intervals lossy chain, the parties can reconstruct $x_j, \ldots, x_l$ from $a_j, \ldots, a_l$ and since $x_j, \ldots, x_l$ were generated as values from a lossy chain realizing g we see that the secret s conditioned on $X_j = x_j, \ldots, X_l = x_l$ is distributed uniformly over a set of size $\alpha_j + 1$, where $\alpha_j + 1 = f(Q)$ as required.

We are left with showing that the size of share for each party is no more than $n \cdot \lceil \log(\max\{n, m\}) \rceil$. Each party receives n different shares, one from each invocation of the threshold secret sharing algorithm done by D. The secrets shared are $a_1, \ldots, a_l$ where we recall that all of them are values picked from at most m values. Using Shamir's threshold secret sharing scheme, each of the values is shared with shares of size $\lceil \log(\max\{n, m\}) \rceil$. This amounts to a share size of at most $n \cdot \lceil \log(\max\{n, m\}) \rceil$ for each party, as required. ◀

5 Conclusions and Open Questions

We introduced the notion of lossy chains – Markov chains which gradually lose information about an initial secret in a controlled fashion. We presented an efficient construction of lossy chains and a matching negative result on the efficiency of lossy chains. Finally, we have shown how lossy chains can be used to realize fractional secret sharing, a natural generalization of traditional secret sharing which supports a fine-grained control over the amount of uncertainty about the secret.

While we essentially settle the main complexity question about lossy chains, it remains open to obtain a characterization of the best achievable information rate for a given loss function g.

The most interesting open question regarding the complexity of fractional secret sharing is to settle the case of *symmetric* fractional access structures, which naturally generalize threshold access structures. While the latter can be realized by an ideal scheme in which the size of each share is equal to the size of the secret (for a sufficiently large secret), we do not know whether an analogous result holds in the fractional domain.

References

1 D. Bertsekas and R. Gallager. Data networks, 1992.

2 G. Blakley and C. Meadows. Security of ramp schemes. In *Proceedings of Crypto '84*, pages 242–268, 1984.

3 G. R. Blakley. Safeguarding cryptographic keys. In *National Computer Conference*, page 313. AFIPS Press, 1979.

4 R. Cleve. Controlled gradual disclosure schemes for random bits and their applications. In *Proceedings of Crypto '89*, pages 573–588, 1989.

5 O. Farràs and C. Padró. Extending brickell-davenport theorem to non-perfect secret sharing schemes. *IACR Cryptology ePrint Archive*, 2012:595, 2012.

6 M. Ito, A. Saito, and T. Nishizeki. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan*, 72(9):56–64, 1989.

7 K. Kurosawa, K. Okada, K. Sakano, W. Ogata, and S. Tsujii. Nonperfect secret sharing schemes and matroids. In *Proceedings of EUROCRYPT '93*, pages 126–141, 1993.

8 A. Rényi. On measures of entropy and information. In *Fourth Berkeley Symposium on Mathematical Statistics and Probability*, pages 547–561, 1961.

9 A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

10 C.E. Shannon. Communication Theory of Secrecy Systems. *Bell Systems Technical Journal*, 28(4):656–715, 1949.

11 H. Yamamoto. Secret sharing system using (k, L, n) threshold scheme. *Electronics and Communications in Japan (Part I: Communications)*, 69(9):46–54, 1986.

Two Hands Are Better Than One (up to constant factors): Self-Assembly In The 2HAM vs. aTAM

Sarah Cannon[*1], Erik D. Demaine[†2], Martin L. Demaine[‡3], Sarah Eisenstat[§4], Matthew J. Patitz[¶5], Robert T. Schweller[‖6], Scott M. Summers[7], and Andrew Winslow[**8]

1 Mathematical Institute, University of Oxford, 24-29 St. Giles', Oxford OX1 3LB, UK sarah.cannon@linacre.ox.ac.uk.

8 Department of Computer Science, Tufts University, Medford, MA 02155, USA, {scanno01,awinslow}@cs.tufts.edu.

2,3,4 Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 32 Vassar St., Cambridge, MA 02139, USA, {edemaine,mdemaine,seisenst}@mit.edu.

5 Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR 72701, USA, patitz@uark.edu.

6 Department of Computer Science, University of Texas–Pan American, Edinburg, TX, 78539, USA, rtschweller@utpa.edu.

7 Department of Computer Science and Software Engineering, University of Wisconsin–Platteville, Platteville, WI 53818, USA, summerss@uwplatt.edu

Abstract

We study the difference between the standard seeded model (aTAM) of tile self-assembly, and the "seedless" two-handed model of tile self-assembly (2HAM). Most of our results suggest that the two-handed model is more powerful. In particular, we show how to simulate any seeded system with a two-handed system that is essentially just a constant factor larger. We exhibit finite shapes with a busy-beaver separation in the number of distinct tiles required by seeded versus two-handed, and exhibit an infinite shape that can be constructed two-handed but not seeded. Finally, we show that verifying whether a given system uniquely assembles a desired supertile is co-NP-complete in the two-handed model, while it was known to be polynomially solvable in the seeded model.

1998 ACM Subject Classification F.1.2

Keywords and phrases abstract tile assembly model, hierarchical tile assembly model, two-handed tile assembly model, algorithmic self-assembly, DNA computing, biocomputing, Wang tiles

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.172

[*] Research supported in part by National Science Foundation grants CCF-0830734 and CBET-0941538.
[†] Research supported in part by NSF grant CDI-0941538.
[‡] Research supported in part by NSF grant CDI-0941538.
[§] Research supported in part by NSF grant CDI-0941538.
[¶] Research supported in part by National Science Foundation grant CCF-1117672.
[‖] Research supported in part by National Science Foundation grant CCF-1117672.
[**] Research supported in part by National Science Foundation grants CCF-0830734 and CBET-0941538.

1 Introduction

Algorithmic self-assembly is a burgeoning area that studies how to computationally design geometric systems of simple parts that self-assemble into desired complex shapes or functionalities. The field began with Erik Winfree's PhD thesis [17] and two STOC papers about a decade ago [2, 15]. The theoretical models introduced in this work have since been implemented in real molecular systems using DNA tiles [5, 16]. From a practical perspective, these systems are exciting because they enable controlled manufacture of precise geometric objects at nanometer resolution (*nanomanufacture*). From a theoretical Computer Science perspective, this area is exciting because it offers a model of computation where the computer consists of geometric objects, which is challenging to work with because the allowed operations are highly constrained (simple, local interactions between the objects), yet there are many results classifying the difficulty of assembling many different shapes.

1.1 A tale of two models

Most work in algorithmic self-assembly uses the *abstract Tile Assembly Model* (*aTAM*) [2, 15, 17]. In this model, the core of a self-assembly system is a set of *Wang tiles*—unit squares with up to one *glue* (label) on each edge, with each type available in infinite supply. One such tile is marked as a *seed* (starting point) of a single assembly, and the model defines how tiles can repeatedly attach to this assembly (according to glue strengths and an overall temperature—see Section 2.1 for details), which ultimately becomes the (usually single) output of the system.

In reality, tiles mix in solution according to Brownian motion, and attractive forces cause them to fuse into larger assemblies. Presumably, the aTAM defines a seed tile to keep track of a single assembly instead of the many copies assembled in reality (as seen in the atomic force microscopy images in [5, 16]). However, as a side effect, the aTAM fails to capture the possibility that multiple assemblies grow (e.g., from multiple copies of the seed) and attach to each other, potentially making unintended assemblies not predicted by the aTAM. In addition, the ability to fuse larger assemblies in reality could potentially be exploited to design more efficient self-assembly systems for a desired shape. These possible discrepancies between the aTAM and reality are the topic of this paper.

The *Two-Handed Tile Assembly Model* (*2HAM*) [1, 7, 9, 10, 12, 13] (also known as *Hierarchical Self-Assembly* [6]) is essentially an unseeded generalization of the aTAM, in which any two assemblies (including but not limited to individual tiles) can fuse to each other. Instead of using seeds, the 2HAM defines the "output" of the system to consist of all assemblies that cannot be fused with any others possibly produced by the system. (See Section 2.2 for the definition.) This model captures the possibility of larger assemblies fusing together, although it remains to be studied whether it accurately models reality.[1]

1.2 Our results

The central problem addressed in this paper is to determine the difference in theoretical power between these two models of self-assembly: the aTAM and the 2HAM. In particular

[1] 2HAM does not model the "floppiness" of assemblies (i.e. non-rigidity), which may allow bending that prevents proper alignment of glues or shifting of potentially blocking portions between two larger assemblies. It also ignores the reduced speed and/or concentration of larger assemblies, which may substantially impact the time required for assembly.

Table 1 Summary of results for simulating the aTAM model using the 2HAM model.

aTAM systems	Simulating 2HAM systems
$\tau \in \{1, 2\}$	$\tau = 2$, scale factor 5(thm. 4.2)
$\tau = 3$	$\tau = 3$, scale factor 5(thm. 4.3)
$\tau \geq 4$	$\tau = 4$, scale factor 5(thm. 4.1)

Table 2 Summary of results showing separation between the aTAM and 2HAM with respect to tile complexity. The value of a cell denotes the tile complexity. Note that some of our results are asymptotic while others are exact complexities. The term *Finite assembly* refers to *finite self-assembly*, which is defined in Section 2. For infinite staircases, "Yes" means the structure does self-assemble, "No" means that it does not self-assemble and "Open" means the question is open. Note that, for table cells that do not contain a reference, the theorem and corresponding proof are omitted from this version of the paper due to space constraints.

	Loops		Staircases	Infinite staircases	
	$\tau = 1$	$\tau = 2$	$\tau = 2$	Finite assembly	Self-assembly
aTAM	$n + 5$	$n + 3$	2^n steps: $\Omega\left(\frac{n}{\log n}\right)$ (thm. 3.2)	No (thm. 3.6)	No (thm. 3.6)
2HAM	$2n + 2$	$\leq n + 3$	$2^{O(\text{running time of } M \text{ on } x)}$ steps: $O(\lvert Q\rvert + \lvert x\rvert)$ (thm. 3.3)	Yes ($\tau = 2$) (thm. 3.5)	Open

we show that, up to constant factors, many results in the standard aTAM can be converted to apply in the 2HAM. On the other hand, we show that the 2HAM enables substantially more efficient self-assembly systems in some cases than what is possible in the aTAM. We conclude that two hands are better than one, up to constant factors.

Our main results are the following (see Tables 1, 2, and 3 for additional results):

Simulation: [Section 4, Table 1]

1. Any aTAM system with temperature $\tau \geq 2$ can be simulated by a 2HAM system with the same temperature τ, which produces a 5×5 scaled version of the same shape plus a portion of a unit-thickness "coating".
2. Any aTAM system with temperature $\tau \geq 4$ can be simulated by a 2HAM system with a temperature of 4. Thus low-temperature 2HAM is at least as powerful as even high-temperature aTAM, up to constant-factor scale.

Separation: [Section 3, Table 2]

3. There is a shape that can be assembled in the aTAM at temperature $\tau = 1$ using $n + 5$ unique tile types but any 2HAM system in which the shape assembles at the same temperature requires $2n + 2$ unique tile types. At temperature $\tau = 2$, the same shape can be assembled in both models using $n + 3$ tile types.
4. There is a shape that can be assembled in the 2HAM using n tile types, while the number of tile types required for any aTAM assembly of the shape is (roughly) exponential in n. This result can be extended to show that there is a shape that can be built in the 2HAM using $O(n)$ tile types, but in the aTAM the same shape requires $BB(n)$ tile types, where $BB(n)$ is the busy beaver function.
5. There is an infinite shape that can self-assemble in the aTAM but not in the 2HAM. Note that this does not contradict our first simulation result because our simulation scales up the simulated system by a constant factor.
6. There is an infinite shape that can self-assemble (in a weaker sense) in the 2HAM but not in the aTAM.

Table 3 Complexities of assembly verification problems for the aTAM and 2HAM. The variable a denotes the size of an input assembly, and τ and t denote the temperature and tileset size for an input aTAM or 2HAM system. "UC" stands for uncomputable. Note that, for table cells that do not contain a reference, the theorem and corresponding proof are omitted from this version of the paper due to space constraints.

	Producible		**Unique Assembly**		**Unique Shape**	
	$\tau = 1$	$\tau = 2$	$\tau = 1$	$\tau = 2$	$\tau = 1$	$\tau = 2$
aTAM	$O(a)$		$O(a^2 + at)$ [3]		co-NPC	co-NPC [7]
2HAM	$O(at)$	$O(a^4)$ [11]	$O(ta^2 + at^2)$	co-NPC (thm. 5.1)	co-NP [7]	co-NPC [7]
	Terminal		**Finite Existence**		**Infinite Existence**	
	$\tau = 1$	$\tau = 2$	$\tau = 1$	$\tau = 2$	$\tau = 1$	$\tau = 2$
aTAM	$O(at)$ [3]		UC		UC	
2HAM	$O(at)$	UC	UC		Open	UC

Verification: [Section 5, Table 3]

7. It is co-NP-complete to determine whether a given 2HAM self-assembly system uniquely assembles a given 3D supertile (the Unique Assembly problem is co-NP-complete in the 2HAM), while the same problem is known to be polynomial time solvable for aTAM [3][2] (This result is the only one in 3D; all other results are in 2D.) We provide results for the complexity for five additional verification problems for the aTAM and the 2HAM.

This paper aims to be a first major step toward a thorough "complexity theory" for self-assembly. Like traditional complexity theory, there are several potential models for self-assembly, and we need to understand the relative power among these models. Even our definition of "simulation" is new in that it is the first to also handle the dynamics of systems such as the 2HAM, and we hope that it forms the foundation for further such results.

2 Preliminaries and notation

We work in the 2-dimensional discrete space $\mathbb{Z}^2$. Define the set $U_2 = \{(0,1),(1,0),(0,-1),(-1,0)\}$ to be the set of all *unit vectors* in $\mathbb{Z}^2$. We also sometimes refer to these vectors by their cardinal directions N, E, S, W, respectively. All *graphs* in this paper are undirected. A *grid graph* is a graph $G = (V, E)$ in which $V \subseteq \mathbb{Z}^2$ and every edge $\{\vec{a}, \vec{b}\} \in E$ has the property that $\vec{a} - \vec{b} \in U_2$.

Intuitively, a tile type t is a unit square that can be translated, but not rotated, having a well-defined "side $\vec{u}$" for each $\vec{u} \in U_2$. Each side $\vec{u}$ of t has a "glue" with "label" $\text{label}_t(\vec{u})$–a string over some fixed alphabet–and "strength" $\text{str}_t(\vec{u})$–a nonnegative integer–specified by its type t. Two tiles t and t' that are placed at the points $\vec{a}$ and $\vec{a} + \vec{u}$ respectively, *bind* with *strength* $\text{str}_t(\vec{u})$ if and only if $(\text{label}_t(\vec{u}), \text{str}_t(\vec{u})) = (\text{label}_{t'}(-\vec{u}), \text{str}_{t'}(-\vec{u}))$.

In the subsequent definitions, given two partial functions f, g, we write $f(x) = g(x)$ if f and g are both defined and equal on x, or if f and g are both undefined on x.

[2] Adleman et. al. [3] actually considered a slight variant of the Unique Assembly problem in which the input is a shape and the output is whether or not the input system uniquely assembles one supertile with that shape. Within the aTAM, the complexity of this variant problem is polynomially related to our problem. In contrast, this is not clearly the case in the 2HAM, making this variant problem a potentially interesting direction for future work. Further, [3] call their problem the Unique Shape problem, which is not the same as our version of the Unique Shape problem in that we do not require the input system be directed. Our version of the Unique Shape problem was first considered in [7].

Fix a finite set T of tile types. A T-*assembly*, sometimes denoted simply as an *assembly* when T is clear from the context, is a partial function $\alpha : \mathbb{Z}^2 \dashrightarrow T$ defined on at least one input, with points $\vec{x} \in \mathbb{Z}^2$ at which $\alpha(\vec{x})$ is undefined interpreted to be empty space, so that dom α is the set of points with tiles. We write $|\alpha|$ to denote $|\text{dom } \alpha|$, and we say α is *finite* if $|\alpha|$ is finite. For assemblies α and α', we say that α is a *subassembly* of α', and write $\alpha \sqsubseteq \alpha'$, if dom $\alpha \subseteq$ dom α' and $\alpha(\vec{x}) = \alpha'(\vec{x})$ for all $x \in$ dom α.

For $\tau \in \mathbb{N}$, an assembly is τ*-stable* if every cut of its binding graph has strength at least τ, where the weight of an edge is the strength of the glue it represents. That is, the supertile is stable if at least energy τ is required to separate the supertile into two parts.

2.1 Informal description of the abstract tile assembly model (aTAM)

In this section we give an informal description of the aTAM. The reader is encouraged to see [14, 15, 17] for a formal development of the model.

In the aTAM, self-assembly begins with a *seed assembly* σ (typically assumed to be finite and τ-stable) and proceeds asynchronously and nondeterministically, with tiles adsorbing one at a time to the existing assembly in any manner that preserves stability at all times.

An aTAM *tile assembly system* (*TAS*) is an ordered triple $\mathcal{T} = (T, \sigma, \tau)$, where T is a finite set of tile types, σ is a seed assembly with finite domain, and τ is the temperature. An *assembly sequence* in a TAS $\mathcal{T} = (T, \sigma, \tau)$ is a (possibly infinite) sequence $\vec{\alpha} = (\alpha_i \mid 0 \le i < k)$ of assemblies in which $\alpha_0 = \sigma$ and each α_{i+1} is obtained from α_i by the "τ-stable" addition of a single tile. The *result* of an assembly sequence $\vec{\alpha}$ is the unique assembly $\text{res}(\vec{\alpha})$ satisfying dom $\text{res}(\vec{\alpha}) = \bigcup_{0 \le i < k}$ dom α_i and, for each $0 \le i < k$, $\alpha_i \sqsubseteq \text{res}(\vec{\alpha})$.

We write $\mathcal{A}[\mathcal{T}]$ for the *set of all producible assemblies of* $\mathcal{T}$. An assembly α is *terminal*, and we write $\alpha \in \mathcal{A}_\Box[\mathcal{T}]$, if no tile can be stably added to it. We write $\mathcal{A}_\Box[\mathcal{T}]$ for the *set of all terminal assemblies of* $\mathcal{T}$. A TAS $\mathcal{T}$ is *directed*, or *produces a unique assembly*, if it has exactly one terminal assembly i.e., $|\mathcal{A}_\Box[\mathcal{T}]| = 1$. The reader is cautioned that the term "directed" has also been used for a different, more specialized notion in self-assembly [4]. We interpret "directed" to mean "deterministic", though there are multiple senses in which a TAS may be deterministic or nondeterministic.

Given a connected shape $X \subseteq \mathbb{Z}^2$, we say a TAS $\mathcal{T}$ *self-assembles* X if every producible, terminal assembly places tiles exactly on those positions in X. (Note that this notion is equivalent to *strict* self-assembly as defined in [14].) For an infinite shape $X \subseteq \mathbb{Z}^2$, we say that $\mathcal{T}$ *finitely self-assembles* X if every finite producible assembly of $\mathcal{T}$ has a possible way of growing into an assembly that places tiles exactly on those points in X. Note that if a shape X self-assembles in $\mathcal{T}$, then X finitely self-assembles in $\mathcal{T}$.

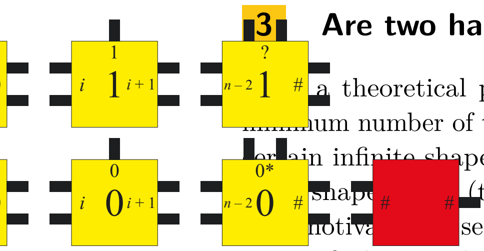
2.2 Informal description of two-handed tile assembly model (2HAM)

The 2HAM [1, 7, 9, 10, 12, 13] is a generalization of the aTAM in that it allows for two assemblies, both possibly consisting of more than one tile, to attach to each other. Since we must allow that the assemblies might require translation before they can bind, we define a *supertile* to be the set of all translations of a τ-stable assembly, and speak of the attachment of supertiles to each other, modeling that the assemblies attach, if possible, after appropriate translation. We now give a brief, informal, sketch of the 2HAM.

A *supertile* (a.k.a., *assembly*) is a positioning of tiles on the integer lattice $\mathbb{Z}^2$. Two adjacent tiles in a supertile *interact* if the glues on their abutting sides are equal and have positive strength. Each supertile induces a *binding graph*, a grid graph whose vertices are tiles, with an edge between two tiles if they interact. The supertile is τ*-stable* if it is τ-stable in the sense of aTAM. A 2HAM *tile assembly system* (TAS) is a pair $\mathcal{T} = (T, \tau)$, where T is a finite tile set and τ is the *temperature*, usually 1 or 2. Given a TAS $\mathcal{T} = (T, \tau)$, a supertile is *producible*, written as $\alpha \in \mathcal{A}[\mathcal{T}]$ if either it is a single tile from T, or it is the τ-stable result of translating two producible assemblies without overlap.[3] A supertile α is *terminal*, written as $\alpha \in \mathcal{A}_{\Box}[\mathcal{T}]$ if for every producible supertile β, α and β cannot be τ-stably attached. A TAS is *directed* if it has only one terminal, producible supertile.

Given a connected shape $X \subseteq \mathbb{Z}^2$, we say a TAS $\mathcal{T}$ *self-assembles* X if it self-assembles in the sense of aTAM (appropriately translated if necessary). For an infinite shape $X \subseteq \mathbb{Z}^2$, we say that $\mathcal{T}$ *finitely self-assembles* X if it finitely self-assembles in the sense of aTAM (appropriately translated if necessary).

3 Are two hands more (tile) efficient than one?



[illegible] a theoretical perspective, is the 2HAM "better" than the aTAM in terms of the minimum number of tiles required to uniquely produce a target shape? Is it possible to build certain infinite shapes in one model but not the other? Or perhaps is it possible to build [illegible] shape[illegible] (tile) efficiently in one model than the other? These are the questions [illegible] motiva[illegible] section.

We find, somewhat surprisingly, that it is possible for both models to "win", in the sense that there exist shapes that self-assemble more efficiently in the aTAM than the 2HAM, and vice versa, depending on both the choice of shape as well as temperature value. At temperature $\tau = 1$, we discover an $O(1)$ separation between the aTAM and 2HAM in favor of the aTAM winning. At temperature $\tau > 1$, we see a nearly exponential (and beyond) separation in favor of the 2HAM.

3.1 Finite Shapes: staircases

We first examine classes of finite shapes that "separate" the aTAM and the 2HAM with respect to the tile complexities of the systems that uniquely produce them.

Given a shape $X \subseteq \mathbb{Z}^2$, we say that $\mathcal{C}^{\tau}_{\text{aTAM}}(X)$ is the *tile complexity* of X in the aTAM at *temperature* $\tau \in \mathbb{N}$. In other words, $\mathcal{C}^{\tau}_{\text{aTAM}}(X) = \min\{|T| \mid \text{ for some } \sigma \text{ where } |\sigma| = 1$ and X self-assembles in $\mathcal{T} = (T, \sigma, \tau)\}$. Intuitively, $\mathcal{C}^{\tau}_{\text{aTAM}}(X)$ is the size of the smallest tile set that produces assemblies that place tiles on–and only on–the target shape X. Let $\mathcal{C}_{\text{aTAM}}(X) = \min\{\mathcal{C}^{\tau}_{\text{aTAM}}(X) | \tau \in \mathbb{N}\}$. The quantities $\mathcal{C}^{\tau}_{\text{2HAM}}(X)$ and $\mathcal{C}_{\text{2HAM}}(X)$ are defined similarly.

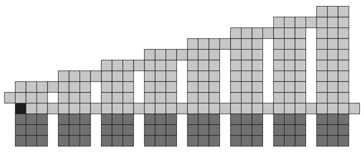


Figure 1 A staircase with 2^3 steps with each step of width 3. The black square represents the point $(0, 0)$.

▶ **Definition 3.1.** For each $i, k \in \mathbb{N}$, let $B_{i,k} = (\{0, \ldots, k-1\} \times \{-k, \ldots, 0, \ldots, i+2\}) \cup \{(-1, i+1), (k, 0)\}$ and define, $S_n = \bigcup_{i=0}^{2^n-1} (B_{i,n} + ((n+1)i, 0))$. Intuitively, the set S_n is a "staircase with 2^n steps with each step of width n." See Figure 1 for an example of S_3.

[3] The restriction on overlap is our formalization of the physical mechanism of steric protection.

We will use S_n to show a non-trivial (nearly) exponential separation between the aTAM and the 2HAM.

▶ **Theorem 3.2.** *For all $n \in \mathbb{N}$, $\mathcal{C}_{aTAM}(S_n) = \Omega(\frac{n}{\log n})$ and $\mathcal{C}^2_{2HAM}(S_n) = O\left(\frac{\log n}{\log\log n}\right)$.*

We use a counting argument to prove $\mathcal{C}_{aTAM}(S_n) = \Omega\left(\frac{n}{\log n}\right)$. It is interesting to note that, if one were to apply the standard, perhaps most obvious information-theoretic argument to prove the bound, one would only obtain a bound of $\Omega\left(\frac{\log n}{\log\log n}\right)$, which would not give more than a $O(1)$ separation between the aTAM and the 2HAM.

We get $\mathcal{C}^2_{\text{2HAM}}(S_n) = O\left(\frac{\log n}{\log\log n}\right)$ because, in 2HAM, we can enforce pairs of connector-column tiles to attach simultaneously, which is not possible in aTAM constructions. Intuitively, the construction works as follows. We begin by using a modified version of the optimal square construction [3] to form the lower $n \times n$ square portion of each stair step. We modify the optimal square construction to allow tiles to nondeterministically attach to the top row of the square to form a length n binary string. Then we use a binary counter [2, 7] to count from the nondeterministically chosen value, say x, up to $2^{n+1} - 1$. Finally, consecutive stair steps come together, in a purely two-handed fashion, via two strength-1 glues that are separated by a distance proportional to the height of the stair step on which they are present.

We can "iterate" the basic staircase construction using Turing machines to build each stair step. This gives an even greater separation.

▶ **Theorem 3.3** ("Busy Beaver" staircase)**.** *Let $M = (Q, \{0,1\}, 0, \{0,1\}, \delta, q_0, F)$ be a Turing machine and $x \in \{0,1\}^*$ such that M halts on x. Then $\mathcal{C}^2_{2HAM}\left(S_{2t(x)+|x|+2}\right) = O(|Q| + |x|)$, where $t(x)$ denotes the running time of M on input x.*

Theorem 3.3 says that, at temperature $\tau = 2$, the 2HAM can be used to build certain shapes much (much much...) more efficiently than in the aTAM, which requires some number of tile types nearly exponential in the number of time steps of a busy beaver Turing machine!

3.2 Infinite Shapes

In this subsection, we examine a class of infinite (staircase-like) shapes that finitely self-assemble in 2HAM but do not self-assemble in aTAM.

We first note that it is easy to exhibit a class of infinite shapes that self-assemble in the aTAM but do not self-assemble in the 2HAM. Simply take any finite shape $X \subset \mathbb{Z}^2$ and union it with a one-way infinite line to get a kind of "blob with an infinite tail" (See Figure 2 for an example of such a shape). Such shapes do not self-assemble in the 2HAM via a straightforward pumping lemma argument on the infinite tail portion of the shape. However, we note that it is easy to take any such blob+tail shape and exhibit an aTAM TAS

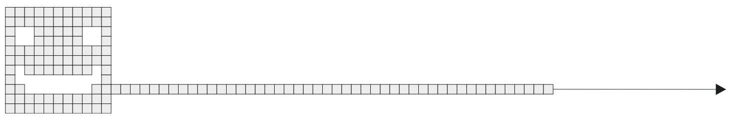


Figure 2 A blob with an infinite tail.

in which that shape self-assembles. To see this, simply create hard-coded tile types for the finite blob portion (with the seed tile placed at some location in the blob) and then have a single tile type that repeats infinitely in one direction for the tail portion. This construction also testifies to the finite self-assembly of a blob+tail shape in the 2HAM.

▶ **Definition 3.4.** For each $i \in \mathbb{N}$, let $B_i = (\{0,\ldots,i+2\} \times \{0,\ldots,i+2\}) \cup \{(-1,i+1),(i,0)\}$ and $S_\infty = \bigcup_{i=0}^{\infty} \left(B_i + \left(\left(\frac{i(i+7)}{2}\right),0\right)\right)$. Intuitively, the set S_∞ is essentially a succession of larger and larger squares that are connected by pairs of tiles positioned at the top right and bottom right of each square. See Figure 3 for an example.

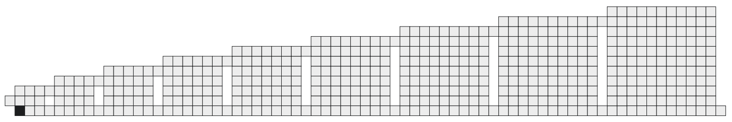


Figure 3 A finite portion of the infinite staircase, denoted as S_∞. The black square represents the origin.

▶ **Theorem 3.5.** *The infinite staircase S_∞ finitely self-assembles in the 2HAM.*

Intuitively, our construction for Theorem 3.5 proceeds as follows. We first assemble horizontal lines using three tile types: one to start the line, one to keep it going and one to stop the line. The tile that stops the line may attach non-deterministically at any step, whence lines of every length are able to form. Each line of length k ultimately grows into a $k \times k$ square. Connector-tiles that attach to the left and right of each square ensure that only a $(k-1) \times (k-1)$ square may attach to the left of a $k \times k$ square.

▶ **Theorem 3.6.** *The infinite staircase S_∞ does not finitely self-assemble in the aTAM.*

Intuitively, the proof for Theorem 3.6 is the "infinite" version of Theorem 3.2 in which there are infinitely many identical and cooperating pairs of connector tiles, and we can use really "tall" cooperating connector-columns to force "shorter" versions of identical connector-columns to grow outside of S_∞.

▶ **Corollary 3.7.** *The infinite staircase S_∞ does not self-assemble in the aTAM.*

4 Simulating aTAM with 2HAM

This section describes how to simulate an aTAM system by a 2HAM system, which suggests that anything the aTAM can do, the 2HAM can do (at least as good as, if not) better. A key property of our constructions is that they not only simulate the produced shapes assembled by the aTAM system, but also simulate the incremental assembly process, where single tiles aggregate on a larger seed assembly.

4.1 Simulation definition: simulate an aTAM (or 2HAM) system with another 2HAM (or aTAM) system

In this subsection, we formally define what it means for one 2HAM TAS to "simulate" another 2HAM (or aTAM) TAS. For a tileset T, let $\mathcal{A}^T$ and $\tilde{\mathcal{A}}^T$ denote the set of all assemblies over T and all supertiles over T respectively.

An *m-block assembly* over tile set S is a partial function $\gamma : \mathbb{Z}_m^2 \dashrightarrow S$. Let B_m^S be the set of all m-block assemblies over S. The m-block with no domain is said to be *empty*. For a general assembly $\alpha \in \mathcal{A}^S$ define $\alpha_{x,y}^m$ to be the m-block defined by $\alpha_{x,y}^m(i,j) = \alpha(mx+i, my+j)$ for $0 \leq i,j < m$. For a partial function $R : B_m^S \dashrightarrow T$, define the *assembly replacement function* $R^* : \mathcal{A}^S \rightarrow \mathcal{A}^T$ such that $R^*(\alpha) = \beta$ if and only if $\beta(x,y) = R(\alpha_{x,y}^m)$ for all $x,y \in \mathbb{Z}^2$. Further, α is said to map *cleanly* to β under R^* if for all non empty blocks $\alpha_{x,y}^m$, either 1)

$(x+u, y+v) \in \text{dom}\,\beta$ for some $u, v \in \{-1, 0, 1\}$, or 2) α has at most one non-empty m-block $\alpha^m_{x,y}$. For a given *assembly replacement function* R^*, define the *supertile replacement function* $\tilde{R} : \tilde{A}^S \to \mathcal{P}(A^T)$ such that $\tilde{R}(\tilde{\alpha}) = \{R^*(\alpha) | \alpha \in \tilde{\alpha}\}$. $\tilde{\alpha}$ is said to *map cleanly* to $\tilde{R}(\tilde{\alpha})$ if $\tilde{R}(\tilde{\alpha}) \in \tilde{A}^T$ and α maps cleanly to $R^*(\alpha)$ for all $\alpha \in \tilde{\alpha}$.

Consider an aTAM or 2HAM system $\mathcal{S}$ with tileset S, and an aTAM or 2HAM system $\mathcal{T}$ with tile set T. $\mathcal{S}$ *simulates* $\mathcal{T}$ *at scale factor* m if there exists an m-block replacement $R : B^S_m \to T$ satisfying the following conditions.

1. Equivalent Production: (1) $\left\{\tilde{R}(\alpha) | \alpha \in \mathcal{A}[\mathcal{S}]\right\} = \mathcal{A}[\mathcal{T}]$ and (2) for all $\alpha \in \mathcal{A}[\mathcal{S}]$, α maps cleanly to $\tilde{R}(\alpha)$
2. Equivalent Dynamics: (1) For any $\alpha, \alpha' \in \mathcal{A}[\mathcal{S}]$ such that $\alpha \to^1_{\mathcal{S}'} \alpha'$, then $\tilde{R}(\alpha) \to^{\leq 1}_{\mathcal{T}} \tilde{R}(\alpha')$, and (2) for any $\beta, \beta' \in \mathcal{A}[\mathcal{T}]$ such that $\beta \to^1_{\mathcal{T}} \beta'$, then for all α such that $\tilde{R}(\alpha) = \beta$, there exists an α'' such that $\tilde{R}(\alpha'') = \beta$, $\alpha \to_{\mathcal{S}} \alpha''$, and $\alpha'' \to^1_{\mathcal{S}} \alpha'$ for some α' with $\tilde{R}(\alpha') = \beta'$.

4.2 Simulating aTAM at $\tau \geq 4$ with 2HAM $\tau = 4$

It is possible to simulate the aTAM at temperature $\tau \geq 4$ using the 2HAM at temperature 4 with a constant scale factor of 5. Given any aTAM system, each tile t in the aTAM system is represented by 25 tiles forming a 5×5 macrotile assembly in the 2HAM system. The macrotile in the 2HAM system consists of a 3×3 center *brick* assembly, surrounded on all sides by a *mortar* one tile thick. These tiles are designed such that bricks and certain mortar pieces can assemble independently, but bricks cannot attach to mortar pieces or other bricks unless additional tiles are present.

We mimic the seeded nature of aTAM systems by allowing the mortar to assemble around a seed brick corresponding to the seed tile in the aTAM system by strengthening the glues at this seed macrotile. Once any brick has its complete set of mortar pieces attached to it, mortar pieces for adjacent tiles can attach to the assembly; new bricks can then attach to this partially built assembly only once their mortar is partially constructed. In this way, we ensure that bricks can only attach to partially built assemblies containing a seed brick, mimicking the seeded nature of an aTAM system. Additionally, we divide instances of glues into *inward* and *outward* glue sets, such that an outward glue g can only attach to an inward glue of the same type. Throughout the assembly process, the invariant that all exposed glues in any assembly containing a seed brick are outward glues is maintained; this prevents partially built seeded assemblies from attaching to each other. An example of the construction in which 3×3 bricks, 3×1 mortar rectangles, and individual mortar tiles attach to form 5×5 supertiles can be seen in Figure 4.

▶ **Theorem 4.1.** *Any aTAM system at $\tau \geq 4$ can be simulated by a 2HAM system at $\tau = 4$.*

4.3 Simulating aTAM at $\tau \in \{1, 2\}$ with 2HAM $\tau = 2$

The construction described in the previous section can be modified to also enable simulating aTAM systems at $\tau = \{1, 2\}$ with the 2HAM at $\tau = 2$ with scale factor 5.

▶ **Theorem 4.2.** *Any aTAM system at $\tau \in \{1, 2\}$ can be simulated by a 2HAM system at $\tau = 2$.*

4.4 Simulating aTAM at $\tau = 3$ with 2HAM $\tau = 3$

The construction used to simulate the $\tau \geq 4$ aTAM model with the $\tau = 4$ 2HAM model can also be modified to simulate the $\tau = 3$ aTAM model with the $\tau = 3$ 2HAM model. The

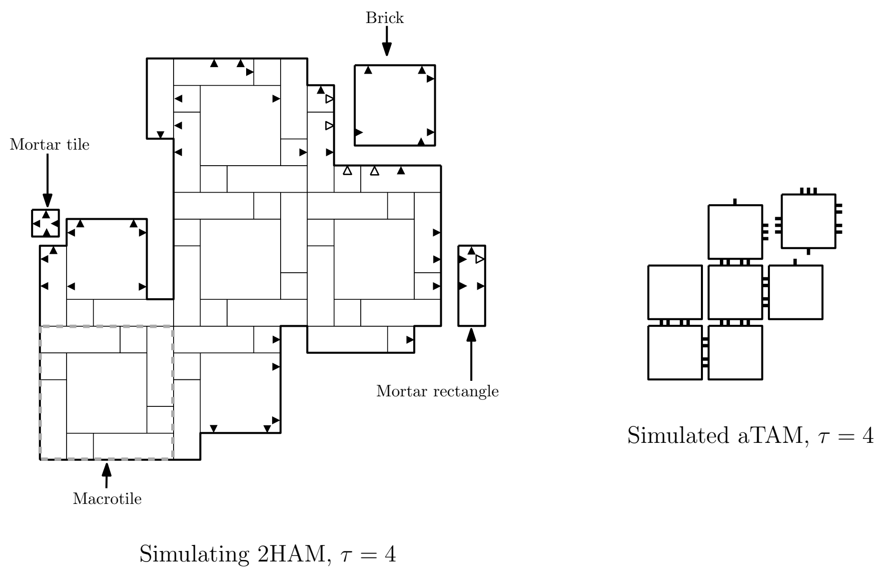


Figure 4 The simulation of an assembly in an aTAM system simulated using a 2HAM system. The filled and unfilled arrows represent glues of strength 2 and 1 respectively in the 2HAM system, while the dashes each represent a bond of strength 1 in the aTAM system (i.e. 4 dashes on the North side of a tile is a glue of strength 4).

construction given also simulates the aTAM model under the restriction of planarity (tiles can only attach at locations on the exterior of the assembly).

▶ **Theorem 4.3.** *Any aTAM system at $\tau = 3$ can be simulated by a 2HAM system at $\tau = 3$.*

5 Verification algorithms for aTAM and 2HAM

In this section, we explore the algorithmic complexities of verifying certain properties of a given (2HAM or aTAM) tile assembly system. Sections 3 and 4 suggest that the 2HAM is at least as (if not perhaps strictly more) powerful than the aTAM. In this section, we show that verifying properties of self-assembly systems in the 2HAM is at least as (if not perhaps strictly more) difficult than verifying properties of aTAM systems.

5.1 Unique assembly verification

A fundamental computational problem in self-assembly is that of deciding whether a given self-assembly system uniquely produces a given assembly. We refer to this problem as the Unique Assembly Verification problem (UAV). The aTAM has enjoyed a polynomial time solution [3] to this problem reaching back to 2002. Fast verification within the aTAM has been of tremendous assistance for self-assembly system designers by allowing for simulators that can quickly spot bugs in tile systems. In contrast, the complexity of UAV for 2HAM systems has been a core open problem since the Palaeolithic era. The results of this paper, thus far, seem to suggest "aTAM = O(2HAM)", i.e., the 2HAM is, in general, at least as powerful as the aTAM. Thus, it should not be difficult for one to believe that, in general, verifying 2HAM systems should be at least as difficult as verifying aTAM systems. In this section, we show that a general fast verification algorithm is unlikely to exist by showing that the UAV is co-NP-complete.

Our UAV co-NP-complete result applies to temperature $\tau = 2$ systems that utilize at most one step into the third dimension[4]. This result resolves the general question of whether efficient unique assembly verification algorithms exist, but leaves open the possibility of a fast algorithm for the important class of 2D 2HAM self-assembly systems. Further, this result is potentially useful for optimistic algorithm designers in search of such 2D efficient systems in that it points out that any such solution will need to make fundamental use of the planarity of self-assembly to have a chance at working. Formally, the UAV problem is stated as follows:

Input:	An aTAM system $\mathcal{T} = (T, \sigma, \tau)$, or a 2HAM system $\mathcal{T} = (T, \tau)$, and a T-assembly α
Output:	Does $\mathcal{T}$ uniquely produce α, i.e., is α such that $\mathcal{A}_{\Box}[\mathcal{T}] = \{\alpha\}$?

▶ **Theorem 5.1.** *The UAV problem is co-NP-complete for 3D, temperature $\tau = 2$ 2HAM systems that use only 2 separate planes of the third dimension.*

Proof sketch. Proving membership in co-NP involves observing that a non-unique producible assembly implies the existence of a small, producible witness to non-uniqueness that is inconsistent with the input assembly. NP-hardness is shown by reducing from 3-SAT (see Figure 5. The assembly input tile system places clause blocks, row by row, from bottom to top, with the completion of a given row verifying that a given clause is satisfied by the variable assignment represented by the attachment of a sequence of variable loops. The assembly has the property that upon completion of all clause rows, 2 glues are exposed that may permit a final attachment that is inconsistent with the input assembly. Such a completion is impossible for non-satisfiable formulas without the use of *cheating* in which some variable is assigned both true and false values. If cheating occurs, the true and false variable loops that *cheated* will restrict the final attachment from growing further, yielding that the target assembly is uniquely produced if and only if the 3-SAT formula has no satisfying assignment. ◀

Acknowledgments

The authors would like to thank Zachary Abel, David Doty and Damien Woods for helpful discussions.

References

1 Zachary Abel, Nadia Benbernou, Mirela Damian, Erik D. Demaine, Martin L. Demaine, Robin Y. Flatland, Scott D. Kominers, and Robert T. Schweller. Shape replication through self-assembly and rnase enzymes. In *SODA*, pages 1045–1064, 2010.

2 Leonard Adleman, Qi Cheng, Ashish Goel, and Ming-Deh Huang. Running time and program size for self-assembled squares. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 740–748, Hersonissos, Greece, 2001.

3 Leonard M. Adleman, Qi Cheng, Ashish Goel, Ming-Deh A. Huang, David Kempe, Pablo Moisset de Espanés, and Paul W. K. Rothemund. Combinatorial optimization problems in self-assembly. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing*, pages 23–32, 2002.

4 Leonard M. Adleman, Jarkko Kari, Lila Kari, Dustin Reishus, and Petr Sosík. The undecidability of the infinite ribbon problem: Implications for computing by self-assembly. *SIAM Journal on Computing*, 38(6):2356–2381, 2009.

[4] We do not formally define the 3D 2HAM because the generalization from the 2D 2HAM is straightforward. Details of the 3D 2HAM that we use can be found in [8]

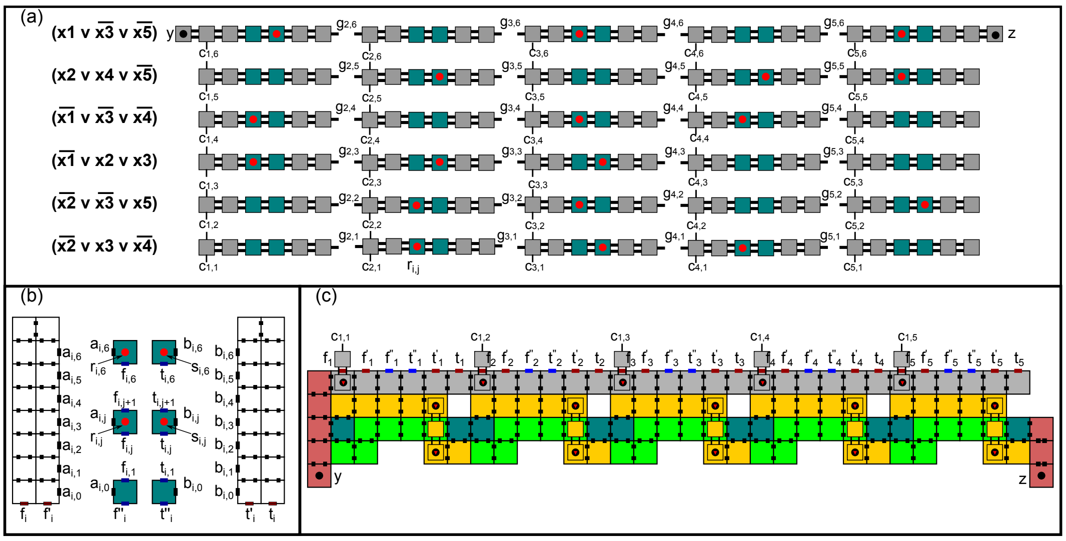


Figure 5 This figure details the tile set for the temperature $\tau = 2$ system used in the polynomial time reduction of the 3-SAT problem to the Unique Assembly problem. The tiles in this figure are those derived for the example 3-SAT instance shown in (a). Tiles that are placed within the $z = 1$ plane appear smaller than those that occur in the $z = 0$ plane. Strength-1 glues are denoted by single dashes for north,south,east and west glues, and solid circles for top and bottom glues. Strength-2 glues are denoted by double dashes and triangle inscribed circles for top/bottom glues. Each glue within this system occurs on exactly two tile faces of opposite orientation. Some tiles are shown as already bound together for the purpose of implicitly specifying which edges share strength-2 glues.

5 Robert D. Barish, Rebecca Schulman, Paul W. Rothemund, and Erik Winfree. An information-bearing seed for nucleating algorithmic self-assembly. *Proceedings of the National Academy of Sciences*, 106(15):6054–6059, March 2009.

6 Ho-Lin Chen and David Doty. Parallelism and time in hierarchical self-assembly. In *SODA 2012: Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms.*

7 Qi Cheng, Gagan Aggarwal, Michael H. Goldwasser, Ming-Yang Kao, Robert T. Schweller, and Pablo Moisset de Espanés. Complexities for generalized models of self-assembly. *SIAM Journal on Computing*, 34:1493–1515, 2005.

8 Matthew Cook, Yunhui Fu, and Robert Schweller. Temperature 1 self-assembly: Deterministic assembly in 3d and probabilistic assembly in 2d. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, 2011.

9 Erik D. Demaine, Martin L. Demaine, Sándor P. Fekete, Mashhood Ishaque, Eynat Rafalin, Robert T. Schweller, and Diane L. Souvaine. Staged self-assembly: nanomanufacture of arbitrary shapes with $O(1)$ glues. *Natural Computing*, 7(3):347–370, 2008.

10 Erik D. Demaine, Matthew J. Patitz, Robert T. Schweller, and Scott M. Summers. Self-assembly of arbitrary shapes using rnase enzymes: Meeting the kolmogorov bound with small scale factor (extended abstract). In *STACS 2011: Proceedings of the Twenty Eighth International Symposium on Theoretical Aspects of Computer Science*, pages 201–212, 2011.

11 David Doty. personal communication, 2011.

12 David Doty, Matthew J. Patitz, Dustin Reishus, Robert T. Schweller, and Scott M. Summers. Strong fault-tolerance for self-assembly with fuzzy temperature. In *FOCS 2010: Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science*, pages 417–426. IEEE, 2010.

13 Bin Fu, Matthew J. Patitz, Robert T. Schweller, and Robert Sheline. Self-assembly with geometric tiles. In *ICALP (1)*, pages 714–725, 2012.

14 James I. Lathrop, Jack H. Lutz, and Scott M. Summers. Strict self-assembly of discrete Sierpinski triangles. *Theoretical Computer Science*, 410:384–405, 2009.

15 Paul W. K. Rothemund and Erik Winfree. The program-size complexity of self-assembled squares (extended abstract). In *STOC '00: Proceedings of the thirty-second annual ACM Symposium on Theory of Computing*, pages 459–468, 2000.

16 Paul W.K. Rothemund, Nick Papadakis, and Erik Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biology*, 2(12):2041–2053, 2004.

17 Erik Winfree. *Algorithmic Self-Assembly of DNA*. PhD thesis, California Institute of Technology, June 1998.

Unlabeled Data Does Provably Help*

Malte Darnstädt[1], Hans Ulrich Simon[1], and Balázs Szörényi[2,3]

1 **Department of Mathematics, Ruhr-University Bochum, Germany**
`{malte.darnstaedt,hans.simon}@rub.de`
2 **INRIA Lille, SequeL project, France**
3 **MTA-SZTE Research Group on Artificial Intelligence, Szeged, Hungary**
`szorenyi@inf.u-szeged.hu`

Abstract

A fully supervised learner needs access to correctly labeled examples whereas a semi-supervised learner has access to examples part of which are labeled and part of which are not. The hope is that a large collection of unlabeled examples significantly reduces the need for labeled-ones. It is widely believed that this reduction of "label complexity" is marginal unless the hidden target concept and the domain distribution satisfy some "compatibility assumptions". There are some recent papers in support of this belief. In this paper, we revitalize the discussion by presenting a result that goes in the other direction. To this end, we consider the PAC-learning model in two settings: the (classical) fully supervised setting and the semi-supervised setting. We show that the "label-complexity gap" between the semi-supervised and the fully supervised setting can become arbitrarily large for concept classes of infinite VC-dimension (or sequences of classes whose VC-dimensions are finite but become arbitrarily large). On the other hand, this gap is bounded by $O(\ln|C|)$ for each finite concept class C that contains the constant zero- and the constant one-function. A similar statement holds for all classes C of finite VC-dimension.

1998 ACM Subject Classification I.2.6 Concept Learning

Keywords and phrases algorithmic learning, sample complexity, semi-supervised learning

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.185

1 Introduction

In the PAC[1]-learning model [11], a learner's input are samples, labeled correctly according to an unknown target concept, and two parameters $\varepsilon, \delta > 0$. He has to infer, with high probability of success, an approximately correct binary classification rule, which is called "hypothesis" in this context. In the non-agnostic setting (that we focus on in this paper), the following assumptions are made:

- There is a concept class C (known to the learner) so that the "correct" labels are assigned to the instances x from the underlying domain X by a function $c : X \rightarrow \{0, 1\}$ from C (the unknown target function).
- There is a probability distribution P on X (unknown to the learner) so that the samples (labeled according to c) are independently chosen at random according to P.

The learner is considered successful if his hypothesis h satisfies $P[h(x) \neq c(x)] < \varepsilon$ (approximate correctness).[2] The probability for success should be larger than $1 - \delta$ (so the learner's

* This work was supported by the bilateral Research Support Programme between Germany (DAAD 50751924) and Hungary (MÖB 14440).
[1] PAC = Probably Approximately Correct
[2] Note, that we don't require the learner to observe that his hypothesis is accurate to be successful.

hypothesis is probably approximately correct). The learner is called *proper* if he commits himself to picking his hypothesis from C. We refer to ε as the *accuracy parameter*, or simply as the *accuracy*, and we refer to δ as the *confidence parameter*, or simply as the *confidence*.

Providing a learner with a large collection of labeled samples is expensive because reliable classification labels are typically generated by a human expert. On the other hand, unlabeled samples are easy to get (e.g., can be collected automatically from the web). This raised the question whether the "label complexity" of a learning problem can be significantly reduced when learning is "semi-supervised", i.e., if the learner is not only provided with labeled samples but also with unlabeled-ones.[3] The existing analysis of the semi-supervised setting can be summarized roughly as follows:

- The benefit of unlabeled samples can be enormous if the target concept and the domain distribution satisfy some suitable "compatibility assumptions" (see [1]).
- On the other hand, the benefit seems to be marginal if we do not impose any extra-assumptions (see [2, 7]).

These findings perfectly match with the common belief that some kind of compatibility between the target concept and the domain distribution is needed for adding horsepower to semi-supervised algorithms. However, the results of the second type are not yet fully convincing:

- The paper [2] provides some upper bounds on the label complexity in the fully supervised setting and some lower bounds, that match up to a small constant factor, in the semi-supervised setting (or even in the setting with a distribution P that is known to the learner). These bounds however are established only for some special concept classes over the real line. It is unclear whether they generalize to a broader variety of concept classes.
- The paper [7] analyzes arbitrary finite concept classes and shows the existence of a purely supervised "smart" PAC-learning algorithm whose label consumption exceeds the label consumption of the best learner with full prior knowledge of the domain distribution at most by a constant factor for the "vast majority" of pairs (c, P). This however does not exclude the possibility that there still exist "bad pairs" (c, P) leading to a poor performance of the smart learner.

In this paper, we reconsider the question whether unlabeled samples can be of significant help to a learner even when we do not impose any extra-assumptions on the PAC-learning model. A comparably old paper, [8], indicates that an affirmative answer to this question is thinkable (despite of the fact that it was written a long time before semi-supervised learning became an issue). In [8] it is shown that there exists a concept class C_∞ and a family $\mathcal{P}_\infty$ of domain distributions such that the following holds:

1. For each $P \in \mathcal{P}_\infty$, C_∞ is properly PAC-learnable under the fixed distribution P (where "fixed" means that the learner has full prior knowledge of P).
2. C_∞ is *not properly* PAC-learnable under unknown distributions taken from $\mathcal{P}_\infty$.

These results point into the right direction for our purpose, but they are not precisely what we want:

- Although "getting a large unlabeled sample" comes close to "knowing the domain distribution", it is not quite the same. (In fact, one can show that C_∞, with domain distributions taken from $\mathcal{P}_\infty$, is *not* PAC-learnable in the semi-supervised setting.)

[3] In contrast to the setting of "active learning", we do however not assume that the learner can actively decide for which samples the labels are uncovered.

- The authors of [8] do *not* show that C_∞ is *not PAC-learnable* under unknown distributions P taken from $\mathcal{P}_\infty$. In fact, their proof uses a target concept that almost surely (w.r.t. P) assigns 1 to every instance in the domain. But the (proper!) learner must not return the constant 1-function of error 0 because of his commitment to hypotheses from C_∞.

Main Results:

The precise statement of our main results requires some more notation. For any concept class C over domain X and any domain distribution P, let $m_{C,P}(\varepsilon,\delta)$ denote the smallest number of labeled samples (in dependence of the accuracy ε and the confidence δ) needed to PAC-learn C under fixed distribution P. For any concept class C and any (semi-supervised or fully supervised) PAC-learning algorithm A, let $m^A_{C,P}(\varepsilon,\delta)$ denote the smallest number of labeled samples such that the resulting hypothesis of A is ε-accurate with confidence δ provided that P, unknown to A, is the underlying domain distribution. We first investigate the conjecture (up to minor differences identical to Conjecture 4 in [2])[4] that there is a purely supervised learner whose label consumption exceeds the label consumption of the best learner with full prior knowledge of the domain distribution at most by a factor $k(C)$ that depends on C only, as opposed to a dependence on ε or δ. The following result, whose proof is found in Section 3.1, confirms this conjecture to a large extent for finite classes, and to a somewhat smaller extent for classes of finite VC-dimension:

▶ **Theorem 1.** *Let C be a concept class over domain X that contains the constant zero- and the constant one-function. Then:*

1. *If C is finite, there exists a fully supervised PAC-learning algorithm A such that, for every domain distribution P, $m^A_{C,P}(2\varepsilon,\delta) = O(\ln|C|)\cdot m_{C,P}(\varepsilon,\delta)$.*
2. *If the VC-dimension of C is finite, there exists a fully supervised PAC-learning algorithm A such that, for every domain distribution P, $m^A_{C,P}(2\varepsilon,\delta) = O(\mathrm{VCdim}(C)\cdot\log(1/\varepsilon))\cdot m_{C,P}(\varepsilon,\delta) = \tilde{O}(\mathrm{VCdim}(C))\cdot m_{C,P}(\varepsilon,\delta)$.*

Can we generalize Theorem 1 to concept classes C of infinite VC-dimension provided that the domain distribution is taken from a family $\mathcal{P}$ such that $m_{C,P}(\varepsilon,\delta) < \infty$ for all $P \in \mathcal{P}$? This question will be answered to the negative by the following result (proved in Section 3.2):

▶ **Theorem 2.** *There exists a concept class C_* over domain $\{0,1\}^*$ and a family $\mathcal{P}_*$ of domain distributions such that the following holds:*

1. *There exists a semi-supervised algorithm A such that, for all $P \in \mathcal{P}_*$, $m^A_{C_*,P} = O(1/\varepsilon^2 + \log(1/\delta)/\varepsilon)$. (This implies the same upper bound on $m_{C_*,P}$ for all $P \in \mathcal{P}_*$.)*
2. *For every fully supervised algorithm A and for all $\varepsilon < 1/2, \delta < 1$:* $\sup_{P\in\mathcal{P}_*} m^A_{C_*,P}(\varepsilon,\delta) = \infty$.

Does there exist a universal constant k (not depending on C) such that we get a result similar to Theorem 1 but with $k(C)$ replaced by k? The following result (proved in Section 3.2) shows that, even for classes of finite VC-dimension, such a universal constant does not exist.

▶ **Theorem 3.** *There exists a sequence $(C_n)_{n\geq1}$ of concept classes over domains $(\{0,1\}^n)_{n\geq1}$ such that $\lim_{n\to\infty}\mathrm{VCdim}(C_n) = \infty$ and a sequence $(\mathcal{P}_n)_{n\geq1}$ of domain distribution families such that the following holds:*

1. *There exists a semi-supervised algorithm A that PAC-learns $(C_n)_{n\geq1}$ under any unknown distribution and, for all $P \in \mathcal{P}_n$, $m^A_{C_n,P}(\varepsilon,\delta) = O(1/\varepsilon^2 + \log(1/\delta)/\varepsilon)$. (This implies the same upper bound on $m_{C_n,P}$ for all $P \in \mathcal{P}_n$.)*

[4] In contrast to [2], we allow the supervised learner to be twice as inaccurate as the semi-supervised learner because, otherwise, it can be shown that results in the manner of Theorem 1 are impossible even for simple classes.

2. *For every fully supervised algorithm A and all $\varepsilon < 1/2, \delta < 1$:*
   $\sup_{n\geq 1, P\in\mathcal{P}_n} m^A_{C_n,P}(\varepsilon,\delta) = \infty$.

Some comments are in place here:

- Since the class C_* from Theorem 2 has a countable domain, namely $\{0,1\}^*$, C_* occurs (via projection) as a subclass in every concept class that shatters a set of infinite cardinality. A similar remark applies to the sequence $(C_n)_{n\geq 1}$ and concept classes that shatter finite sets of arbitrary size. Thus every concept class of infinite VC-dimension contains subclasses that are significantly easier to learn in the semi-supervised setting of the PAC-model (in comparison to the full supervised setting).
- An error bound $\varepsilon = 1/2$ is trivially achieved by random guesses for the unknown label. Let α and β be two arbitrary small, but strictly positive, constants. Theorems 2 and 3 imply that even the modest task of returning, with a success probability of at least α, a hypothesis of error at most $1/2 - \beta$ cannot be achieved in the fully supervised setting unless the number of labeled examples becomes arbitrarily large.
- Theorem 3 implies that the results from [2] do *not* generalize (from the simple classes discussed there) to arbitrary finite classes. It implies furthermore that the "bad pairs" (c, P) occurring in the main result from [7] are unavoidable and not an artifact of the analysis in that paper.
- C_n is not an artificially constructed or exotic class: it is in fact the class of non-negated literals over n boolean variables, which occurs as a subset of many popular concept classes (e.g. monomials, decision lists, half spaces). The class C_* is a natural generalization of C_n to the set of boolean strings of arbitrary length.
- The classes $C_*, \mathcal{P}_*$ from Theorem 2, defined in Section 3.2, are close relatives of the classes $C_\infty, \mathcal{P}_\infty$ from [8], but the adversary argument that we have to employ is much more involved than the corresponding argument in [8] (where the learner was assumed to be proper and had been fooled mainly because of his commitment to hypotheses from C_∞).

2 Definitions, Notations and Facts

For any $n \in \mathbb{N}$, we define $[n] = \{1, \ldots, n\}$. The symmetric difference between two sets A and B is denoted $A \oplus B$, i.e., $A \oplus B = (A \setminus B) \cup (B \setminus A)$. The indicator function $\mathbb{I}(\text{cond})$ yields 1 if "cond" is a true condition, and 0 otherwise.

2.1 Prerequisites from Probability Theory

Let X be an integer-valued random variable. As usual, a most likely value a for X is called a *mode* of X. In this paper, the largest integer that is a mode of X is denoted $\mathrm{mode}(X)$. As usual, X is said to be *unimodal* if $\Pr[X = x]$ is increasing with x for all $x \leq \mathrm{mode}(X)$, and decreasing with x for all $x \geq \mathrm{mode}(X)$.

Let Ω be a space equipped with a σ-algebra of events and with a probability measure P. For any sequence $(A_n)_{n\geq 1}$ of events, $\limsup_{n\to\infty} A_n$ is defined as the set of all $\omega \in \Omega$ that occur in infinitely many of the sets A_n, i.e., $\limsup_{n\to\infty} A_n = \cap_{n=1}^\infty \cup_{m=n}^\infty A_m$. We briefly remind the reader of the Borel-Cantelli Lemma:

▶ **Lemma 4** ([9]). *Let $(A_n)_{n\geq 1}$ be a sequence of independent events, and let $A = \limsup_{n\to\infty} A_n$. Then $P(A) = 1$ if $\sum_{n=1}^\infty P(A_n) = \infty$, and $P(A) = 0$ otherwise.*

▶ **Corollary 5.** *Let $(A_n)_{n\geq 1}$ be a sequence of independent events such that $\sum_{n=1}^{\infty} P(A_n) = \infty$. Let $B_{k,n}$ be the set of all $\omega \in \Omega$ that occur in at least k of the events $A_1, \ldots, A_n$. Then, for any $k \in \mathbb{N}$, $\lim_{n\to\infty} P(B_{k,n}) = 1$.*

Proof. Note that $B_{k,n} \subseteq B_{k,n+1}$ for every n. Since probability measures are continuous from below, it follows that $\lim_{n\to\infty} P(B_{k,n}) = P(\cup_{n=1}^{\infty} B_{k,n})$. Since, obviously, $\limsup_{n\to\infty} A_n \subseteq \cup_{n=1}^{\infty} B_{k,n}$, an application of the Borel-Cantelli Lemma yields the result. ◀

The following result, which is a variant of the Central Limit Theorem for triangular arrays, is known in the literature as the Lindeberg-Feller Theorem:

▶ **Theorem 6** ([6]). *Let $(X_{n,i})_{n\in\mathbb{N},i\in[n]}$ be a (triangular) array of random variables such that*
1. *$\mathbb{E}[X_{n,i}] = 0$ for all $n \in \mathbb{N}$, $i = 1, \ldots, n$.*
2. *$X_{n,1}, \ldots, X_{n,n}$ are independent for every $n \in \mathbb{N}$.*
3. *$\lim_{n\to\infty} \sum_{i=1}^{n} \mathbb{E}[X_{n,i}^2] = \sigma^2 > 0$.*
4. *For each $\varepsilon > 0$, $\lim_{n\to\infty} s_n(\varepsilon) = 0$ where $s_n(\varepsilon) = \sum_{i=1}^{n} \mathbb{E}[X_{n,i}^2 \mathbb{I}(|X_{n,i}| \geq \varepsilon)]$.*

Then $\lim_{n\to\infty} P\left[a < \frac{1}{\sigma} \cdot \sum_{i=1}^{n} X_{n,i} < b\right] = \varphi(b) - \varphi(a)$ where φ denotes the density function of the standard normal distribution.

An easy padding argument shows that this theorem holds “mutatis mutandis” for triangular arrays of the form $(X_{n_k,i})$ where $i = 1, \ldots, n_k$ and $(n_k)_{k\geq 1}$ is an increasing and unbounded sequence of positive integers. (The limes is then taken for $k \to \infty$.) We furthermore note that, for the special case of independent Bernoulli variables $X_{n,i}$ with probability p_i of success, Theorem 6 applies to the triangular array $(X_{n,i} - p_i)/\sigma_n$ where $\sigma_n^2 = \sum_{i=1}^{n} p_i(1 - p_i)$. (A similar remark applies to the more general case of bounded random variables.)

The following result is an immediate consequence of Theorem 6 (plus the remarks thereafter):

▶ **Lemma 7.** *Let $l(k) = o(\sqrt{k})$. Let $(n_k)_{k\geq 1}$ be an increasing and unbounded sequence of positive integers. Let $(p_{k,i})_{k\in\mathbb{N},i\in[n_k]}$ range over all triangular arrays of parameters in $[0, 1]$ such that*

$$\forall k \in \mathbb{N} : \sum_{i=1}^{n_k} p_{k,i}(1 - p_{k,i}) \geq k \ . \tag{1}$$

Let $(X_{k,i})_{k\in\mathbb{N},i\in[n_k]}$ be the corresponding triangular array of row-wise independent Bernoulli variables. Then the function h given by

$$h(k) = \sup_{(p_{k,i})} \sup_{s\in\{0,\ldots,n_k\}} P\left[\left|\sum_{i=1}^{n_k} X_{k,i} - s\right| < l(k)\right]$$

approaches 0 as k approaches infinity.

Proof. Assume for sake of contradiction that $\limsup_{k\to\infty} h(k) > 0$. Then there exist $(p_{k,i})$ satisfying (1) and $s_k \in \{0, \ldots, n_k\}$ such that

$$\limsup_{k\to\infty} P\left[\left|\sum_{i=1}^{n_k} X_{k,i} - s_k\right| < l(k)\right] > 0 \ . \tag{2}$$

The random variable $S_k = \sum_{i=1}^{n_k} X_{k,i}$ has mean $\mu_k = \sum_{i=1}^{n_k} p_{k,i}$ and variance $\sigma_k^2 = \sum_{i=1}^{n_k} p_{k,i} \cdot (1 - p_{k,i}) \geq k$. The Lindeberg-Feller Theorem applied to the triangular array $\left(\frac{X_{k,i}-p_i}{\sigma_k}\right)$ yields

$$\lim_{k\to\infty} P\left[a < \frac{S_k - \mu_k}{\sigma_k} < b\right] = \varphi(b) - \varphi(a) \ . \tag{3}$$

For S_k to hit a given interval of length $2l(k)$ (like the interval $[s_k - l(k), s_k + l(k)]$ in (2)) it is necessary for $(S_k - \mu_k)/\sigma_k$ to hit a given interval of length $2l(k)/\sigma_k$. Note that $\lim_{k\to\infty} l(k)/\sigma_k = 0$ because $\sigma_k \geq \sqrt{k}$ and $l(k) = o(\sqrt{k})$. Thus the hitting probability approaches 0 as k approaches infinity. This contradicts to (2). ◀

For ease of later reference, we let $k(\beta)$ for $\beta > 0$ be a function such that $h(k) \leq \beta$ for all $k \geq k(\beta)$. (Such a function must exist according to Lemma 7.)

▶ **Corollary 8.** *With the notation and assumptions from Lemma 7, the following holds: the probability mass of the mode of $\sum_{i=1}^{n_k} X_{k,i}$ is at most β for all $k \geq k(\beta)$.*

The following result implies the unimodality of binomially distributed random variables:

▶ **Lemma 9** ([10]). *Every sum of independent Bernoulli variables (with possibly different probabilities of success) is unimodal.*

2.2 Prerequisites from Learning Theory

A *concept class C over domain X* is a family of functions from X to $\{0,1\}$. C is said to be *PAC-learnable with sample size* $m(\varepsilon, \delta)$ if there exists a (possibly randomized) algorithm A with the following property. For every concept $c \in C$, for every distribution P on X, and for all $\varepsilon, \delta > 0$ and $m = m(\varepsilon, \delta)$, if $\vec{x} = (x_1, \ldots, x_m)$ is drawn at random according to P^m, $\vec{b} = (c(x_1), \ldots, c(x_m))$, and A is given access to $\varepsilon, \delta, \vec{x}, \vec{b}$, then, with probability greater than $1 - \delta$, A outputs a hypothesis $h : X \to \{0,1\}$ such that $P[h(x) = c(x)] > 1 - \varepsilon$. We say that h is *ε-accurate* (resp. ε-inaccurate) if $P[h(x) = c(x)] > 1 - \varepsilon$ (resp. $P[h(x) \neq c(x)] \geq \varepsilon$). We say the learner *fails* when he returns an ε-inaccurate hypothesis. As mentioned in the introduction already, we refer to ε as the *accuracy* and to δ as the *confidence*. In this paper, we consider the following variations of the basic model:

Proper PAC-learnability: The hypothesis $h : X \to \{0,1\}$ must be a member of C.

PAC-learnability under a fixed distribution: P is fixed and known to the learner.

The semi-supervised setting: The input of the learning algorithm is augmented by a finite number (depending on the various parameters of the learning task) of unlabeled samples. All samples, labeled- and unlabeled-ones, are drawn independently from X according to the domain distribution P.

Note that PAC-learnability with sample size $m(\varepsilon, \delta)$ under a fixed distribution follows from PAC-learnability with sample size $m(\varepsilon, \delta)$ in the semi-supervised setting because, if A knows the domain distribution P, it can first generate sufficiently many unlabeled samples and then run a simulation of the semi-supervised learning algorithm.

Throughout the paper, a mapping from X to $\{0,1\}$ is identified with the set of instances from X that are mapped to 1. Thus, concepts are considered as mappings from X to $\{0,1\}$ or, alternatively, as subsets of X. (E.g., we may write $P(h \oplus c)$ instead of $P[h(x) \neq c(x)]$.) $X' \subseteq X$ is said to be *shattered by* C if $\{X' \cap c | \; c \in C\}$ coincides with the powerset of X'. The VC-dimension of C, denoted $\mathrm{VCdim}(C)$, is infinite if there exist arbitrarily large sets that are shattered by C, and it is the size of the largest set shattered by C otherwise. We remind the reader to the following well-known results:

▶ **Lemma 10** ([4]). *A finite class C is properly PAC-learnable by any consistent hypothesis finder from $\lceil \ln(|C|/\delta)/\varepsilon \rceil$ labeled samples.*

▶ **Lemma 11** ([5]). *A class C of finite VC-dimension is properly PAC-learnable by any consistent hypothesis finder from $O((\mathrm{VCdim}(C) \cdot \log(1/\varepsilon) + \log(1/\delta))/\varepsilon)$ labeled samples.*

$C' \subseteq C$ is called an *ε-covering of C with respect to P* if for any $c \in C$ there exists $c' \in C'$ such that $P(c \oplus c') < \varepsilon$. The *covering number* $N_{C,P}(\varepsilon)$ is the size of the smallest ε-covering of C with respect to P. With this notation, the following holds:

▶ **Lemma 12** ([3]). *A concept class C is properly PAC-learnable under a fixed distribution P from $O(\log(N_{C,P}(\varepsilon/2)/\delta)/\varepsilon)$ labeled samples.*

A result by Balcan and Blum[5] implies the same upper bound on the label complexity for semi-supervised algorithms and concept classes of finite VC-dimension:

▶ **Lemma 13** ([1]). *Let C be a concept class of finite VC-dimension. Then C is PAC-learnable in the semi-supervised setting from $O(\mathrm{VCdim}(C)\log(1/\varepsilon)/\varepsilon^2 + \log(1/\delta)/\varepsilon^2)$ unlabeled and $O(\log(N_{C,P}(\varepsilon/6)/\delta)/\varepsilon)$ labeled samples.*

The following game between the learner and his adversary is useful for proving lower bounds on the sample size m:

Step 1: An "adversary" fixes a probability distribution D on pairs of the form (c, P) where $c \in C$ and P is a probability distribution on the domain X.

Step 2: The target concept c and the domain distribution P (representing the learning task) are chosen at random according to D.

Step 3: $(x_1, \ldots, x_m)$ is drawn at random according to P^m, and $\varepsilon, \delta, (x_1, \ldots, x_m), (b_1, \ldots, b_m)$ such that $b_i = c(x_i)$ is given as input to the learner.

Step 4: The adversary might give additional pieces of information to the learner.[6]

Step 5: The learner returns a hypothesis h. He "fails" if $P[h(x) \neq c(x)] \geq \varepsilon$.

This game differs from the PAC-learning model mainly in two respects. First, the learner is not evaluated against the pair (c, P) on which he performs worst but on a pair (c, P) chosen at random according to D (albeit D is chosen by an adversary). Second, the learner possibly obtains additional pieces of information in Step 4. Since both maneuvers can be to the advantage of the learner only, they do not compromise the lower bound argument. Thus, if we can show that, with probability at least δ, the learner fails in the above game, we may conclude that the sample size m does not suffice to meet the (ε, δ)-criterion of PAC-learning. Moreover, according to Yao's principle [12], lower bounds obtained by this technique even apply to randomized learning algorithms.

3 The Semi-supervised Versus the Purely Supervised Setting

This section is devoted to the proofs of our main results. The proof for Theorem 1 is presented in Section 3.1. The proofs for Theorems 2 and 3 are presented in Section 3.2.

3.1 Proof of Theorem 1

We start with the following lower bound on $m_{C,P}(\varepsilon, \delta)$:

▶ **Lemma 14.** *Let C be a concept class and let P be a distribution on domain X. For any $\varepsilon > 0$, let*

$$\lceil \varepsilon \rceil_{C,P} = \min\{\varepsilon' \mid (\varepsilon' \geq \varepsilon) \wedge (\exists c, c' \in C : P(c \oplus c') = \varepsilon'\}$$

[5] Apply Theorem 13 from [1] with a constant compatibility of 1 for all concepts and distributions.
[6] This step has purely proof-technical reasons: sometimes the analysis becomes simpler when the power of the learner is artificially increased.

where, by convention, the minimum of an empty set equals ∞. With this notation, the following holds:

1. *If $\lceil 2\varepsilon \rceil_{C,P} \le 1$, then $m_{C,P}(\varepsilon, \delta) \ge 1$.*
2. *Let $\gamma = 1 - \lceil 2\varepsilon \rceil_{C,P}$. If $\lceil 2\varepsilon \rceil_{C,P} < 1$, then*

$$m_{C,P}(\varepsilon, \delta) \ge \log_{1/\gamma} \frac{1}{2\delta} = \Omega\left(\log_{1/\gamma} \frac{1}{\delta}\right) \quad . \tag{4}$$

3. *If $\lceil 2\varepsilon \rceil_{C,P} \le 1/4$, then*

$$m_{C,P}(\varepsilon, \delta) \ge \left\lfloor \frac{\ln(1/(2\delta))}{2\lceil 2\varepsilon \rceil_{C,P}} \right\rfloor = \Omega\left(\frac{\ln(1/\delta)}{\lceil 2\varepsilon \rceil_{C,P}}\right) \quad . \tag{5}$$

Proof. It is easy to see that at least one labeled sample is needed if $\lceil 2\varepsilon \rceil_{C,P} \le 1$. Let us now assume that $\lceil 2\varepsilon \rceil_{C,P} < 1$. Let $c, c' \in C$ be chosen such that $P(c \oplus c') = \lceil 2\varepsilon \rceil_{C,P}$. The adversary picks c and c' as target concept with probability 1/2, respectively. With a probability of $(1 - \lceil 2\varepsilon \rceil_{C,P})^m$, none of the labeled samples hits $c \oplus c'$. Since $P(c \oplus c') \ge 2\varepsilon$, the learner has no hypothesis at his disposal that is ε-accurate for c and c'. Thus, if none the samples distinguishes between c and c', the learner will fail with a probability of 1/2. We can conclude that the learner fails with an overall probability of at least $\frac{1}{2}(1 - \lceil 2\varepsilon \rceil_{C,P})^m = \frac{1}{2}\gamma^m$.

Setting this probability less than or equal to δ and solving for m leads to the lower bound (4). If $\lceil 2\varepsilon \rceil_{C,P} \le 1/4$, a straightforward computation shows that $\frac{1}{2}\gamma^m$ is bounded from below by $\frac{1}{2}\exp(-2\lceil 2\varepsilon \rceil_{C,P} m)$. Setting this expression less than or equal to δ and solving for m leads to the lower bound (5). ◀

We are ready now for the **Proof of Theorem 1:**
We use the notation from Lemma 14. We first present the main argument under the (wrong!) assumption that $\lceil \varepsilon \rceil_{C,P}$ is known to the learner. At the end of the proof, we explain how a fully supervised learning algorithm can compensate for not knowing P. The first important observation, following directly from the definition of $\lceil \varepsilon \rceil_{C,P}$, is that, in order to achieve an accuracy of ε, it suffices to achieve an accuracy $\lceil \varepsilon \rceil_{C,P}$ with a hypothesis from C. Thus, for the purpose of Theorem 1, it suffices to have a supervised proper learner that achieves accuracy $\lceil 2\varepsilon \rceil_{C,P}$ with confidence δ. We proceed with the following case analysis:

Case 1: $\lceil 2\varepsilon \rceil_{C,P} \le 1/4$.
There is a gap of $O(\ln |C|)$ only between the upper bound from Lemma 10 (with $\lceil 2\varepsilon \rceil_{C,P}$ in the role of ε) and the lower bound (5). Returning a consistent hypothesis, so that Lemma 10 applies, is appropriate in this case.

Case 2: $1/4 < \lceil 2\varepsilon \rceil_{C,P} < 15/16$.
We may argue similarly as in Case 1 except that the upper bound from Lemma 10 is compared to the lower bound (4). (Note that $\gamma = \theta(1)$ in this case.) As in Case 1, returning a consistent hypothesis is appropriate.

Case 3: $15/16 < \lceil 2\varepsilon \rceil_{C,P} < 1$.
In this case $0 < \gamma = 1 - \lceil 2\varepsilon \rceil_{C,P} < 1/16$. The learner will exploit the fact that one of the hypotheses $\emptyset$ and X is a good choice. He returns hypothesis X if label "1" has the majority within the labeled samples, and hypothesis $\emptyset$ otherwise. Let c, as usual, denote the target concept. If $\gamma < P(c) < 1 - \gamma$, then both of $\emptyset$ and X are $\lceil 2\varepsilon \rceil_{C,P}$-accurate. Let us assume that $P(c) \le \gamma$. (The case $P(c) \ge 1 - \gamma$ is symmetric.) The learner will fail only if, despite of the small probability γ for label "1", these labels have the majority. It is easy to see that the probability for this to happen is bounded by $(m/2)\binom{m}{m/2}\gamma^{m/2}$ and therefore also bounded by $2^{3m/2}\gamma^{m/2} = (8\gamma)^{m/2}$. Setting the last expression less than

or equal to δ and solving for m reveals that $O(\log_{1/\gamma}(1/\delta))$ many labeled samples are enough. This matches the lower bound (4) modulo a constant factor.

Case 4: $\lceil 2\varepsilon \rceil_{C,P} = 1$.
This is a trivial case where each labeled sample almost surely makes inconsistent any hypothesis $h \in C$ of error at least ε. The learner may return any hypothesis that is supported by at least one labeled sample.

Case 5: $\lceil 2\varepsilon \rceil_{C,P} = \infty$.
This is another trivial case where any concept from C is 2ε-accurate with respect to any other concept from C. The learner needs no labeled example and may return any $h \in C$.

In any case, the "label-complexity" gap is bounded by $O(\ln |C|)$. We finally have to explain how this can be exploited by a supervised learner A who does not have any prior knowledge of P. The main observation is that, according to the bound in Lemma 10, the condition $m > \lceil \ln(|C|/\delta)/(15/16) \rceil$ indicates that the sample size is large enough to achieve an accuracy below 15/16 so that returning a consistent hypothesis is the appropriate action (as in Cases 1 and 2 above). If, on the other hand, the above condition on m is violated, then A will set either $h = \emptyset$ or $h = X$ depending on which label holds the majority (which would also be an appropriate choice in Cases 3 and 4 above). It is not hard to show that this procedure leads to the desired performance, which concludes the proof for the first part of Theorem 1.
As for the second part, one can use a similar argument that employs Lemma 11 instead of Lemma 10.

3.2 Proof of Theorems 2 and 3

Throughout this section, we set $X_n = \{0,1\}^n$ and $X_* = \{0,1\}^*$. We will identify a finite string $x \in X_*$ with the infinite string that starts with x and ends with an infinite sequence of zeros. C_* denotes the family of functions $c_i : X_* \to \{0,1\}$, $i \in \mathbb{N} \cup \{0\}$, given by $c_0(x) = 0$ and $c_i(x) = x_i$ for all $i \geq 1$. Note that $c_i(x) = 0$ for all $i > |x|$. C_n denotes the class of functions obtained by restricting a function from C_* to the subdomain X_n. For every $i \geq 1$, let $p_i = 1/\log(3+i)$. For every permutation σ of $1, \ldots, n$, let P_σ be the probability measure on X_n obtained by setting $x_{\sigma(i)} = 1$ with probability p_i (resp. $x_{\sigma(i)} = 0$ with probability $1 - p_i$) independently for $i = 1, \ldots, n$. $\mathcal{P}_n = \{P_\sigma\}$ denotes the family of all such probability measures on X_n. Note that P_σ can also be considered as a probability measure on X_* (that is centered on X_n). $\mathcal{P}_*$, a family of probability measures on X_*, is defined as $\cup_{n \geq 1} \mathcal{P}_n$.

▶ **Lemma 15.** *1. C_* is properly PAC-learnable under any fixed distribution $P_\sigma \in \mathcal{P}_*$ from $O(1/\varepsilon^2 + \log(1/\delta)/\varepsilon)$ labeled samples.*
2. For any (unknown) $P_\sigma \in \mathcal{P}_*$, C_* is properly PAC-learnable in the semi-supervised setting from $O(\log(n/\delta)/\varepsilon)$ unlabeled and $O(1/\varepsilon^2 + \log(1/\delta)/\varepsilon)$ labeled samples. Here, n denotes the smallest index such that $P_\sigma \in \mathcal{P}_n$.
3. There exists a semi-supervised algorithm A that PAC-learns C_n under any unknown domain distribution. Moreover, for all $P \in \mathcal{P}_n$, $m^A_{C_n,P}(\varepsilon, \delta) = O(1/\varepsilon^2 + \log(1/\delta)/\varepsilon)$.

Proof. 1. Let σ be a permutation of $1, \ldots, n$. For all $i > n$: $c_i = \emptyset$ almost surely w.r.t. P_σ. For all $2^{2/\varepsilon} - 3 \leq i \leq n$: $P_\sigma[c_{\sigma(i)} \oplus \emptyset] = P_\sigma[c_{\sigma(i)}] = p_i \leq \varepsilon/2$. Thus, setting $N = \lceil 2^{2/\varepsilon} \rceil - 4$, $\{\emptyset, c_{\sigma(1)}, \ldots, c_{\sigma(N)}\}$ forms an $\varepsilon/2$-covering of C_* with respect to P_σ. An application of Lemma 12 now yields the result.

2. The very first unlabeled sample reveals the parameter n such that the unknown measure P_σ is centered on X_n. Note that, for every $i \in [n]$, $x_i = 1$ with probability $p_{\sigma^{-1}(i)}$. It is an easy application of the multiplicative Chernov-bound (combined with the Union-bound) to see that $O(\log(n/\delta)/\varepsilon)$ unlabeled samples suffice to retrieve (with probability $1 - \delta/2$

of success) an index set $I \subset [n]$ with the following properties. On one hand, I includes all $i \in [n]$ such that $p_{\sigma^{-1}(i)} \geq \varepsilon/2$. On the other hand, I excludes all $i \in [n]$ such that $p_{\sigma^{-1}(i)} \leq \varepsilon/8$. Consequently $\{\emptyset\} \cup \{c_i | \ i \in I\}$ is an $\varepsilon/2$-covering of C_n with respect to P_σ and its size is bounded by $1 + |I| \leq 2^{8/\varepsilon}$. Another application of Lemma 12 now yields the result.

3. The third statement in Lemma 15 is an immediate consequence of Lemma 13 and the fact that, as proved above, $N_{C_n}(\varepsilon/6) = 2^{O(1/\varepsilon)}$ (regardless of the value of n). ◀

▶ **Lemma 16.** *Let A be a fully supervised algorithm designed to PAC-learn C_* under any unknown distribution taken from $\mathcal{P}_*$. For every finite sample size m and for all $\alpha, \beta > 0$, an adversary can achieve the following: with a probability of at least $1 - \alpha$ the hypothesis returned by A has an error of at least $1/2 - \beta$.*[7]

Proof. The proof will run through the following stages:
1. We first fix some technical notations and conditions (holding in probability) which the proof builds on.
2. Then we specify the strategy of the learner's adversary.
3. We argue that, given the strategy of the adversary, the learner has probably almost no advantage over random guesses.
4. We finally verify the technical conditions.

Let us start with Stage 1. (Though somewhat technical it will help us to provide a precise description of the subsequent stages.) Let $M \in \{0,1\}^{(m+1)\times(\mathbb{N}\setminus\{1\})}$ be a random matrix (with columns indexed by integers not smaller than 2) such that the entries are independent Bernoulli variables where the variable $M_{i,j}$ has probability $p_j = 1/\log(3+j) < 1/2$ of success. Let $M(n)$ denote the finite matrix composed of the first $n-1$ columns of M. Let $k = \max\{\lceil 1/\alpha \rceil, k(2\beta)\}$ where $k(\beta)$ is the function from the remark right after Lemma 7. In Stage 4 of the proof, we will show that there exists $n = n_k \in \mathbb{N}$ such that, with probability at least $1 - 1/k$, the following conditions are valid for each bit pattern $b \in \{0,1\}^{m+1}$:

(A) $b \in \{0,1\}^{m+1}$ coincides with at least $4k^2$ columns of $M(n)$.

(B) Let $b' \in \{0,1\}^m$ be the bit pattern obtained from b by omission of the final bit. Call column $j \geq 2$ of $M(n)$ "marked" if its first m bits yield pattern b'. Let $I \subseteq \{2, \ldots, n\}$ denote the set of indices for marked columns. Then, $\sum_{i \in I} p_i \geq 2k$ so that $\sum_{i \in I} p_i(1-p_i) \geq k$ (because $p_i < 1/2$).

The strategy of the adversary (Stage 2 of the proof) is as follows: she sets $n = n_k$, picks a permutation σ of $1, \ldots, n$ uniformly at random, chooses domain distribution P_σ, and selects the target concept c_t such that $t = \sigma(1)$. In the sequel, probabilities are simply denoted $P[\cdot]$. Note that the component x_t of a sample x can be viewed as a fair coin since $P[x_t = 1] = p_1 = 1/\log(4) = 1/2$. The learning task resulting from this setting is related to the technical definitions and conditions from Stage 1 as follows:

- The first m rows of the matrix $M(n)$ are the components $\sigma(2), \ldots, \sigma(n)$ of the m labeled samples.
- The bits of $b' \in \{0,1\}^m$ are the t-th components of the m labeled samples. These bits are perfectly random, and they are identical to the classification labels.
- The set $I \subseteq \{2, \ldots, n\}$ points to all marked columns of $M(n)$, i.e., it points to all columns of $M(n)$ which are duplicates of b'.
- Row $m+1$ of M represents an unlabeled test sample that has to be classified by the learner.

[7] Loosely speaking, the learner has "probably almost no advantage over random guesses".

The adversary passes also the set $J = \{\sigma(i) | \ i \in I \cup \{1\}\}$, with the understanding that index t of the target concept is an element of J, and the set $I \subseteq \{2, \dots, n\}$ as additional information to the learner. This maneuver marks the end of Stage 2 in our proof.
We now move on to Stage 3 of the proof and explain why the strategy of the adversary leads to a poor learning performance (thereby assuming that conditions (A) and (B) hold). Note that, by symmetry, every index in J has the same a-posteriori probability to coincide with t. Because the learner has no way to break the symmetry between the indices in J before he sees the test sample x, the best prediction for the label of x does not depend on the individual bits in x but only on the number of ones in the bit positions from J, i.e., it only depends on the value of

$$Y' = \sum_{j \in J} x_j = x_{\sigma(1)} + \sum_{i \in I} x_{\sigma(i)} = x_{\sigma(1)} + Y \quad \text{where} \quad Y = \sum_{i \in I} x_{\sigma(i)} = \sum_{i \in I} M_{m+1,i} \ .$$

Note that the learner knows the distribution of Y (given by the parameters $(p_i)_{i \in I}$) since the set I had been passed on to him by the adversary. For sake of brevity, let $\ell = x_{\sigma(1)}$ denote the classification label of the test sample x. Given a value s of Y' (and the fact that the a-priori probabilities for $\ell = 0$ and $\ell = 1$ are equal), the Bayes decision is in favor of the label $\ell \in \{0, 1\}$ which maximizes $P[Y' = s|\ell]$. Clearly, $P[Y' = s|\ell = 1] = P[Y = s - 1]$ and $P[Y' = s|\ell = 0] = P[Y = s]$. Thus, the Bayes decision is in favor of $\ell = 0$ if and only if $P[Y = s] \geq P[Y = s - 1]$. Since Y is a sum of independent Bernoulli variables, we may apply Lemma 9 and conclude that Y has a unimodal distribution. It follows that the Bayes decision is the following threshold function: be in favor of $\ell = 0$ iff $Y' \leq \text{mode}(Y)$. The punchline of this discussion is as follows: the Bayes decision is independent of the true label ℓ unless Y hits its mode (so that $Y' = Y + \ell$ is either mode(Y) or mode$(Y) + 1$). It follows that the Bayes error is at least $(1 - P[Y = \text{mode}(Y)])/2$. Because of Condition (B) and the fact that $k \geq k(2\beta)$, we may apply Corollary 8 and obtain $P[Y = \text{mode}(Y)] \leq 2\beta$ so that the Bayes error is at least $1/2 - \beta$.
We finally enter Stage 4 of the proof and show that conditions (A) and (B) hold with a probability of at least $1 - \alpha$ provided that $n = n_k$ is large enough. Let b range over all bit patterns from $\{0, 1\}^{m+1}$. Consider the events

- $A_r(b)$: $b \in \{0, 1\}^{m+1}$ coincides with the r-th column of M.
- $B_{k,n}(b)$: $b \in \{0, 1\}^{m+1}$ coincides with at least k columns of $M(n)$.

It is easy to see that $\sum_{r=1}^{\infty} P(A_r(b)) = \infty$. Applying the Borel-Cantelli Lemma to the events $(A_r(b))_{r \geq 1}$ and Corollary 5 to the events $(B_{4k^2,n}(b))_{k,n \geq 1}$, we arrive at the following conclusion. There exists $n_k(b) \in \mathbb{N}$ such that, for all $n \geq n_k(b)$, the probability of $B_{4k^2,n}(b)$ is at least $1 - 1/(2^{m+3}k)$. We set $n = n_k = \max_b n_k(b)$. Then, the probability of $B_{4k^2,n} = \bigcap_{b \in \{0,1\}^{m+1}} B_{4k^2,n}(b)$ is at least $1 - 1/(4k)$. In other words: with a probability of at least $1 - 1/(4k)$, each $b \in \{0, 1\}^{m+1}$ coincides with at least $4k^2$ columns of $M(n)$. Thus condition (A) is violated with a probability of at most $1/(4k)$.
We move on to condition (B). With $p = \sum_{i \in I} p_i$, we can decompose $P[B_{4k^2,n}]$ as follows:

$$P[B_{4k^2,n}] = P\left[B_{4k^2,n} \,|p < 2k\right] \cdot P\left[p < 2k\right] + P\left[B_{4k^2,n} \,|p \geq 2k\right] \cdot P\left[p \geq 2k\right]$$

Note that, according to the definitions of $B_{4k^2,n}(b)$ and $B_{4k^2,n}$, event $B_{4k^2,n}$ implies that $Y \geq 4k^2$ because there must be at least $4k^2$ occurrences of 1 in row $m + 1$ and in the marked columns of $M(n)$. On the other hand, $\mathbb{E}[Y] = p$. According to Markov's inequality, $P\left[Y \geq 4k^2|p < 2k\right] \leq (2k)/(4k^2) = 1/(2k)$. Thus, $P[B_{4k^2,n}] \leq 1/(2k) + P\left[p \geq 2k\right]$. Recall that, according to condition (A), $1 - 1/(4k) \leq P[B_{4k^2,n}]$. Thus, $P\left[p \geq 2k\right] \geq 1 - 1/(4k) - 1/(2k) = 1 - 3/(4k)$. Since $k \geq 1/\alpha$, we conclude that the probability to violate one of the conditions (A) and (B) is bounded by $1/k \leq \alpha$. ◀

We are now ready to complete the proofs of our main results. Theorem 2 is a direct consequence of the second statement in Lemma 15 and of Lemma 16. The first part of Theorem 3 is a direct consequence of the third statement in Lemma 15. As for the second part, an inspection of the proof of Lemma 16 reveals that the adversary argument uses a "finite part" C_n of C_* only (with n chosen sufficiently large).

4 Final Remarks:

As we have seen in this paper, it is impossible to show in full generality that unlabeled samples have a marginal effect only in the absence of any compatibility assumptions. It would be interesting to explore which concept classes are similar in this respect to the artificial classes C_* and $(C_n)_{n\geq 1}$ that were discussed in this paper. We would also like to know if the bounds of Theorem 1 are tight (either for special classes or for the general case). It would be furthermore interesting to extend our results to the agnostic setting.

References

1. Maria-Florina Balcan and Avrim Blum. A discriminative model for semi-supervised learning. *Journal of the Association on Computing Machinery*, 57(3):19:1–19:46, 2010.
2. Shai Ben-David, Tyler Lu, and Dávid Pál. Does unlabeled data provably help? Worst-case analysis of the sample complexity of semi-supervised learning. In *Proceedings of the 21st Annual Conference on Learning Theory*, pages 33–44, 2008.
3. Gyora M. Benedek and Alon Itai. Learnability with respect to fixed distributions. *Theoretical Computer Science*, 86(2):377–389, 1991.
4. Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Occam's razor. *Information Processing Letters*, 24:377–380, 1987.
5. Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the Association on Computing Machinery*, 36(4):929–965, 1989.
6. Kai Lai Chung. *A Course in Probability Theory.* Academic Press, 1974.
7. Malte Darnstädt and Hans U. Simon. Smart PAC-learners. *Theoretical Computer Science*, 412(19):1756–1766, 2011.
8. Richard M. Dudley, Sanjeev R. Kulkarni, Thomas J. Richardson, and Ofer Zeitouni. A metric entropy bound is not sufficient for learnability. *IEEE Transactions on Information Theory*, 40(3):883–885, 1994.
9. William Feller. *An Introduction to Probability Theory and its Applications*, volume 1. John Wiley & Sons, 1968.
10. Julian Keilson and Hans Gerber. Some results for discrete unimodality. *Journal of the American Statistical Association*, 66(334):386–389, 1971.
11. Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.
12. Andrew Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proceedings of the 18th Symposium on Foundations of Computer Science*, pages 222–227, 1977.

FO^2 with one transitive relation is decidable

Wiesław Szwast[1] **and Lidia Tendera**[*1]

1 **Institute of Mathematics and Informatics,**
Opole University, Oleska 48, 45-052 Opole, Poland
`[szwast,tendera]@math.uni.opole.pl`

Abstract

We show that the satisfiability problem for the two-variable first-order logic, FO^2, over transitive structures when only one relation is required to be transitive, is decidable. The result is optimal, as FO^2 over structures with two transitive relations, or with one transitive and one equivalence relation, are known to be undecidable, so in fact, our result completes the classification of FO^2-logics over transitive structures with respect to decidability. We show that the satisfiability problem is in 2-NExpTime. Decidability of the finite satisfiability problem remains open.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.1 Mathematical Logic, F.4.3 Formal Languages

Keywords and phrases classical decision problem, two-variable first-order logic, decidability, computational complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.317

1 Introduction

FO^2 is the restriction of the classical first-order logic over relational signatures to formulae with at most two distinct variables. It is well-known that FO^2 enjoys the finite model property [20], and its satisfiability (hence also finite satisfiability) problem is NExpTime-complete [5].

One particular drawback of FO^2 is that it can neither express transitivity of a binary relation nor say that a binary relation is a partial (or linear) order, or an equivalence relation. These natural properties are important for practical applications, thus research has started to investigate FO^2 over restricted classes of structures in which some distinguished binary symbols are required to be interpreted as transitive relations, orders, equivalences, etc. The idea comes from modal correspondence theory, where various conditions on the accessibility relations allow to restrict the class of Kripke structures considered, e.g. to transitive structures for the modal logic K4 or equivalence structures for the modal logic S5. Orderings, on the other hand, are very natural when considering temporal logics, where they model time flow, but they also are used in different scenarios, e.g. in databases or description logics, to compare objects with respect to some parameters.

Unfortunately, the remarkably robust decidability of modal logics and its various extensions towards greater expressibility does not transfer immediately to extensions of FO^2, and the picture for FO^2 is more complex and to some extent less understood. It appeared that both the satisfiability and the finite satisfiability problems for FO^2 are undecidable in the presence of several equivalence or several transitive relations [6, 7]. These results were later

* Lidia Tendera would like to acknowledge the support of Polish Ministry of Science and Higher Education grant N N206 37133.

strengthened: FO^2 is undecidable in the presence of two transitive relations [11, 9], three equivalence relations [15], one transitive and one equivalence relation [17], or three linear orders [12].

On the positive side it is known that FO^2 with one or two equivalence relations is decidable [16, 17, 14]. The same holds for FO^2 with one linear order [22]. The intriguing questions left open by this research was the case of FO^2 with one transitive relation and FO^2 with two linear orders.

In this paper we answer the first question positively: we prove that the satisfiability problem for the extension of FO^2 where exactly one binary relation is required to be transitive, FO^2_T, is decidable in 2-NExpTime. The result completes the classification of variants of FO^2 over transitive structures with respect to decidability.

For the special case of two linear orders, ExpSpace-completeness of finite satisfiability is shown, subject to certain restrictions on signatures, in [24]. (The case of unrestricted signatures, and decidability of the general satisfiability problem are currently open.)

It is also worth to compare the above results with results concerning GF^2 i.e. the two-variable restriction of the *guarded fragment* GF [1] where quantifiers are guarded by atoms. GF+TG is the restriction of GF^2 with transitive relations where the transitive relation symbols are allowed to appear only in guards. As shown in [26] undecidability of FO^2 with transitivity transfers to GF^2 with transitivity; however, GF+TG is decidable irrespective of the number of transitive symbols. Moreover, as noted in [11], the decision procedure developed for GF^2+TG can be applied to GF^2 with one transitive relation that is allowed to appear also outside guards, giving 2-ExpTime-completeness of the latter fragment.

Also of note in this context is the interpretation of FO^2 over *data words* and *data trees* that appear e.g. in verification and XML processing. Decidability of FO^2 over data words with one additional equivalence relation was shown in [3]. For more results related to FO^2 over data words or data trees see e.g. [18, 24, 4, 21, 2].

It makes sense to also consider more expressive systems in which we may refer to the transitive closure of some relation. In fact, relatively few decidable fragments of first-order logic with transitive closure are known. One exception is the logic GF^2 with a transitive closure operator applied to binary symbols appearing only in guards [19]. This fragment captures the two-variable guarded fragment with transitive guards, GF^2+TG, preserving its complexity [27, 10]. Also decidable is the satisfiability problem for the logic $\exists\forall(\text{DTC}^+[E])$, i.e. the prefix class $\exists\forall$ extended by the positive deterministic transitive closure operator of one binary relation, which is shown to enjoy the exponential model property [8]. Recently, it has been shown that the satisfiability problem for the two-variable universal fragment of first-order logic with constants remains decidable when extended by the transitive closure of a single binary relation [13]. Whether the same holds for full FO^2 is open.

Expressive power of FO^2_T. As has already been mentioned, FO^2 has the finite model property. Adding one transitive relation to GF^2 (even restricted only to guards) we can write infinity axioms, however models for this logic still enjoy the so called tree-like property, i.e. new elements required by $\forall\exists$-conjuncts can be added independently. Below we give an example of an infinity axiom in FO^2_T that enforces models where in some triples all elements depend on each other.

We use the transitive relation symbol T and two unary symbols P and Q. It is not difficult to formalize the following statements by an FO^2_T formula: (a) T is strictly antisymmetric. (b) Elements of P form one infinite chain. (c) Elements of Q are incomparable. (d) Every element of P has an incomparable element in Q. (e) Every element of Q is smaller than some element in P.

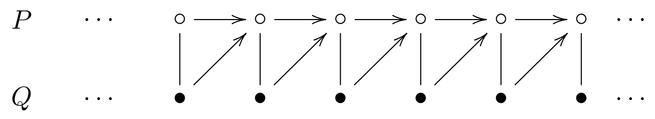


Figure 1 A model satisfying (a)-(e). Arrows depict elements related by T. Lines connect elements required by (d), not connected by T.

In any model satisfying (a)-(e) there is an infinite chain of elements in P that induces an infinite antichain of elements in Q (see Figure 1). Note also, that it suffices that the unique transitive relation is supposed to be a partial ordering.

Outline of the proof. Models for our logic, taking into account the interpretation of the transitive relation, can obviously be seen as partitioned into cliques. As usually for two-variable logics, we first establish a “Scott-type” normal form for FO^2_T: $\forall\forall\ \wedge \bigwedge\forall\exists$, allowing us to restrict the nesting of quantifiers to depth two, as well as to concentrate on the $\forall\exists$-conjuncts demanding “witnesses” for all elements in a model. The form of the $\forall\exists$-conjuncts enables to distinguish witnesses required inside cliques (i.e. realizing a 2-type containing both $T(x,y)$ and $T(y,x)$, see Section 2 for a precise definition) from witnesses outside cliques. We also establish a *small clique property* for FO^2_T (practically the same as in [15]), allowing us to restrict attention to models with cliques exponentially bounded in the size of the signature. Further constructions proceed on levels of cliques rather than individual elements. (An alternative approach would be to consider first the satisfiability problem over an antisymmetric relation T and then reduce the general problem to the aforementioned one taking into account the bound on the clique sizes.)

Crucial to our argument is this property: any infinitely satisfiable sentence has an infinite *narrow* model, i.e. a model whose universe can be partitioned into segments (i.e. sets of cliques) $S_0, S_1, \ldots$, each of doubly exponential size, such that every element in $\bigcup_{i=0}^{j-1} S_i$ requiring a witness outside its clique has the witness either in S_0 or in S_j (so, in every S_k, $k \geq j$, Def. 16). This immediately implies that, when needed, every single segment S_j ($j > 0$) can be removed from the structure, to yield a model with new properties.

To prove existence of narrow models, we first make some useful observations. In particular, we show that a single clique can be duplicated, provided its type called *splice* appears at least twice in a model (Claim 7). The property is used to show the main technical result (Claim 10 and Corollary 11). Next, the idea is generalized in Lemma 13 to show that for any finite subset F of elements, the model can be extended by a fixed number of cliques (depending only on the signature, and not depending on the cardinality of F) providing all required witnesses for elements from F.

As the main result of the paper, we show that from any narrow model we can build a *canonical model* where every two segments of the infinite partition (except the first) are isomorphic and they are connected using at most two distinct similarity types (Def. 19). In fact, these constructions can be seen as an application of the infinite Ramsey theorem [23], where segments of the models are considered to be nodes in a colored graph, and similarity types of pairs of segments are colors of edges.

The above properties suffice to obtain the 2-NExpTime decision procedure for the satisfiability problem for FO^2_T given in Theorem 21 and Corollary 22. We note however that the best lower bound coming from GF^2+TG is 2-ExpTime, thus our result leaves a gap in complexity. We also note that our decision procedure cannot be straightforwardly generalized to solve the finite satisfiability problem for FO^2_T and to the best of our knowledge, the latter problem remains open (see Outlook for some discussion).

2 Preliminaries

We denote by FO^2 the two-variable fragment of first-order logic (with equality) over relational signatures. By FO^2_T we understand the set of FO^2_T-formulas over any signature $\sigma = \sigma_0 \cup \{T\}$, where T is a distinguished binary predicate. The semantics for FO^2_T is as for FO^2, subject to the restriction that T is always interpreted as a *transitive* relation.

In this paper, σ-structures are denoted by Gothic capital letters and their universes by corresponding Latin capitals. Where a structure is clear from context, we frequently equivocate between predicates and their realizations, thus writing, for example, R in place of the technically correct $R^{\mathfrak{A}}$. If $\mathfrak{A}$ is a σ-structure and $B \subseteq A$, then $\mathfrak{A} \restriction B$ denotes the substructure of $\mathfrak{A}$ with the universe B.

An (atomic and proper) *k-type* (over a given signature) is a maximal consistent set of atoms or negated atoms over k distinct variables not containing equality atoms $x_i = x_j$ with $i \neq j$. If $\beta(x, y)$ is a 2-type over variables x and y, then $\beta \restriction x$ (respectively, $\beta \restriction y$) denotes the unique 1-type that is obtained from β by removing atoms with the variable y (respectively, the variable x). We denote by $\boldsymbol{\alpha}$ the set of all 1-types and by $\boldsymbol{\beta}$ the set of all 2-types (over a given signature). Note that $|\boldsymbol{\alpha}|$ and $|\boldsymbol{\beta}|$ are bounded exponentially in the size of the signature. We often identify a type with the conjunction of all its elements.

For a given σ-structure $\mathfrak{A}$ and $a \in A$ we say that a *realizes* a 1-type α if α is the unique 1-type such that $\mathfrak{A} \models \alpha[a]$. We denote by $tp^{\mathfrak{A}}(a)$ the 1-type realized by a. Similarly, for distinct $a, b \in A$, we denote by $tp^{\mathfrak{A}}(a, b)$ the unique 2-type *realized* by the pair a, b, i.e. the 2-type β such that $\mathfrak{A} \models \beta[a, b]$. In general, for finite $B, C \subseteq A$, $B \cap C = \emptyset$, by $tp^{\mathfrak{A}}(B, C)$ we denote the similarity type of the substructure $\mathfrak{A} \restriction (B \cup C)$ (or, in other words, its $card(B \cup C)$-type).

Assume $\mathfrak{A}$ is a σ-structure and $B, C \subseteq A$. We denote by $\boldsymbol{\alpha}^{\mathfrak{A}}$ (respectively, $\boldsymbol{\alpha}^{\mathfrak{A}}[B]$) the set of all 1-types realized in $\mathfrak{A}$ (respectively, realized in $\mathfrak{A} \restriction B$), and by $\boldsymbol{\beta}^{\mathfrak{A}}$ (respectively, $\boldsymbol{\beta}^{\mathfrak{A}}[B]$) the set of all 2-types realized in $\mathfrak{A}$ (respectively, realized in $\mathfrak{A} \restriction B$). We denote by $\boldsymbol{\beta}^{\mathfrak{A}}[a, B]$ the set of all 2-types $tp^{\mathfrak{A}}(a, b)$ with $b \in B$, and by $\boldsymbol{\beta}^{\mathfrak{A}}[B, C]$ the set of all 2-types $tp^{\mathfrak{A}}(b, c)$ with $b \in B, c \in C$.

Let γ be a σ-sentence of the form $\forall x \exists y\, \psi(x, y)$ and $a \in A$. We say that an element $b \in A$ is a *γ-witness* for a in the structure $\mathfrak{A}$ if $\mathfrak{A} \models \psi(a, b)$; b is a *proper γ-witness*, if b is a γ-witness and $a \neq b$.

Scott normal form. As with FO^2, so too with FO^2_T, analysis is facilitated by the availability of normal forms.

▶ **Definition 1.** An FO^2-sentence Ψ is in *Scott normal form* if it is of the following form: $\forall x \forall y\, \psi_0(x, y) \wedge \bigwedge_{i=1}^{M} \forall x \exists y\, \psi_i(x, y)$, where every ψ_i is quantifier-free and includes unary and binary predicate letters only.

Without loss of generality we suppose that for $i \geq 1$, $\psi_i(x, y)$ entails $x \neq y$ (replacing $\psi_i(x, y)$ with $(\psi_i(x, y) \vee \psi_i(x, x)) \wedge x \neq y$, which is sound over all structures with at least two elements).

Two formulas are said to be *strongly equisatisfiable* if they are satisfiable over the same universe. The following Lemma is typical for two-variable logics.

▶ **Lemma 2** ([25, 5]). *For every formula $\varphi \in \mathrm{FO}^2$ one can compute in polynomial time a strongly equisatisfiable normal form formula $\psi \in \mathrm{FO}^2$ over a new signature whose length is linear in the length of φ.*

Suppose the signature σ consists of predicates of arity at most 2. To define a σ-structure $\mathfrak{A}$, it suffices to specify the 1-types and 2-types realized by elements and pairs of elements from the universe A. In the presence of a transitive relation, we classify 2-types according to the transitive connection between x and y. And so, we distinguish $\boldsymbol{\beta}^{\rightarrow}$, $\boldsymbol{\beta}^{\leftarrow}$, $\boldsymbol{\beta}^{\leftrightarrow}$ and $\boldsymbol{\beta}^{-}$ such that $\boldsymbol{\beta} = \boldsymbol{\beta}^{\rightarrow} \dot{\cup} \boldsymbol{\beta}^{\leftarrow} \dot{\cup} \boldsymbol{\beta}^{\leftrightarrow} \dot{\cup} \boldsymbol{\beta}^{-}$ and for instance: $\beta \in \boldsymbol{\beta}^{\rightarrow}$ iff $(T(x,y) \wedge \neg T(y,x)) \in \beta$, $\beta \in \boldsymbol{\beta}^{\leftrightarrow}$ iff $(T(x,y) \wedge T(y,x)) \in \beta$, etc.

For a quantifier-free formula $\varphi(x,y)$ we use superscripts $^{\rightarrow}$, $^{\leftarrow}$, $^{\leftrightarrow}$ and $^{-}$ to define new formulas that explicitly specify the transitive connection between x and y. For instance, for a quantifier-free formula $\varphi(x,y) \in \mathrm{FO}^2_T$ we let $\varphi^{\rightarrow}(x,y) := \varphi(x,y) \wedge T(x,y) \wedge \neg T(y,x)$.

This conversion of FO^2_T-formulae leads to the the following variant of the Scott normal form:

$$\forall x \forall y\, \psi_0 \wedge \bigwedge_{i=1}^{m} \gamma_i \wedge \bigwedge_{i=1}^{\overline{m}} \delta_i \tag{1}$$

where $\gamma_i = \forall x \exists y\, \psi_i^{d_i}(x,y)$ with $d_i \in \{^{\rightarrow}, ^{\leftarrow}, ^{-}\}$, and $\delta_i = \forall x \exists y\, \psi_i^{\leftrightarrow}(x,y)$.

For a fixed sentence Ψ in normal form (1) we often write $\gamma_i \in \Psi$ to indicate that γ_i is a conjunct of Ψ of the form $\forall x \exists y\, \psi_i^{d_i}(x,y)$.

▶ **Lemma 3.** *Let φ be an FO^2_T-formula over a signature τ. We can compute, in polynomial time, a strongly equisatisfiable FO^2_T-formula in normal form, over a signature σ consisting of τ together with a number of additional unary and binary predicates.*

Sketch. We employ the standard technique of renaming subformulas familiar from [25] and [5], noting that any formula $\exists y \psi$ is logically equivalent to $\exists y \psi^{\rightarrow} \vee \exists y \psi^{\leftarrow} \vee \exists y \psi^{\leftrightarrow} \vee \exists y \psi^{-}$. ◀

The following trivial observation will be very useful in the paper.

▶ **Proposition 4.** *Assume $\mathfrak{A}$ is a σ-structure and Ψ is a FO^2_T-sentence over σ in normal form (1). Then $\mathfrak{A} \models \Psi$ if and only if*

(a) for every $a \in A$, for every γ_i $(1 \leq i \leq m)$ there is a γ_i-witness for a in $\mathfrak{A}$,
(b) for every $a \in A$, for every δ_i $(1 \leq i \leq \overline{m})$ there is a δ_i-witness for a in $\mathfrak{A}$,
(c) for every $a, b \in A$, $tp^{\mathfrak{A}}(a,b) \models \psi_0$,
(d) $T^{\mathfrak{A}}$ is transitive in $\mathfrak{A}$.

A small clique property for FO^2_T. Let $\mathfrak{A}$ be a σ-structure. A subset B of A is called *T-connected* if $\boldsymbol{\beta}[B] \subseteq \boldsymbol{\beta}^{\leftrightarrow}[\mathfrak{A}]$. Maximal T-connected subsets of A are called *cliques*. Note that if $\boldsymbol{\beta}[a, \mathfrak{A}] \cap \boldsymbol{\beta}^{\leftrightarrow}[A] = \emptyset$, for some $a \in A$, then $\{a\}$ is a clique. We prove the following *small clique property*.

▶ **Lemma 5.** *Let Ψ be a satisfiable FO^2_T-sentence in normal form, over a signature σ. Then there exists a model of Ψ in which the size of each clique is bounded exponentially in $|\sigma|$.*

We first show how to replace a single clique in models of normal-form FO^2_T-sentences by an equivalent small one. The idea is not new, it was used in [27] to show that T-cliques in models of GF^2+TG can be replaced by appropriate small structures called T-petals (Lemma 17). Later, in [16] it was proved that for any structure $\mathfrak{A}$ and its substructure $\mathfrak{B}$, one may replace $\mathfrak{B}$ by an alternative structure $\mathfrak{B}'$ of a bounded size in such a way that the obtained structure $\mathfrak{A}'$ and the original structure $\mathfrak{A}$ satisfy exactly the same normal form FO^2 formulas. Due to space limitations, a precise statement of the latter lemma and the proof of the small clique model property will appear in the full version of the paper.

3 Splices and duplicability

In the remainder of the paper we fix a relational signature σ and assume Ψ is an FO^2_T-sentence in normal form (1). By Lemma 5, we may already assume (and we do so) that models of Ψ have the small clique property. In the next two sections we assume that Ψ is satisfiable and, if not stated otherwise, $\mathfrak{A} \models \Psi$.

In this section we analyze properties of models of Ψ on the level of cliques rather than individual elements. We give here the key technical argument of the paper (Corollary 11). It says, roughly speaking, that if $\mathfrak{A} \models \Psi$ and elements of a finite subset F of the universe A have their γ_i-witnesses in several "similar" cliques then $\mathfrak{A}$ can be extended by one new clique, where all the elements of F have their γ_i-witnesses.

First, we need to introduce some new notions and notation. For $a \in A$ denote by $Cl^{\mathfrak{A}}(a)$ the unique clique $C \subseteq A$ with $a \in C$. When $F \subseteq A$, denote $Cl^{\mathfrak{A}}(F) = \{Cl^{\mathfrak{A}}(a) : a \in F\}$ and finally, $Cl^{\mathfrak{A}} = Cl^{\mathfrak{A}}(A)$. Note that whenever $B \in Cl^{\mathfrak{A}}$, and $a \in A$ is an element outside the clique B, then the 2-types between a and any element $b \in B$ belong to the same subset of $\boldsymbol{\beta}$, i.e. either to $\boldsymbol{\beta}^{\rightarrow}$, $\boldsymbol{\beta}^{\leftarrow}$ or $\boldsymbol{\beta}^{-}$. So, we might speak about elements of $\mathfrak{A}$ connected with the clique B using "directed" edges. Similarly, we can identify cliques connected with B using "incoming" and "outgoing" edges.

▶ **Definition 6.** Let $\mathfrak{A}$ be a σ-structure and let $B \in Cl^{\mathfrak{A}}$. Define:
- $In^{\mathfrak{A}}(B) \stackrel{def}{=} \{tp^{\mathfrak{A}}(C) : C \in Cl^{\mathfrak{A}}$ and $\boldsymbol{\beta}[C, B] \subseteq \boldsymbol{\beta}^{\rightarrow}\}$,
- $Out^{\mathfrak{A}}(B) \stackrel{def}{=} \{tp^{\mathfrak{A}}(C) : C \in Cl^{\mathfrak{A}}$ and $\boldsymbol{\beta}[C, B] \subseteq \boldsymbol{\beta}^{\leftarrow}\}$,
- $sp^{\mathfrak{A}}(B) = \langle tp^{\mathfrak{A}}(B), In^{\mathfrak{A}}(B), Out^{\mathfrak{A}}(B)\rangle$,
- $Sp^{\mathfrak{A}} = \{sp^{\mathfrak{A}}(B) : B \in Cl^{\mathfrak{A}}\}$. Elements of $Sp^{\mathfrak{A}}$ are called $\mathfrak{A}$-*splices.*

Splices define cliques reachable from a given clique via T.

We say that two cliques $B, B' \in Cl^{\mathfrak{A}}$ *realize the same splice*, written $B \equiv^{\mathfrak{A}} B'$, if $sp^{\mathfrak{A}}(B) = sp^{\mathfrak{A}}(B')$. When $\mathfrak{A}$ is understood we often omit the superscript in $\equiv^{\mathfrak{A}}$ and write $\equiv$. Note that $\equiv^{\mathfrak{A}}$ is an equivalence relation on $Cl^{\mathfrak{A}}$. Moreover, if we have an a priori upper bound on the size of cliques in $Cl^{\mathfrak{A}}$, then $Cl^{\mathfrak{A}}/_{\equiv}$ is finite (and of bounded cardinality).

Additionally, we distinguish the set $\mathbb{K}(\mathfrak{A})$ of *unique cliques* in $\mathfrak{A}$: $\mathbb{K}(\mathfrak{A}) = \{B \in Cl^{\mathfrak{A}} : card([B]_{\equiv}) = 1\}$ and the corresponding subset $K(\mathfrak{A})$ of the universe of $\mathfrak{A}$, that consists of the elements of the unique cliques: $K(\mathfrak{A}) = \bigcup_{B \in \mathbb{K}(\mathfrak{A})} B$.

Our task is now to show that any model of Ψ containing a non-unique clique B can be extended into a new model of Ψ by adding a copy of B. The copy of B is added in such a way that it also provides, for all conjuncts of the form γ_i, all the witnesses for elements outside the two cliques that have been provided by B. This property will be explored later, when new models will be constructed by removing *segments* (i.e. sets of cliques) from given ones.

For every $\gamma_i \in \Psi$ (recall $\gamma_i = \forall x \exists y\, \psi_i^{d_i}(x, y)$ with $d_i \in \{^{\rightarrow}, ^{\leftarrow}, ^{-}\}$) and for every $a \in A$ we define $W_i^{\mathfrak{A}}(a)$ as the set of all proper γ_i-witnesses for a in $\mathfrak{A}$:

$$W_i^{\mathfrak{A}}(a) \stackrel{def}{=} \{b \in A : \mathfrak{A} \models \psi_i(a, b),\, b \neq a\}.$$

Similarly, for every $\gamma_i \in \Psi$ and for every $F \subseteq A$ we define $W_i^{\mathfrak{A}}(F) \stackrel{def}{=} \bigcup_{a \in F} W_i^{\mathfrak{A}}(a)$.

The following claim states that every non-unique clique in a given model can be properly duplicated, as informally described above.

▶ **Claim 7** (Duplicability). *Assume $\mathfrak{A} \models \Psi$, $B_1 \in Cl^{\mathfrak{A}}$ and $B_1 \notin \mathbb{K}(\mathfrak{A})$. There is an extension $\mathfrak{A}'$ of $\mathfrak{A}$ by one new clique D such that*
1. $\mathfrak{A}' \models \Psi$,

2. *for every conjunct γ_i of Ψ, for every $a \in A$ we have:*
$$B_1 \cap W_i^{\mathfrak{A}}(a) \neq \emptyset \quad \textit{iff} \quad D \cap W_i^{\mathfrak{A}'}(a) \neq \emptyset, \textit{ and}$$
3. $sp^{\mathfrak{A}'}(D) = sp^{\mathfrak{A}'}(B_1) = sp^{\mathfrak{A}}(B_1)$.

Proof. Let $\mathfrak{A} \models \Psi$, $B_1 \in Cl^{\mathfrak{A}}$. Since $B_1 \notin \mathbb{K}(\mathfrak{A})$, there exists $B_2 \in Cl^{\mathfrak{A}}$, $B_2 \neq B_1$ such that $sp^{\mathfrak{A}}(B_2) = sp^{\mathfrak{A}}(B_1)$. Assume $\mathfrak{D}$ is a duplicate of $\mathfrak{A} \restriction B_1$, $D \cap A = \emptyset$, $f_1 : D \mapsto B_1$ and $f_2 : D \mapsto B_2$ are appropriate isomorphism functions. Let $\mathfrak{A}'$ be an extension of $\mathfrak{A}$ with the universe $A \dot{\cup} D$ such that:

- $tp^{\mathfrak{A}'}(d,b) \overset{def}{=} tp^{\mathfrak{A}}(b, f_2(d))$, for every $b \in B_1$, $d \in D$ (note that $\boldsymbol{\beta}[d, B_1] = \boldsymbol{\beta}[B_1, f_2(d)]$ and so $\boldsymbol{\beta}[D, B_1] = \boldsymbol{\beta}[B_1, B_2]$),
- $tp^{\mathfrak{A}'}(d,a) \overset{def}{=} tp^{\mathfrak{A}}(f_1(d), a)$, for every $a \in A \setminus (B_1 \cup D)$, $d \in D$ (note that $\boldsymbol{\beta}[d, A \setminus B_1] = \boldsymbol{\beta}[f_1(d), A \setminus B_1]$ and so $\boldsymbol{\beta}[D, A \setminus B_1] = \boldsymbol{\beta}[B_1, A \setminus B_1]$).

To see that $\mathfrak{A}' \models \Psi$ one can show that conditions (a)–(d) of Proposition 4 hold for $\mathfrak{A}'$. ◀

Using the above claim we may build *saturated models* in the following sense.

▶ **Definition 8.** Assume $\mathfrak{A} \models \Psi$. We say that $\mathfrak{A}$ is *witness-saturated*, if $\mathfrak{A}$ has the small clique property and for every $a \in A$, for every $\gamma_i \in \Psi$ $(1 \leq i \leq m)$

$$W_i(a) \subseteq K(\mathfrak{A}) \quad \text{or} \quad W_i(a) \text{ is infinite.}$$

Note that if a witness-saturated model $\mathfrak{A}$ is finite then $A = K(\mathfrak{A})$. By Lemma 5 and by iterative application of Claim 7 we get the following.

▶ **Lemma 9** (Saturated model). *Every satisfiable normal form sentence Ψ has a countable witness-saturated model. Additionally, if $\mathfrak{A}$ is witness-saturated, then the extension $\mathfrak{A}'$ given by Claim 7 is also witness-saturated.*

The above Lemma is essential for the proof of the key technical tool for the paper, Corollary 11, given below. It says that when several elements $a_1, a_2, \ldots, a_n$ of a model $\mathfrak{A}$ have γ_i-witnesses in several distinguished cliques that realize the same splice, one can extend the model $\mathfrak{A}$ by a single clique D (realizing the same splice) in which $a_1, a_2, \ldots, a_n$ have their γ_i-witnesses. The proof is based on a more subtle (than in Claim 7) analysis of models of a normal form sentence Ψ given in Claim 10. In the Claim note that whenever $\boldsymbol{\beta}(C_1, C_2) \in \boldsymbol{\beta}^{\leftarrow}$ then $\boldsymbol{\beta}(C_2, C_1) \notin \boldsymbol{\beta}^{\leftarrow}$.

▶ **Claim 10.** *Assume $\mathfrak{A}$ is countable witness-saturated and $\gamma_i \in \Psi$. Let $C_1, C_2, B_1, B_2 \in Cl^{\mathfrak{A}}$, $\boldsymbol{\beta}(C_1, C_2) \notin \boldsymbol{\beta}^{\leftarrow}$, $B_1 \neq B_2$, $B_1 \cap W_i(C_1) \neq \emptyset$, $B_2 \cap W_i(C_2) \neq \emptyset$, $C_1 \notin \mathbb{K}(\mathfrak{A})$, $C_1 \equiv C_2$ and $B_1 \equiv B_2$. Then, there exists an extension $\mathfrak{A}_1$ of $\mathfrak{A}$ by at least one clique D such that*
(i) $\mathfrak{A}_1 \models \Psi$ *and* $\mathfrak{A}_1$ *is witness-saturated,*
(ii) *for every $a \in A$: if $B_1 \cap W_i^{\mathfrak{A}}(a) \neq \emptyset$ then $D \cap W_i^{\mathfrak{A}_1}(a) \neq \emptyset$,*
(iii) *for every $a \in C_2$: if $B_2 \cap W_i^{\mathfrak{A}}(a) \neq \emptyset$ then $D \cap W_i^{\mathfrak{A}_1}(a) \neq \emptyset$,*
(iv) $sp^{\mathfrak{A}_1}(D) = sp^{\mathfrak{A}_1}(B_1) = sp^{\mathfrak{A}}(B_1)$.

Proof. We have several cases. In each case we add a duplicate D of the clique B_1 where both C_1 and C_2 will get their γ_i-witnesses. We sketch only one of the interesting cases.
Case 1. $\boldsymbol{\beta}[C_1, B_1] \subseteq \boldsymbol{\beta}^{-}$ and $\boldsymbol{\beta}[C_1, C_2] \subseteq \boldsymbol{\beta}^{\rightarrow} \cup \boldsymbol{\beta}^{-}$. In this case $C_1 \neq C_2$ and $\gamma_i = \forall x \exists y\, \psi_i^{-}(x,y)$. So, $\boldsymbol{\beta}[C_1, B_1] \subseteq \boldsymbol{\beta}^{-}$ and $\boldsymbol{\beta}[C_2, B_2] \subseteq \boldsymbol{\beta}^{-}$.
Subcase 1.a. $\boldsymbol{\beta}[C_1, C_2] \in \boldsymbol{\beta}^{\rightarrow}$, $\boldsymbol{\beta}[B_1, B_2] \subseteq \boldsymbol{\beta}^{-}$, $\boldsymbol{\beta}[C_1, B_2] \subseteq \boldsymbol{\beta}^{\rightarrow}$ and $\boldsymbol{\beta}[B_1, C_2] \in \boldsymbol{\beta}^{\rightarrow}$.

The construction proceeds in four steps.
Step 1 (copying of B_1). Let $\mathfrak{A}'$ be the witness saturated extension of $\mathfrak{A}$ by one new clique D – a duplicate of the clique B_1 given by Claim 7 and Lemma 9. Observe that all conditions *(i)–(iv)* hold in $\mathfrak{A}'$ except *(iii)* since by construction of $\mathfrak{A}'$ $\boldsymbol{\beta}^{\mathfrak{A}'}[C_2, D] = \boldsymbol{\beta}^{\mathfrak{A}}[C_2, B_1] \subseteq \boldsymbol{\beta}^{\leftarrow}$.

Step 2 (modification of $tp^{\mathfrak{A}'}(C_2, D)$). To ensure that *(iii)* holds, a new structure $\mathfrak{A}_2$ is built by defining $tp^{\mathfrak{A}_2}(C_2, D) \stackrel{def}{=} tp^{\mathfrak{A}'}(C_2, B_2)$.
Step 3 (transitivity correction). To ensure that T is transitive we construct a structure $\mathfrak{A}_3$: for every $X \in Cl^{\mathfrak{A}_2}$, if $\boldsymbol{\beta}[D, X] \subseteq \boldsymbol{\beta}^{\rightarrow}$ and $\boldsymbol{\beta}[X, C_2] \subseteq \boldsymbol{\beta}^{\rightarrow}$ then replace $tp^{\mathfrak{A}_2}(D, X)$ by $tp^{\mathfrak{A}_2}(B_2, X)$. One can observe that $\boldsymbol{\beta}[X, B_2] \subseteq \boldsymbol{\beta}^{-}$, so T is transitive in $\mathfrak{A}_3$.
Step 4 (γ_j-witness in X correction). Let $X \in Cl^{\mathfrak{A}_2}$ be such that the type $tp^{\mathfrak{A}_2}(D, X)$ is changed in Step 3. We show that then there is a clique $X_2 \in Cl^{\mathfrak{A}_3}$ such that $tp^{\mathfrak{A}_3}(X_2) = tp^{\mathfrak{A}_3}(X)$ and $\boldsymbol{\beta}^{\mathfrak{A}_3}[D, X_2] \subseteq \boldsymbol{\beta}^{\rightarrow}$. For, observe that $tp^{\mathfrak{A}_3}(X) \in Out^{\mathfrak{A}_3}(B_1)$ and so $tp^{\mathfrak{A}_3}(X) \in Out^{\mathfrak{A}_3}(B_2)$ since $B_1 \equiv B_2$. Let $X_1 \in Cl^{\mathfrak{A}_3}$, $tp^{\mathfrak{A}_3}(X_1) = tp^{\mathfrak{A}_3}(X)$ and $\boldsymbol{\beta}[B_2, X_1] \subseteq \boldsymbol{\beta}^{\rightarrow}$. Since $\boldsymbol{\beta}[C_1, B_2] \subseteq \boldsymbol{\beta}^{\rightarrow}$ we have $\boldsymbol{\beta}[C_1, X_1] \subseteq \boldsymbol{\beta}^{\rightarrow}$ and so $tp^{\mathfrak{A}_3}(X_1) \in Out^{\mathfrak{A}_3}(C_1)$. Hence, since $C_1 \equiv C_2$ there exist $X_2 \in Cl^{\mathfrak{A}_3}$ such that $tp^{\mathfrak{A}_3}(X_2) = tp^{\mathfrak{A}_3}(X_1)$ and $\boldsymbol{\beta}[C_2, X_2] \subseteq \boldsymbol{\beta}^{\rightarrow}$. Since $\boldsymbol{\beta}[B_1, C_2] \subseteq \boldsymbol{\beta}^{\rightarrow}$, so $\boldsymbol{\beta}[B_1, X_2] \subseteq \boldsymbol{\beta}^{\rightarrow}$, and hence, by construction of $\mathfrak{A}'$, $\boldsymbol{\beta}[D, X_2] \subseteq \boldsymbol{\beta}^{\rightarrow}$. To obtain the required model $\mathfrak{A}_1$ replace $tp^{\mathfrak{A}_3}(D, X_2)$ by $tp^{\mathfrak{A}_2}(D, X)$ $(= tp^{\mathfrak{A}}(B_1, X))$.

Correctness proof of the above construction, as well as other cases, will appear in the full version of the paper. ◀

▶ **Corollary 11.** *Assume $\mathfrak{A}$ is countable witness-saturated, $\gamma_i \in \Psi$ and $X \in Sp^{\mathfrak{A}}$. Let $F \subseteq A \setminus K(\mathfrak{A})$ be a finite set such that for every $a \in F$ there is $b \in W_i^{\mathfrak{A}}(a)$ such that $b \notin K(\mathfrak{A})$ and $sp(Cl^{\mathfrak{A}}(b)) = X$. Then, there exists an extension $\mathfrak{A}'$ of $\mathfrak{A}$ by at least one clique D such that*

(i) $\mathfrak{A}' \models \Psi$ *and* $\mathfrak{A}'$ *is witness-saturated,*
(ii) $D \cap W_i^{\mathfrak{A}'}(a) \neq \emptyset$, *for every* $a \in F$,
(iii) $sp^{\mathfrak{A}'}(D) = X$.

Proof. Let $F = \{a_1, a_2, \ldots, a_p\}$, where a_1 denotes an element of F such that for every $a \in F \setminus \{a_1\}$, $tp^{\mathfrak{A}}(a_1, a) \notin \boldsymbol{\beta}^{\leftarrow}$. (Note that a_1 can always be found since F is finite and T is transitive in $\mathfrak{A}$.) We iteratively apply Claim 10. Denote $\mathfrak{A}^1 = \mathfrak{A}$ and for $k = 2, 3, \ldots, p$ let $\mathfrak{A}^k$ and D^k be the extension of $\mathfrak{A}_1^{k-1}$ by at least one clique D^k given by Claim 10 for a_1 and a_k. Obviously, for every k $(2 \leq k \leq p)$ we have $D^k \cap W_i^{\mathfrak{A}'}(a) \neq \emptyset$, for every $a \in \{a_1, a_2, \ldots, a_k\}$. ◀

4 Canonical models

In this section we analyze properties of models of Ψ on the level of segments which consist of several cliques, and constitute a partition $S_0, S_1, \ldots$ of the universe of a model. Every segment S_j has a fixed (doubly exponential) size and is meant to contain all γ_i-witnesses for elements from earlier segments $S_0, S_1, \ldots, S_{j-1}$. On this level of abstraction cliques and splices of a model become less important.

▶ **Definition 12.** A finite subset $S \subset A$ is a *segment* in $\mathfrak{A}$ if $Cl^{\mathfrak{A}}(a) \subseteq S$ for every $a \in S$.

In the following we reserve the letter S (possibly decorated) to denote segments. Define $s = |Sp(\sigma)|$ and denote by h the bound of the size of each clique in a small-clique σ-structure, given by Lemma 5. Note that s is doubly exponential and h is exponential in $|\sigma|$.

We first prove a generalization of Corollary 11. It says, roughly speaking, that if $\mathfrak{A} \models \Psi$ and F is a finite subset of A, then it is possible to extend $\mathfrak{A}$ by a segment of fixed cardinality in which all elements of F have their γ_i-witnesses, for every i $(1 \leq i \leq m)$.

▶ **Lemma 13** (Witnesses compression)**.** *Assume $\mathfrak{A}$ is a countable witness-saturated model of Ψ and $F \subseteq A \setminus K(\mathfrak{A})$ is finite. There is a witness-saturated extension $\mathfrak{A}'$ of $\mathfrak{A}$ by a segment S such that*

1. $\mathfrak{A}' \models \Psi$,
2. $|S| \leq m \cdot s \cdot h$,
3. *for every* $\gamma_i \in \Psi$, *for every* $a \in F$, *if* $W_i^{\mathfrak{A}}(a) \setminus K(\mathfrak{A}) \neq \emptyset$, *then* $W_i^{\mathfrak{A}'}(a) \cap S \neq \emptyset$.

Proof. First, for given i $(1 \leq i \leq m)$ and $X \in Sp^{\mathfrak{A}}$ let $F_i^X \subseteq S$ be a maximal subset of F such that for every $a \in F_i^X$ there is $b \in W_i^{\mathfrak{A}}(a)$ such that $b \notin K(\mathfrak{A})$ and $sp(Cl^{\mathfrak{A}}(b)) = X$. Now, for every i $(1 \leq i \leq m)$ and for every $X \in Sp^{\mathfrak{A}}$ iteratively apply Corollary 11 for F_i^X and denote each new clique added in the process by D_i^X. Let S be the segment consisting of elements of the new cliques: $S \stackrel{def}{=} \bigcup_{1 \leq i \leq m} \bigcup_{X \in Sp^{\mathfrak{A}}} D_i^X$.

Condition (i) of Lemma 11 implies that $\mathfrak{A}' \models \Psi$. Obviously, $|S| \leq m \cdot s \cdot h$. To show that condition 3 of our lemma holds, assume $\gamma_i \in \Psi$, $a \in F$ and there exists $b \in W_i^{\mathfrak{A}}(a)$ such that $b \notin K(\mathfrak{A})$. So $a \in F_i^X$, where $X = sp(Cl^{\mathfrak{A}}(b))$. By condition (ii) of Lemma 11 we obtain $D_i^X \cap W_i^{\mathfrak{A}'}(a) \neq \emptyset$, and so, $W_i^{\mathfrak{A}'}(a) \cap S \neq \emptyset$. ◀

▶ **Definition 14.** A segment $S \subsetneq A$ is *redundant in* $\mathfrak{A}$, if for every $a \in A \setminus S$ and for every $\gamma_i \in \Psi$ we have: $W_i^{\mathfrak{A}}(a) \cap S \neq \emptyset$ implies there exists $c \in A \setminus S$ such that $c \in W_i^{\mathfrak{A}}(a)$.

▶ **Claim 15.** *If* $\mathfrak{A} \models \Psi$ *and* $S \subsetneq A$ *is redundant in* $\mathfrak{A}$, *then* $\mathfrak{A} \restriction (A \setminus S) \models \Psi$.

Proof. Every subgraph of a transitive graph is also transitive. Conditions (a)–(c) of Proposition 4 obviously hold for $\mathfrak{A} \restriction (A \setminus S)$. ◀

▶ **Definition 16.** A model $\mathfrak{A}$ of Ψ is *narrow* if there is an infinite partition $P_A = \{S_0, S_1, \ldots\}$ of the universe A such that:
1. $K(\mathfrak{A}) \subset S_0$, $|S_0| \leq (m+1) \cdot s \cdot h$,
2. $|S_j| \leq m \cdot s \cdot h$, for every $j \geq 1$,
3. for every $j \geq 0$, for every $e \in \bigcup_{k=0}^{j} S_k$ and for every $\gamma_i \in \Psi$,
if $W_i^{\mathfrak{A}}(e) \cap S_0 = \emptyset$, then $W_i^{\mathfrak{A}}(e) \cap S_{j+1} \neq \emptyset$.

▶ **Lemma 17.** *Every infinitely satisfiable sentence* Ψ *has a narrow model.*

Proof. Assume $\mathfrak{A}$ is an infinite witness-saturated model of Ψ that exists by Lemma 13. For $\gamma_i \in \Psi$ and $a \in A$ denote by $\bar{\gamma}_i(a)$ an arbitrarily chosen element $b \in W_i^{\mathfrak{A}}(a)$. Define $\mathfrak{A}_0 = \mathfrak{A}$ and $S_0 = K(\mathfrak{A}) \cup \bigcup_{a \in K(\mathfrak{A})} Cl^{\mathfrak{A}}(\bar{\gamma}_i(a))$.
Now, for $j = 0, 1, 2, \ldots$ define:
- $\mathfrak{A}_{j+1} = \mathfrak{A}'_j$, where $\mathfrak{A}'_j$ is the extension of $\mathfrak{A}_j$ given by Lemma 13 for $F = \bigcup_{k=0}^{j} S_k$,
- $S_{j+1} = B$, where B is the finite set given by Lemma 13; B extends $\mathfrak{A}_j$ to $\mathfrak{A}_{j+1}$ in such a way, that:
  - $\mathfrak{A}_{j+1} \models \Psi$,
  - $|B| \leq m \cdot s \cdot h$,
  - for every $\gamma_i \in \Psi$, for every $a \in F$, if $W_i^{\mathfrak{A}}(a) \setminus K(\mathfrak{A}_j) \neq \emptyset$, then $W_i^{\mathfrak{A}_{j+1}}(a) \cap B \neq \emptyset$.

Now, define $\mathfrak{A}' = (\bigcup_{k=0}^{\infty} \mathfrak{A}_k) \restriction \bigcup_{k=0}^{\infty} S_k$. By Claim 15 and Lemma 13, it is easy to see, that $\mathfrak{A}'$ is a narrow model of Ψ with partition $P_{A'} = \{S_0, S_1, \ldots\}$. ◀

▶ **Definition 18.** Assume $\mathfrak{A}$ is a σ-structure, $x, y \in \mathbb{N}^+$ and B, B', C, C' are finite subsets of A with fixed orderings: $B = \{b_1 < \ldots < b_x\}$, $B' = \{b'_1 < \ldots < b'_y\}$, $C = \{c_1 < \ldots < c_x\}$, $C' = \{c'_1 < \ldots, c'_y\}$ such that $B \cap B' = \emptyset$, $C \cap C' = \emptyset$.

A *connection type* of B and B' in $\mathfrak{A}$ is the structure $\langle B, B' \rangle \stackrel{def}{=} \mathfrak{A} \restriction (B \cup B')$. Two connection types $\langle B, B' \rangle$ and $\langle C, C' \rangle$ are *the same connection types in* $\mathfrak{A}$, denoted $\langle B, B' \rangle \cong^{\mathfrak{A}} \langle C, C' \rangle$, if the function $f : B \cup B' \mapsto C \cup C'$ defined by $f(b_j) = c_j$ and $f(b'_j) = c'_j$ is an isomorphism of $\langle B, B' \rangle$ and $\langle C, C' \rangle$.

▶ **Definition 19.** Assume $\mathfrak{A}$ is a narrow model of Ψ and $P_A = \{S_0, S_1, \ldots\}$ is any partition satisfying conditions 1-3 of Definition 16. We say that $\mathfrak{A}$ is *canonical* if for every $j, k \in \mathbb{N}^+$, $0 < j < k$, we have $\langle S_j, S_0\rangle \cong^{\mathfrak{A}} \langle S_1, S_0\rangle$, and $\langle S_k, S_j\rangle \cong^{\mathfrak{A}} \langle S_2, S_1\rangle$.

▶ **Lemma 20.** *Every infinitely satisfiable sentence Ψ has a canonical model.*

Proof. Let $\mathfrak{A}$ be a narrow model of Ψ with partition $P_A = \{S_0, S_1, \ldots\}$ given by Definition 16. Additionally, assume that in every segment S_j, $j \geq 0$, there is a fixed linear ordering.

Observe first that for every $p > 0$, S_p is redundant in $\mathfrak{A}$. For, assume (cf. Definition 14) $b \in S_p$, $a \in A \setminus S_p$ and $b \in W_i^{\mathfrak{A}}(a)$. Assume $a \in S_q$ and take $j \in \mathbb{N}^+$ such that $j > \max\{p, q\}$. By Definition 16, we have that if $W_i^{\mathfrak{A}}(a) \cap S_0 = \emptyset$, then $W_i^{\mathfrak{A}}(a) \cap S_{j+1} \neq \emptyset$. So, there is $c \in S_0 \cup S_{j+1}$ such that $c \in W_i^{\mathfrak{A}}(a)$. Similarly, for every infinite $V \subset \mathbb{N}^+$, the segment $\bigcup_{j \in \mathbb{N}^+ \setminus V} S_j$ is redundant in $\mathfrak{A}$ and, by Claim 15, $\mathfrak{A} \restriction \bigcup_{j \in V \cup \{0\}} S_j \models \Psi$.

To construct the canonical model we first find an infinite set $V \subset \mathbb{N}^+$ such that for every $j,\ l \in V$, $j \neq l$, $\langle S_l, S_0\rangle \cong^{\mathfrak{A}} \langle S_j, S_0\rangle$. Observe that the set V does exist (by the infinite Ramsey theorem, there are infinitely many segments in P_A and only a finite number of similarity types).

Secondly, we find an infinite set $W \subseteq V$ such that for every $i, j, k, l \in W$ with $i < j$ and $k < l$ we have $\langle S_j, S_i\rangle \cong^{\mathfrak{A}} \langle S_l, S_k\rangle$. (Again, W exists by the infinite Ramsey theorem).

Finally, we define $\mathfrak{A}' = \mathfrak{A} \restriction \bigcup_{j \in W \cup \{0\}} S_j$. By Claim 15, $\mathfrak{A}' \models \Psi$. Obviously, $\mathfrak{A}'$ is canonical with partition $P_{A'} = \{S_j : j \in \mathbb{N}^+, j = \#p\}$, where $\#p$ is the position number of $p \in W \cup \{0\}$. ◀

5 Decidability and complexity

From Lemma 20 we get immediately the following theorem.

▶ **Theorem 21.** *An FO^2_T-sentence Ψ is satisfiable if and only if there exist a σ-structure $\mathfrak{A}$ and $S_0, S_1, S_2, S_3 \subseteq A$, such that:*

1. $|A| \leq (4m + 1) \cdot s \cdot h$,
2. *either $S_1 = S_2 = S_3 = \emptyset$, or $\{S_0, S_1, S_2, S_3\}$ is a partition of A and then*
   - **a.** $\langle S_1, S_0\rangle \cong^{\mathfrak{A}} \langle S_2, S_0\rangle \cong^{\mathfrak{A}} \langle S_3, S_0\rangle$,
   - **b.** $\langle S_2, S_1\rangle \cong^{\mathfrak{A}} \langle S_3, S_2\rangle \cong^{\mathfrak{A}} \langle S_3, S_1\rangle$,
3. *for every $a, b \in A$, $tp^{\mathfrak{A}}(a, b) \models \psi_0$,*
4. *$T^{\mathfrak{A}}$ is transitive in $\mathfrak{A}$,*
5. *for every $j = 0, 1, 2$, for every $e \in S_j$ and for every $\gamma_i \in \Psi$,*
   if $W_i^{\mathfrak{A}}(e) \cap S_0 = \emptyset$, then $W_i^{\mathfrak{A}}(e) \cap S_{j+1} \neq \emptyset$.

Proof. ($\Rightarrow$) There are two cases. Either Ψ has only finite models and then, by Lemma 9, Ψ has a witness-saturated model $\mathfrak{A}$ with $A = K(\mathfrak{A})$. In this case, we put $S_0 = A$ and $S_1 = S_2 = S_3 = \emptyset$. Or, Ψ has infinite models, and then, by Lemma 20, Ψ has a canonical model $\mathfrak{A}'$ with partition $P_{A'} = \{S_0, S_1, \ldots\}$. In this case, we define $\mathfrak{A} \overset{def}{=} \mathfrak{A}' \restriction (S_0 \dot\cup S_1 \dot\cup S_2 \dot\cup S_3)$. Note that in either case $|S_0| \leq s \cdot h + m \cdot s \cdot h = (m + 1) \cdot s \cdot h$.
($\Leftarrow$) Define a structure $\mathfrak{A}'$ such that $A' \overset{def}{=} S_0 \mathbin{\dot\cup} S_1 \mathbin{\dot\cup} S_2 \mathbin{\dot\cup} S_3 \mathbin{\dot\cup} \dot{\bigcup}_{j=4}^{\infty} S_j$ and, for every $j, k \in \mathbb{N}^+$ $(0 < j < k) : \langle S_j, S_0\rangle \cong^{\mathfrak{A}'} \langle S_1, S_0\rangle$ and $\langle S_k, S_j\rangle \cong^{\mathfrak{A}'} \langle S_2, S_1\rangle$. It is obvious that $\mathfrak{A}'$ satisfies conditions (a)–(c) of Proposition 4. To show that T is transitive in $\mathfrak{A}'$ it suffices to prove that for every j, k, l $(0 \leq j \leq k \leq l)$, $T^{\mathfrak{A}' \restriction (S_j \cup S_k \cup S_l)}$ is transitive in $\mathfrak{A}' \restriction (S_j \cup S_k \cup S_l)$. The latter condition can be easily verified; hence, $\mathfrak{A}' \models \Psi$. ◀

▶ **Corollary 22.** *SAT(FO^2_T) $\in$ 2-NExpTime.*

Proof. To check whether a given FO^2_Tsentence Ψ is satisfiable we follow Theorem 21 and we obtain a nondeterministic double exponential time procedure, as described below.

```
1. Guess a $\sigma$-structure $\mathfrak{A}$ of cardinality $|A| \leq (4m+1) \cdot s \cdot h$ and partition
   $P_A = \{S_0, S_1, S_2, S_3\}$;
2. Guess enumerations of every segment $S_0, S_1, S_2, S_3$;
3. If not:
   a. $\langle S_1, S_0\rangle \cong^{\mathfrak{A}} \langle S_2, S_0\rangle \cong^{\mathfrak{A}} \langle S_3, S_0\rangle$ and
   b. $\langle S_2, S_1\rangle \cong^{\mathfrak{A}} \langle S_3, S_2\rangle \cong^{\mathfrak{A}} \langle S_3, S_1\rangle$
   then reject;
4. For every $a, b \in A$, if $tp^{\mathfrak{A}}(a,b) \not\models \psi_0$ then reject;
5. For every $a, b, c \in A$, if not $(T^{\mathfrak{A}}(a,b) \wedge T^{\mathfrak{A}}(b,c) \Rightarrow T^{\mathfrak{A}}(a,c))$ then reject;
6. For every $j = 0,1,2$, for every $e \in S_j$, for every $\gamma_i \in \Psi$ such that $W_i^{\mathfrak{A}}(e) \cap S_0 = \emptyset$
   if $W_i^{\mathfrak{A}}(e) \cap S_{j+1} = \emptyset$ then reject;
7. Accept;
```
◀

6 Outlook

Since the finite model property fails for FO^2_T, an interesting question is whether the finite satisfiability problem is also decidable. Immediately from Lemma 17 we have the following observation.

▶ **Corollary 23.** *An FO^2_T-sentence Ψ is satisfiable if and only if*
Ψ has a model of cardinality $\leq s \cdot h$, or Ψ has an infinite model.

The $s \cdot h$ bound on the size of the finite model of Ψ depends on the number of different σ-splices and the size of cliques in a structure with the small clique property. Unfortunately, this observation does not suffice to answer the finite satisfiability problem, as in general, one can imagine that a finite model contains several realizations of the same splice. So, to the best of our knowledge, the finite satisfiability problem for FO^2_T remains open. We believe that the latter problem is decidable; however, we suppose that an essential extension of the above approach is required to get the proof.

We also note that the 2-NExpTime bound for the satisfiability problem leaves a gap in complexity, as the best lower bound coming from the two-variable guarded logic with transitive guards is 2-ExpTime [10]. We believe that the upper bound proved in our paper can be improved; however, as FO^2_T does not enjoy the tree-like property, standard techniques using alternating machines cannot be applied directly.

Acknowledgements The authors acknowledge insightful comments from the anonymous referees that helped to improve the presentation and simplify some proofs in Section 4.

References

1 H. Andréka, J. van Benthem, and I. Németi. Modal languages and bounded fragments of predicate logic. *J. Philos. Logic*, 27:217–274, 1998.

2 M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and xml reasoning. In *Proc. of PODS-06*, pages 10–19, New York, NY, USA, 2006. ACM.

3 M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on words with data. In *LICS-06*, pages 7–16, 2006.

4 C. David, L. Libkin, and Tony Tan. On the satisfiability of two-variable logic over data words. In Christian G. Fermüller and Andrei Voronkov, editors, *LPAR (Yogyakarta)*, volume 6397 of *Lecture Notes in Computer Science*, pages 248–262. Springer, 2010.

5 E. Grädel, P. Kolaitis, and M. Vardi. On the decision problem for two-variable first-order logic. *Bull. of Symb. Logic*, 3(1):53–69, 1997.

6 E. Grädel and M. Otto. On Logics with Two Variables. *Theoretical Computer Science*, 224:73–113, 1999.

7 E. Grädel, M. Otto, and E. Rosen. Undecidability results on two-variable logics. *Arch. Math. Log.*, 38(4-5):313–354, 1999.

8 N. Immerman, A. M. Rabinovich, T. W. Reps, S. Sagiv, and G. Yorsh. The boundary between decidability and undecidability for transitive-closure logics. In *CSL*, pages 160–174, 2004.

9 Y. Kazakov. *Saturation-based decision procedures for extensions of the guarded fragment.* PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 2006.

10 E. Kieroński. The two-variable guarded fragment with transitive guards is 2EXPTIME-Hard. In *Proc. of FOSSACS*, volume LNCS 2620, pages 299–312, 2003.

11 E. Kieroński. Results on the guarded fragment with equivalence or transitive relations. In *Computer Science Logic*, volume 3634, pages 309–324. Springer Verlag, 2005.

12 E. Kieroński. Decidability issues for two-variable logics with several linear orders. In *CSL-11*, pages 337–351, 2011.

13 E. Kieroński and J. Michaliszyn. Two-variable universal logic with transitive closure. In *CSL-12*, LIPIcs, pages 337–351, 2012.

14 E. Kieroński, J. Michaliszyn, I. Pratt-Hartmann, and L. Tendera. Two-variable first-order logic with equivalence closure. In *Proc. of LICS2012*, pages 431–440. IEEE, 2012.

15 E. Kieroński and M. Otto. Small substructures and decidability issues for first-order logic with two variables. In *Proc. of LICS2005*, pages 448–457, 2005.

16 E. Kieroński and M. Otto. Small substructures and decidability issues for first-order logic with two variables. *Journal of Symbolic Logic*, 77 (3):729–765, 2012.

17 E. Kieroński and L. Tendera. On finite satisfiability of two-variable first-order logic with equivalence relations. In *Proc. of LICS2009*, pages 123–132, 2009.

18 A. Manuel. Two variables and two successors. In Petr Hlinený and Antonín Kucera, editors, *MFCS*, volume 6281 of *Lecture Notes in Computer Science*, pages 513–524. Springer, 2010.

19 J. Michaliszyn. Decidability of the guarded fragment with the transitive closure. In *Proc. of ICALP2009*, pages 261–272, 2009.

20 M. Mortimer. On languages with two variables. *Zeitschr. f. Logik und Grundlagen d. Math.*, 21:135–140, 1975.

21 M. Niewerth and T. Schwentick. Two-variable logic and key constraints on data words. In Tova Milo, editor, *ICDT*, pages 138–149. ACM, 2011.

22 M. Otto. Two-variable first-order logic over ordered domains. *Journal of Symbolic Logic*, 66:685–702, 2001.

23 F. Ramsey. On a problem of formal logic. *Proc. London Math. Soc. series 2*, 30:264–286, 1930.

24 T. Schwentick and T. Zeume. Two-variable logic with two order relations - (extended abstract). In *CSL*, pages 499–513, 2010.

25 D. Scott. A decision method for validity of sentences in two variables. *J. Symb. Logic*, 27:477, 1962.

26 W. Szwast and L. Tendera. On the decision problem for the guarded fragment with transitivity. In *Proc. of LICS2001*, pages 147–156, 2001.

27 W. Szwast and L. Tendera. The guarded fragment with transitive guards. *Annals of Pure and Applied Logic*, 128:227–276, 2004.

On Pairwise Spanners*

Marek Cygan[1], Fabrizio Grandoni[1], and Telikepalli Kavitha[2]

1 **IDSIA, University of Lugano, Switzerland;** `{marek, fabrizio}@idsia.ch`
2 **Tata Institute of Fundamental Research, India;** `kavitha@tcs.tifr.res.in`

Abstract

Given an undirected n-node unweighted graph $G = (V, E)$, a spanner with *stretch function* $f(\cdot)$ is a subgraph $H \subseteq G$ such that, if two nodes are at distance d in G, then they are at distance at most $f(d)$ in H. Spanners are very well studied in the literature. The typical goal is to construct the sparsest possible spanner for a given stretch function.

In this paper we study *pairwise spanners*, where we require to approximate the u-v distance only for pairs (u, v) in a given set $\mathcal{P} \subseteq V \times V$. Such $\mathcal{P}$-spanners were studied before [Coppersmith,Elkin'05] only in the special case that $f(\cdot)$ is the identity function, i.e. distances between relevant pairs must be preserved exactly (a.k.a. *pairwise preservers*).

Here we present pairwise spanners which are at the same time sparser than the best known preservers (on the same $\mathcal{P}$) and of the best known spanners (with the same $f(\cdot)$). In more detail, for arbitrary $\mathcal{P}$, we show that there exists a $\mathcal{P}$-spanner of size $O(n(|\mathcal{P}| \log n)^{1/4})$ with $f(d) = d + 4 \log n$. Alternatively, for any $\varepsilon > 0$, there exists a $\mathcal{P}$-spanner of size $O(n|\mathcal{P}|^{1/4}\sqrt{\frac{\log n}{\varepsilon}})$ with $f(d) = (1 + \varepsilon)d + 4$. We also consider the relevant special case that there is a critical set of nodes $S \subseteq V$, and we wish to approximate either the distances within nodes in S or from nodes in S to any other node. We show that there exists an $(S \times S)$-spanner of size $O(n\sqrt{|S|})$ with $f(d) = d + 2$, and an $(S \times V)$-spanner of size $O(n\sqrt{|S| \log n})$ with $f(d) = d + 2 \log n$. All the mentioned pairwise spanners can be constructed in polynomial time.

1998 ACM Subject Classification G.2.2 Graphs Algorithms

Keywords and phrases Undirected graphs, shortest paths, additive spanners, distance preservers

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.209

1 Introduction

Let $G = (V, E)$ be an undirected unweighted graph. A subgraph H of G is a *spanner* with stretch function $f(\cdot)$ if, given any two nodes $s, t \in V$ at distance $\delta_G(s, t)$ in G, the distance $\delta_H(s, t)$ between the same two nodes in H is at most $f(\delta_G(s, t))$. An (α, β) spanner is a spanner with stretch functions $f(d) = \alpha \cdot d + \beta$. ($\alpha$ and β are the *multiplicative stretch* and *additive stretch* of the spanner, respectively). If $\beta = 0$ the spanner is called *multiplicative*, and if $\alpha = 1$ the spanner is called *purely-additive*. Spanners are very well studied in the literature (see Section 1.2). The typical goal is to achieve the sparsest possible spanner for a given stretch function $f(\cdot)$ [4, 5, 11, 12, 13, 15, 17, 19, 20, 22].

In this paper we address the natural problem of finding (even sparser) spanners in the case that we want to approximately preserve distances only among a given subset $\mathcal{P} \subseteq V \times V$ of pairs. More formally a *pairwise spanner* on pairs $\mathcal{P}$, or $\mathcal{P}$-spanner for short, with stretch function $f(\cdot)$ is a subgraph $H \subseteq G$ such that, for any $(s, t) \in \mathcal{P}$, $\delta_H(s, t) \leq f(\delta_G(s, t))$. In

* Partially supported by the ERC Starting Grant NEWNET 279352. This work was partially done while the third author was visiting IDSIA.

particular, a classical (all-pairs) spanner is a $(V \times V)$-spanner. Pairwise spanners capture scenarios where we only (or mostly) care about some distances in the graph.

To the best of our knowledge, pairwise spanners were studied before only in the special case that $f(\cdot)$ is the identity function, i.e. distances between relevant pairs have to be preserved exactly. Coppersmith and Elkin [8] call such spanners *pairwise (distance) preservers*, and show that one can compute pairwise preservers of size (i.e., number of edges) $O(\min\left\{|\mathcal{P}|\sqrt{n},\ n\sqrt{|\mathcal{P}|}\right\})$.

The authors left it as an open problem to study the *approximate* variants of these preservers, i.e. what we call pairwise spanners here. This paper takes the first step in answering this question. We show that (for suitable $\mathcal{P}$) it is possible to achieve $\mathcal{P}$-spanners which are at the same time sparser than the preservers in [8] (on the same set $\mathcal{P}$) and than the sparsest known classical spanners (with the same stretch function).

1.1 Our Results and Techniques

In this paper we present some polynomial-time algorithms to construct (α, β) $\mathcal{P}$-spanners for unweighted graphs. Our spanners are either purely-additive (i.e. $\alpha = 1$) or *near-additive* (i.e. $\alpha = 1 + \varepsilon$ for an arbitrarily small $\varepsilon > 0$). For arbitrary $\mathcal{P}$, we achieve the following main results (see Section 5).

▶ **Theorem 1. (near-additive pairwise)** *For any $\varepsilon > 0$ and any $\mathcal{P} \subseteq V \times V$, there is a polynomial time algorithm to compute a $(1+\varepsilon, 4)$ $\mathcal{P}$-spanner of size $O(n|\mathcal{P}|^{1/4}\sqrt{\log n/\varepsilon})$.*

▶ **Theorem 2. (purely-additive pairwise)** *For any integer $k \geq 1$ and any $\mathcal{P} \subseteq V \times V$, there is a polynomial time algorithm to compute a $(1, 4k)$ $\mathcal{P}$-spanner of size $O(n^{1+1/(2k+1)}(\sqrt{(4k+5)|\mathcal{P}|})^{k/(2k+1)})$.*

We also consider the relevant special case that all the pairs involve at least one node from a critical set $S \subseteq V$. More precisely, we distinguish two types of such pairwise spanners: in *subsetwise spanners* (see Section 3) we wish to approximate distances *between* nodes in S, i.e. $\mathcal{P} = S \times S$; in *sourcewise spanners* (see Section 4) we wish to approximate distances *from* nodes in S, i.e. $\mathcal{P} = S \times V$. We obtain the following improved results for the mentioned cases.

▶ **Theorem 3. (subsetwise)** *For any $S \subseteq V$, there is a polynomial time algorithm to compute a $(1, 2)$ $(S \times S)$-spanner of size $O(n\sqrt{|S|})$.*

▶ **Theorem 4. (sourcewise)** *For any $S \subseteq V$ and any integer $k \geq 1$, there is a polynomial time algorithm to compute a $(1, 2k)$ $(S \times V)$-spanner of size $O(n^{1+1/(2k+1)}(k|S|)^{k/(2k+1)})$.*

In particular, by choosing $k = \log n$, we obtain a $(1, 2\log n)$ sourcewise spanner of size $O(n\sqrt{|S|\log n})$, and a $(1, 4\log n)$ pairwise spanner of size $O(n(|\mathcal{P}|\log n)^{1/4})$.

All our spanners rely on a path-buying strategy which was first exploited in the $(1, 6)$ spanner by Baswana et al. [4]. The high-level idea is as follows. There is an initial clustering phase, where we compute a suitable clustering of the nodes, and an associated subset of edges which are added to the spanner. Then there is a path-buying phase, where we consider an appropriate sequence of paths, and decide whether to add or not each path in the spanner under construction[1]. In particular, each path has a *cost* which is given by the number of

[1] In the spanner from Theorem 1 there is also a final step where we add a multiplicative $(2\log n, 0)$-spanner.

edges of the path not already contained in the spanner, and a *value* which measures *how much* the path helps to satisfy the considered set of constraints on pairwise distances. If the value is sufficiently larger than the cost, we add the considered path to the spanner, otherwise we do not.

In more detail, all our pairwise spanners exploit the same clustering phase. We compute a partition $\mathcal{C} = \{C_1, \ldots, C_q\}$ of a subset of the nodes, and call *unclustered* the remaining nodes $V - \cup_i C_i$. The initial value of the spanner is $G_{\mathcal{C}} = (V, E_{\mathcal{C}})$, where $E_{\mathcal{C}}$ contains all the edges of G but possibly a subset of the *inter-cluster* edges (with endpoints in two different clusters). The common clustering phase is described in Section 2.

During the path-buying phase we add to the spanner some extra *inter-cluster* edges. Here we need to finely tune the sequence of paths that we consider, and also the definition of value of a path. In our subsetwise and sourcewise spanners the value of a path ρ reflects the number of pairs (v, C), where v is the endpoint of some pair and C is a cluster, such that adding ρ to the current spanner decreases the distance between v and (the closest node in) C. In the remaining pairwise spanners, we use a similar notion of value, but considering the distance between pairs of clusters (C', C'').

The sequence of paths used in our subsetwise spanner and near-additive pairwise spanner is simply given by the shortest paths among the relevant pairs. This naturally generalizes the set of paths considered in [4]. However, for the sourcewise spanner and the purely-additive pairwise spanner we need to consider a carefully constructed sequence of paths, which includes slightly suboptimal paths. In more detail, we start with the set of shortest paths between the relevant pairs. Then, for each such path ρ, if the cost of ρ is sufficiently smaller than its value, we include ρ in the spanner. Otherwise, we replace ρ with a *slightly longer* path ρ' between the same endpoints which is *much cheaper*, and iterate the process on ρ'. After a small number of iterations, the considered path becomes cheap enough and hence we include it in the spanner. This (non-trivial) iterative construction of candidate paths during the path-buying phase is probably the main algorithmic contribution of this paper.

1.2 Related Work

Graph spanners were introduced by Peleg and Schaffer [17] in 1989. Spanners have been extensively studied since then, and there are numerous applications involving spanners, such as algorithms for approximate shortest paths [1, 7, 12], labeling schemes [16, 14], approximate distance oracles [21, 6, 3], routing [2, 9, 10], and network design [18].

There are several algorithms for computing multiplicative and additive spanners in weighted and unweighted graphs. In unweighted graphs, for any integer $k \geq 1$, Halperin and Zwick [15] gave a linear time algorithm to compute a multiplicative $(2k-1, 0)$-spanner of size $O(n^{1+1/k})$, where n is the number of vertices. Note that for $k = \log n$ one obtains a spanner with multiplicative stretch $O(\log n)$ and with size $O(n)$: we will use this type of spanner in Theorem 1. Analogous results are also known for weighted graphs [5, 20, 19].

The first purely-additive spanner (for unweighted graphs) is due to Dor et al. [11]. They describe a $(1, 2)$ spanner of size $O(n^{3/2} \log n)$. This was subsequently improved to $O(n^{3/2})$ [13]. Note that our subsetwise spanner from Theorem 3 generalizes this result: in particular, it has the same stretch function and is sparser whenever $|S| = o(n)$. Baswana et al. [4] describe a $(1, 6)$-spanner of size $O(n^{4/3})$. Whenever $|\mathcal{P}| = O(n^{4/3-\delta})$ for some constant $\delta > 0$, we achieve an asymptotically sparser pairwise spanner with constant additive stretch (depending on δ). The same holds for our sourcewise spanner if $|S| = O(n^{2/3-\delta})$.

The result in [15] shows an elegant trade-off between the size of the spanner and its multiplicative stretch. No such trade-off is known for purely-additive spanners. In particular, the spanner in [4] is the sparsest known purely-additive spanner. Theorems 2 and 4 show a non-trivial trade-off between the size and additive stretch of pairwise spanners.

There have also been several results on near-additive spanners [13, 12, 22]. For example, there is a $(1+\epsilon, 4)$-spanner of size $O(\frac{n^{4/3}}{\epsilon})$ for any $\epsilon > 0$ [13]. Our pairwise spanner from Theorem 1 has the same stretch function, and is sparser for $|\mathcal{P}| = o(n^{4/3}/(\varepsilon \log n)^2)$.

Compared to the preservers in [8], we achieve sparser pairwise spanners with additive stretch $O(\log n)$ for $|\mathcal{P}| = \omega(n^{2/3} \log^{1/3} n)$, and a sparser subsetwise spanners for $|S| = \omega(n^{1/3})$. Interestingly, our sourcewise spanners are always sparser than the pairwise preservers from [8].

2 Clustering

A *clustering* of a graph $G = (V, E)$ is a collection $\mathcal{C} = \{C_1, \ldots, C_q\}$ of pairwise disjoint subsets of nodes $C_i \subseteq V$. Note that we do not require $\mathcal{C}$ to span all the nodes V: we call *unclustered* the nodes $V - \cup_i C_i$.

We will crucially exploit the following construction of a clustering $\mathcal{C}$ and of an associated *cluster subgraph* $G_{\mathcal{C}}$.

▶ **Lemma 5.** *There is a polynomial time algorithm which, given $\beta \in [0,1]$ and a graph $G = (V, E)$, computes a clustering $\mathcal{C}$ with at most $n^{1-\beta}$ clusters and a subgraph $G_{\mathcal{C}}$ of size $O(n^{1+\beta})$ such that:*

1. **(missing-edge property)** *If an edge $uv \in E$ is absent in $G_{\mathcal{C}}$, then u and v belong to two different clusters.*
2. **(cluster-diameter property)** *The distance in $G_{\mathcal{C}}$ between any two vertices of the same cluster is at most 2.*

Proof. Let U be the set of nodes which are not yet clustered (initially we set $U := V$). As long as there exists a vertex $v \in V$ with at least $\lceil n^{\beta} \rceil$ neighbors in U, let C contain exactly $\lceil n^{\beta} \rceil$ arbitrary neighbors of v in U. Add C to $\mathcal{C}$, set $U := U \setminus C$ and add to $G_{\mathcal{C}}$ all the edges of G with both endpoints in $C \cup \{v\}$. When no node v satisfies the mentioned property, we stop creating new clusters and add to $G_{\mathcal{C}}$ all the edges incident to the final set of unclustered nodes U.

By construction, clusters are pairwise disjoint. Each time we create a new cluster, the size of U decreases by at least n^{β}, hence there cannot be more than $n^{1-\beta}$ clusters. Any two nodes in the same cluster C have some common neighbor v in $G_{\mathcal{C}}$, hence Property 2 is satisfied. By construction, all the edges incident to unclustered nodes plus the intra-cluster edges (with both endpoints in the same cluster) belong to $G_{\mathcal{C}}$, which implies Property 1.

It remains to bound the number of edges of $G_{\mathcal{C}}$. Each time we create a new cluster, the number of edges of $G_{\mathcal{C}}$ grows by at most $O(n^{2\beta})$: this gives $O(n^{1-\beta} n^{2\beta}) = O(n^{1+\beta})$ edges altogether. When we stop creating clusters, each (clustered or unclustered) node v has at most n^{β} neighbors in U: consequently the number of edges incident to unclustered nodes that we add at the end of the procedure is at most $O(n^{1+\beta})$. ◀

The following technical lemma turns out to be useful in the remaining sections.

▶ **Lemma 6.** *Let $\mathcal{C}$ and $G_{\mathcal{C}}$ be constructed with the procedure from Lemma 5 w.r.t. a given graph $G = (V, E)$. If the shortest path ρ in G between any two nodes $u, v \in V$ contains t edges that are absent in $G_{\mathcal{C}}$, then there are at least $t/2$ clusters of $\mathcal{C}$ having at least one vertex on ρ.*

Proof. We prove the lemma by counting pairs (u, e), where e is an edge of ρ absent in $G_{\mathcal{C}}$ and u is one of the endpoints of e: let $\mathcal{S}$ be the set of such pairs. Since ρ contains t edges that are absent in $G_{\mathcal{C}}$ there are exactly $2t$ pairs in $\mathcal{S}$ (each edge $e = uv$ belongs to two pairs: (u, e) and (v, e)). We say that a cluster $C \in \mathcal{C}$ *owns* a pair (u, e) if $u \in C$. By the missing-edge property, each edge e of ρ absent in $G_{\mathcal{C}}$ has both endpoints clustered, hence each pair of $\mathcal{S}$ is owned by some cluster.

Let us assume that there are x clusters of $\mathcal{C}$ having at least one vertex on ρ. By the cluster-diameter property any cluster $C \in \mathcal{C}$ contains at most 3 vertices on ρ, since otherwise ρ would not be a shortest path between u and v. However, if a cluster $C \in \mathcal{C}$ contains exactly 3 vertices on ρ, those have to be consecutive vertices a, b, c of ρ, since ρ is a shortest path and we know by the cluster-diameter property that there is a path of length at most 2 between every pair in $\{a, b, c\}$. By the missing-edge property both edges ab and bc are present in $G_{\mathcal{C}}$, and consequently C owns at most two pairs of $\mathcal{S}$. Clearly if a cluster $C \in \mathcal{C}$ contains at most 2 vertices on ρ, then it owns at most 4 pairs of $\mathcal{S}$. Therefore each cluster owns at most 4 pairs of $\mathcal{S}$: since $\mathcal{S}$ has $2t$ pairs we have $x \geq t/2$. ◀

3 Subsetwise Spanners

In this section we present our algorithm to compute a subsetwise spanner, and prove Theorem 3.

Our algorithm consists of two main phases: a clustering phase and a path-buying phase. In the clustering phase we invoke Lemma 5 and obtain a cluster subgraph $G_{\mathcal{C}}$ of G of size $O(n^{1+\beta})$, together with a set $\mathcal{C}$ of at most $n^{1-\beta}$ clusters. The value of β will be defined later.

In the path-buying phase we proceed as follows. Initially set $G_0 := G_{\mathcal{C}}$ and let $\{\rho_1, \ldots, \rho_z\}$ denote the set of $z = \binom{|S|}{2}$ shortest paths between all pairs of vertices in S. We let (u_i, v_i) denote the endpoints of ρ_i. Next, we iterate over the paths ρ_i for $i = 1, \ldots, z$. To determine which paths are affordable, we define the functions value$(\cdot)$ and cost$(\cdot)$:

- let cost(ρ_i) be the number of edges of ρ_i that are absent in G_{i-1}
- let value(ρ_i) be the number of pairs (x, C), where $x \in \{u_i, v_i\}$ and $C \in \mathcal{C}$ is a cluster, such that ρ_i contains at least one vertex of C and the distance between x and C in the graph G_{i-1} is strictly greater than the distance between u and C in ρ_i, i.e., $\delta_{G_{i-1}}(x, C) > \delta_{\rho_i}(x, C)$.

Our path-buying strategy is as follows. If

$$\text{cost}(\rho_i) \leq 2\,\text{value}(\rho_i)$$

then we buy the path ρ_i, that is we set $G_i := G_{i-1} \cup \rho_i$ (in words, G_i is given by G_{i-1} plus the edges of ρ_i not in G_{i-1}). Otherwise (i.e., 2value$(\rho_i) <$ cost(ρ_i)), we do not buy ρ_i and set $G_i := G_{i-1}$. The subsetwise spanner is given by $G_s := G_z$.

The next two lemmas bound the stretch and the size of the constructed spanner G_s, respectively

▶ **Lemma 7.** *For any $(u_i, v_i) \in \mathcal{P}$, $\delta_{G_s}(u_i, v_i) \leq \delta_G(u_i, v_i) + 2$.*

Proof. Clearly the claim holds if our algorithm bought the path ρ_i, hence we assume 2value$(\rho_i) <$ cost(ρ_i). Let cost$(\rho_i) = t$, that is there are exactly t edges of ρ_i which are not present in the graph G_{i-1}. By Lemma 6 there are at least $t/2$ clusters having at least one vertex on ρ_i. If there is no cluster C among them such that $\delta_{G_{i-1}}(u_i, C) = \delta_G(u_i, C)$ and $\delta_{G_{i-1}}(v_i, C) = \delta_G(v_i, C)$, then all these clusters would contribute to value(ρ_i) (either with

u_i or with v_i or both) which leads to a contradiction, because $t = 2 \cdot (t/2) \leq 2\text{value}(\rho_i) < \text{cost}(\rho_i) = t$.

Thus there is a cluster C having a vertex of ρ_i such that $\delta_{G_{i-1}}(u_i, C) = \delta_G(u_i, C)$ and $\delta_{G_{i-1}}(v_i, C) = \delta_G(v_i, C)$. This implies:

$$\begin{aligned} \delta_{G_s}(u_i, v_i) \ \leq \ \delta_{G_{i-1}}(u_i, v_i) &\leq \delta_{G_{i-1}}(u_i, C) + \delta_{G_{i-1}}(v_i, C) + 2 \\ &\leq \delta_G(u_i, C) + \delta_G(v_i, C) + 2 \\ &\leq \delta_G(u_i, v_i) + 2, \end{aligned}$$

where the first inequality is because G_{i-1} is a subgraph of G_s, the second inequality holds since any two vertices of C are at distance at most two in $G_{\mathcal{C}} \subseteq G_{i-1}$ (by the cluster-diameter property) and the last inequality follows from the assumption that C contains a vertex of ρ_i. ◀

▶ **Lemma 8.** *For β such that $n^\beta = \sqrt{|S|}$ the graph G_s contains $O(n\sqrt{|S|})$ edges.*

Proof. The clustering phase produces a graph with $O(n^{1+\beta}) = O(n\sqrt{|S|})$ edges. Let $\mathcal{B}$ be the set of paths bought in the path-buying phase. The total number of edges that appear in $\mathcal{B}$ and do not appear in $G_{\mathcal{C}}$ is equal to $\sum_{\rho_i \in \mathcal{B}} \text{cost}(\rho_i)$, which is upper bounded by $\sum_{\rho_i \in \mathcal{B}} 2\text{value}(\rho_i)$. Observe that after the first contribution of a pair (x, C) to the above sum, the distance between x and C is at most $\delta_G(x, C) + 2$, hence each pair (x, C) can contribute to the sum at most 3 times. Therefore the total number of edges added in the second phase of our algorithm is upper bounded by $O(n^{1-\beta}|S|) = O(n\sqrt{|S|})$. ◀

The proof of Theorem 3 follows from Lemmas 7 and 8.

4 Sourcewise spanners

In this section we present our algorithm to compute a sourcewise spanner from sources S, and prove Theorem 4.

Our algorithm again consists of two phases, where the first is a clustering phase and the second is a path-buying phase. The clustering phase is as in the algorithm from previous section, for a proper value of β to be defined later. Let $\mathcal{C}$ and $G_{\mathcal{C}}$ be the resulting clustering and cluster subgraph.

At the start of the second phase we set $G_0 := G_{\mathcal{C}}$ and define $\{\rho_1, \ldots, \rho_z\}$ as the set of shortest paths between any two vertices of V such that at least one of them belongs to S. Let us assume that the path ρ_i is a shortest path between $u_i \in S$ and $v_i \in V$. Next, we iterate over paths ρ_i for $i = 1, \ldots, z$. For a given i we are going to define paths ρ_i^j, where $0 \leq j \leq k$, maintaining the following invariants:

(i) ρ_i^j is a path between u_i and v_i in G of length at most $\delta_G(u_i, v_i) + 2j$,
(ii) any cluster $C \in \mathcal{C}$ contains at most three vertices of ρ_i^j,
(iii) $\text{cost}(\rho_i^j) \leq 2n^{1-\beta}/\gamma^j$, where $\text{cost}(\rho_i^j)$ is the number of edges of ρ_i^j absent in G_{i-1}, and $\gamma = (3n^{1-\beta})^{1/k}$.

Our algorithm will buy exactly one path ρ_i^j for $0 \leq j \leq k$, which will ensure (by Invariant (i)) that in G_i, the distance between u_i and v_i is at most $\delta_G(u_i, v_i) + 2k$.

We set $\rho_i^0 := \rho_i$. Observe that for $j = 0$, Invariant (i) is trivially satisfied, Invariant (ii) is satisfied by the cluster-diameter property (otherwise ρ_i would not be a shortest path), and Invariant (iii) is satisfied because there are at most $n^{1-\beta}$ clusters in $\mathcal{C}$ and consequently by Lemma 6 $\text{cost}(\rho_i) \leq 2n^{1-\beta}$.

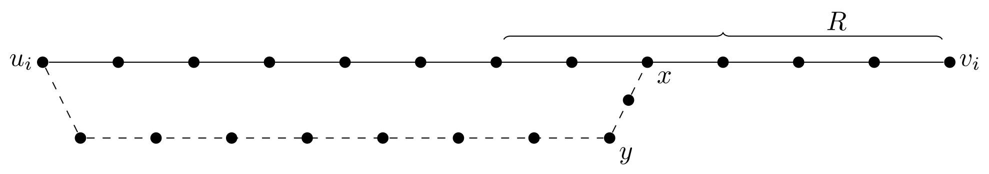


Figure 1 The solid edges represent a path ρ_i^j, while the dashed edges denote the new prefix of the path ρ_i^{j+1}.

Say we have constructed ρ_i^j, where $j \in \{0, \dots, k\}$. Let us define the function value(ρ_i^j) as the number of clusters $C \in \mathcal{C}$ such that C contains a vertex of ρ_i^j and the distance between u_i and C in G_{i-1} is strictly greater than the distance between u_i and C in ρ_i^j, i.e., $\delta_{G_{i-1}}(u_i, C) > \delta_{\rho_i^j}(u_i, C)$. Now we check the condition

$$\text{cost}(\rho_i^j) \leq 3\gamma\text{value}(\rho_i^j).$$

If that is the case, then we buy the path ρ_i^j. That is, G_i is set to $G_{i-1} \cup \rho_i^j$. We ignore the remaining values of j and proceed with the next value of i. Else we construct ρ_i^{j+1} as follows:

Let R be the longest suffix of ρ_i^j containing exactly $\lfloor \text{cost}(\rho_i^j)/\gamma \rfloor$ edges that are absent in G_{i-1}. Observe that the first node of R is clustered: by the maximality of R, the edge e of ρ_i^j preceding R is absent in G_{i-1}, and hence both the endpoints of e (one of which is the first node of R) are clustered by the missing-edge property of $G_{\mathcal{C}}$. Consequently at least $1 + \lfloor \text{cost}(\rho_i^j)/\gamma \rfloor \geq \text{cost}(\rho_i^j)/\gamma$ vertices of R are clustered, as R contains $\lfloor \text{cost}(\rho_i^j)/\gamma \rfloor$ edges absent in G_{i-1} and the endpoints of these edges are clustered.

By Invariant (ii) there are at least $\text{cost}(\rho_i^j)/(3\gamma)$ clusters in $\mathcal{C}$ having at least one vertex of R. Since we did not buy ρ_i^j, there exists a cluster $C \in \mathcal{C}$ containing a vertex $x \in C$ of R such that the distance between u_i and C in G_{i-1} is at most the distance between u_i and x in ρ_i^j. We construct the path ρ_i^{j+1} by taking a shortest path in G_{i-1} from u_i to the closest node $y \in C$, then we add a path of length at most two between y and x (which exists in $G_{\mathcal{C}}$ hence in G_{i-1} by the cluster-diameter property), and finally add the suffix of R starting at x (see Fig. 1).

Let us show that ρ_i^{j+1} maintains the invariants. Note that by construction, Invariant (i) is satisfied, since the length of ρ_i^{j+1} is at most the length of ρ_i^j plus 2. Then, as long as there is a cluster $C \in \mathcal{C}$ containing at least four vertices on ρ_i^{j+1}, we let a, b be the vertices of ρ_i^{j+1} closest to u_i and v_i respectively. Note that there are at least three edges on ρ_i^{j+1} between a and b, hence we can replace the subpath of ρ_i^{j+1} by adding the at most two edges of $G_{\mathcal{C}}$ guaranteed by the cluster-diameter property. Consequently, Invariant (ii) is satisfied. Moreover, by the choice of R, Invariant (iii) is also satisfied. This finishes the construction of ρ_i^{j+1}.

Observe that by Invariant (iii) we have $\text{cost}(\rho_i^k) \leq 2/3$: since cost$(\cdot)$ has only integral values, it has to be that $\text{cost}(\rho_i^k) = 0$, which ensures that we buy a path ρ_i^j for some $j \leq k$.

Finally, as our spanner G_s we take $G_s := G_z$.

▶ **Lemma 9.** *For any pair $(u_i, v_i) \in \mathcal{P}$, $\delta_{G_s}(u_i, v_i) \leq \delta_G(u_i, v_i) + 2k$.*

Proof. From the above discussion, we buy at least one path ρ_i^j for some $0 \leq j \leq k$. By Invariant (i), the length of the latter path is at most the length of the shortest path ρ_i between u_i and v_i plus $2k$. ◀

▶ **Lemma 10.** *For β such that $n^{\beta} = (n^{1/k}(2k+3)|S|)^{k/(2k+1)}$, the subgraph G_s contains $O(n^{1+1/(2k+1)}(k|S|)^{k/(2k+1)})$ edges.*

Proof. To bound the size of G_s we recall that in the first phase we have inserted $O(n^{1+\beta})$ edges. Let $0 \le j_i \le k$ be the index of a path $\rho_i^{j_i}$ bought for a given i. We claim, that any cluster C contributes to value$(\rho_i^{j_i})$ of at most $|S|(2k+3)$ bought paths. This holds because when for $u_i \in S$ a supported path is bought the distance between u_i and C is at most $2k+2$ greater than the distance between u_i and C in G: otherwise one could shorten $\rho_i^{j_i}$ by more than $2k$, obtaining a contradiction with Invariant (i). Therefore the total number of edges added during the second phase is upper bounded by $\sum_{i=1}^{z} \text{cost}(\rho_i^{j_i}) \le \sum_{i=1}^{z} 3\gamma\text{value}(\rho_i^{j_i}) \le 3\gamma(2k+3)|S|n^{1-\beta}$, since each cluster $C \in \mathcal{C}$ supports at most $|S|(2k+3)$ bought paths. The claim follows. ◀

The proof of Theorem 4 follows from Lemmas 9 and 10.

5 Pairwise spanners

In this section we present our pairwise spanners for arbitrary $\mathcal{P}$. We start with a near-additive spanner (see Section 5.1) and then present a purely-additive spanner (see Section 5.2). In both cases we let $\mathcal{P} = \{(s_1, t_1), \ldots, (s_N, t_N)\}$ denote the set of pairs, $N = |\mathcal{P}|$.

5.1 A Near-Additive Pairwise Spanner

Our algorithm to construct the near-additive $\mathcal{P}$-spanner from Theorem 1 consists of three phases. First, we use Lemma 5 with the value of β to be determined later, obtaining a cluster subgraph $G_{\mathcal{C}}$ of G of size $O(n^{1+\beta})$ together with a set $\mathcal{C}$ of at most $n^{1-\beta}$ clusters.

At the start of the second phase we set $G_0 := G_{\mathcal{C}}$ and consider the set of paths $\{\rho_1, \ldots, \rho_N\}$, where ρ_i is a shortest path between s_i and t_i in G. Next we iterate over the paths ρ_i for $i = 1, \ldots, N$. By cost(ρ_i) we denote the number of edges of ρ_i absent in G_{i-1}, and by value(ρ_i) we denote the number of pairs of clusters $(C_1, C_2) \in \mathcal{C}$, such that both C_1 and C_2 contain at least one vertex of ρ_i and $\delta_{\rho_i}(C_1, C_2) < \delta_{G_{i-1}}(C_1, C_2)$. For a given i if

$$\text{cost}(\rho_i) \le \frac{12 \log n}{\epsilon} \sqrt{\text{value}(\rho_i)},$$

then we buy ρ_i, that is we set $G_i := G_{i-1} \cup \rho_i$. Otherwise we set $G_i := G_{i-1}$.

In the third phase we add to G_N the multiplicative $(2\log n, 0)$ spanner of size $O(n)$ given in [15]: this way we obtain the desired spanner G_s.

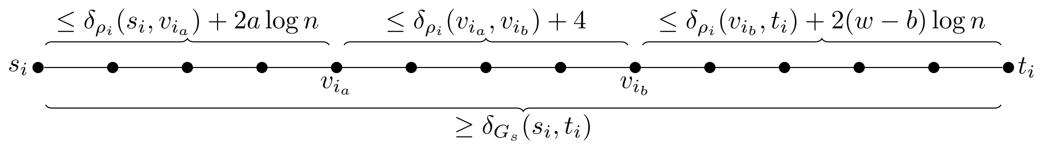
In the following two lemmas we bound the stretch and size of G_s, respectively.

▶ **Lemma 11.** *For each $(s_i, t_i) \in \mathcal{P}$, $\delta_{G_s}(s_i, t_i) \le (1+\varepsilon)\delta_G(s_i, t_i) + 4$.*

Proof. Clearly we can assume that the path ρ_i was not bought in the second phase, since otherwise the claim trivially holds. Therefore cost$(\rho_i) > \frac{12 \log n}{\epsilon}\sqrt{\text{value}(\rho_i)}$.

Let $\rho_i = (v_0 = s_i, v_1, \ldots, v_{\ell-1}, v_\ell = t_i)$ and let $I \subseteq \{0, \ldots, \ell\}$ be the set of all indices j such that v_j is clustered. Observe that if $|I| \le 1$, then by the missing-edge property the whole path ρ_i is present in $G_{\mathcal{C}}$, and hence the claim holds. Therefore denote $I = \{i_0, \ldots, i_w\}$, where $i_0 < i_1 < \ldots < i_w$ and $w \ge 1$. Let $0 \le a \le b \le w$ be two indices, such that $v_{i_a} \in C_1$, $v_{i_b} \in C_2$ (for some $C_1, C_2 \in \mathcal{C}$), $\delta_{\rho_i}(C_1, C_2) \ge \delta_{G_{i-1}}(C_1, C_2)$ and the value of $b - a$ is maximized. Note that such a pair of indices a, b always exists, since we can take $a = b$.

Let $x = a + (w - b)$. Observe that any cluster $C \in \mathcal{C}$ contains at most 3 vertices of $V_I = \{v_{i_j} : 0 \le j \le w\}$, since otherwise by the cluster-diameter property ρ_i would not be a



$\leq \delta_{\rho_i}(s_i, v_{i_a}) + 2a\log n$
$\leq \delta_{\rho_i}(v_{i_a}, v_{i_b}) + 4$
$\leq \delta_{\rho_i}(v_{i_b}, t_i) + 2(w-b)\log n$
s_i
t_i
v_{i_a}
v_{i_b}
$\geq \delta_{G_s}(s_i, t_i)$

Figure 2 Illustration of the three paths concatenation in the proof of Lemma 11.

shortest s_i-t_i path. Therefore there are at least $x/6$ clusters $\mathcal{C}'$ having at least one vertex in the set $\{v_{i_j} : 0 \leq j < \lceil x/2 \rceil\}$, and at least $x/6$ clusters $\mathcal{C}''$ having at least one vertex in the set $\{v_{i_j} : w - \lceil x/2 \rceil < j \leq w\}$. However, each of the at least $(x/6)^2$ pairs of clusters in $\mathcal{C}' \times \mathcal{C}''$ contributes to value(ρ_i) since the difference between indices in the corresponding set is at least $w - \lceil x/2 \rceil + 1 - (\lceil x/2 \rceil - 1) > w - x$. Therefore $w \geq \text{cost}(\rho_i) > \frac{12\log n}{\epsilon}\frac{x}{6}$ and hence $x \leq \frac{w\epsilon}{2\log n}$.

The latter bound on x is sufficient to prove the claim. In fact, consider the path between s_i and t_i in G_s obtained by concatenating the following paths (as illustrated in Fig. 2):

- A shortest path in G_s from s_i to v_{i_a}. Note that in the prefix of ρ_i between s_i and v_{i_a} there are $a+1$ clustered nodes and hence at most a edges absent in G_{i-1} (by the missing-edge property). Since G_s contains the $(2\log n, 0)$-spanner added in the third phase, each missing edge can be replaced by at path of length $2\log n$. Consequently, there is a path from s_i to v_{i_a} of length at most $\delta_{\rho_i}(s_i, v_{i_a}) + 2a\log n$ in G_s.
- A shortest path in G_s from v_{i_a} to v_{i_b}. Let $C_1, C_2 \in \mathcal{C}$ be the clusters containing v_{i_a} and v_{i_b} respectively. We know that in G_{i-1} there is a path from C_1 to C_2 of length at most $\delta_{\rho_i}(v_{i_a}, v_{i_b})$, which can be extended to a path between v_{i_a} and v_{i_b} in G_{i-1} by adding at most 4 edges (by the cluster-diameter property).
- A shortest path in G_s from v_{i_b} to t_i. Observe that in the suffix of ρ_i between v_{i_b} and t_i there are at most $w - b$ edges absent in G_{i-1} by the same argument as above. Hence, thanks to the $(2\log n, 0)$-spanner added in the third phase, there is a path from v_{i_b} to t_i of length at most $\delta_{\rho_i}(v_{i_b}, t_i) + (w-b)2\log n$ in G_s.

The resulting path is of length at most

$$\begin{aligned}&\delta_{\rho_i}(s_i, v_{i_a}) + 2a\log n + \delta_{\rho_i}(v_{i_a}, v_{i_b}) + 4 + \delta_{\rho_i}(v_{i_b}, t_i) + (w-b)2\log n\\ =\ &\delta_G(s_i, t_i) + 2x\log n + 4 \leq (1+\epsilon)\delta_G(s_i, t_i) + 4,\end{aligned}$$

where the last inequality follows from $x \leq \frac{\epsilon w}{2\log n}$ together with $w \leq \delta_G(s_i, t_i)$. ◀

Due to space limitation we postpone the proof of the following lemma to the full version.

▶ **Lemma 12.** *For β such that $n^{2\beta} = \sqrt{N}\frac{\log n}{\epsilon}$ the size of G_s is $O(nN^{1/4}\sqrt{\log n/\epsilon})$.*

Having Lemmas 11 and 12, the proof of Theorem 1 follows.

5.2 A Purely-Additive Pairwise Spanner

In this section we describe an algorithm to compute the purely-additive $\mathcal{P}$-spanner from Theorem 2. To that aim we will combine ideas from the proofs of Theorems 1 and 4.

Our algorithm consists of the usual clustering phase (for an appropriate parameter β) followed by a path-buying phase that we next describe.

Let $\mathcal{C}$ and $G_\mathcal{C}$ be the clustering and the associated cluster graph. At the beginning of the path-buying phase, we set $G_0 := G_\mathcal{C}$ and consider the set $\{\rho_1, \ldots, \rho_N\}$, where ρ_i is a shortest path between s_i and t_i in G. Next we iterate over the paths ρ_i for $i = 1, \ldots, N$. For a given i we are going to define paths ρ_i^j, where $0 \leq j \leq k$, maintaining the following invariants:

(i) ρ_i^j is a path between s_i and t_i in G of length at most $\delta_G(s_i, t_i) + 4j$,
(ii) any cluster $C \in \mathcal{C}$ contains at most three vertices of ρ_i^j,
(iii) $\text{cost}(\rho_i^j) \leq 2n^{1-\beta}/\gamma^j$, where $\text{cost}(\rho_i^j)$ is the number of edges of ρ_i^j absent in G_{i-1}, and $\gamma = (3n^{1-\beta})^{1/k}$.

Our algorithm will buy exactly one path ρ_i^j, which will ensure by Invariant (i) that in G_i the distance between s_i and t_i is at most $\delta_G(u_i, v_i) + 4k$. By $\text{value}(\rho_i^j)$ let us denote the number of pairs of clusters $C_1, C_2 \in \mathcal{C}$, such that both C_1 and C_2 contain at least one vertex of ρ_i^j and $\delta_{\rho_i^j}(C_1, C_2) < \delta_{G_{i-1}}(C_1, C_2)$.

We set $\rho_i^0 := \rho_i$. Observe that for $j = 0$ Invariant (i) is trivially satisfied, Invariant (ii) is satisfied by the cluster-diameter property (otherwise ρ_i would not be a shortest path), and Invariant (iii) is satisfied because there are at most $n^{1-\beta}$ clusters in $\mathcal{C}$ and consequently by Lemma 6, $\text{cost}(\rho_i) \leq 2n^{1-\beta}$.

Say we have constructed ρ_i^j, where $j \in \{0, \ldots, k\}$. If

$$\text{cost}(\rho_i^j) \leq 6\gamma\sqrt{\text{value}(\rho_i^j)}\,,$$

then we buy the path ρ_i^j, i.e. as G_i we take the union of G_{i-1} and ρ_i^j, ignore remaining values of j and proceed with the next value of i. Otherwise (i.e., $\text{cost}(\rho_i^j) > 6\gamma\sqrt{\text{value}(\rho_i^j)}$), we construct a path ρ_i^{j+1} as follows:

Let $\rho_i^j = (v_0 = s_i, v_1, \ldots, v_{\ell-1}, v_\ell = t_i)$ and let $I \subseteq \{0, \ldots, \ell\}$ be the set of all indices j such that v_j is clustered. Observe that if $|I| \leq 1$, then by the missing-edge property the whole path ρ_i^j is present in $G_{\mathcal{C}}$, and hence it is of zero cost, which contradicts the assumption $\text{cost}(\rho_i^j) > 6\gamma\sqrt{\text{value}(\rho_i^j)}$. Therefore denote $I = \{i_0, \ldots, i_w\}$, where $i_0 < i_1 < \ldots < i_w$ and $w \geq 1$. Let $0 \leq a \leq b \leq w$ be two indices, such that $v_{i_a} \in C_1$, $v_{i_b} \in C_2$ (for some $C_1, C_2 \in \mathcal{C}$), $\delta_{\rho_i^j}(C_1, C_2) \geq \delta_{G_{i-1}}(C_1, C_2)$ and the value of $b - a$ is maximized. Note that such a pair of indices a, b always exists, since we can take $a = b$.

Let $x = a + (w - b)$. By Invariant (ii) there are at least $x/6$ clusters $\mathcal{C}'$ having at least one vertex in the set $\{v_{i_j} : 0 \leq j < \lceil x/2 \rceil\}$, and at least $x/6$ clusters $\mathcal{C}''$ having at least one vertex in the set $\{v_{i_j} : w - \lceil x/2 \rceil < j \leq w\}$. However, each of the at least $(x/6)^2$ pairs of clusters in $\mathcal{C}' \times \mathcal{C}''$ contributes to $\text{value}(\rho_i^j)$ since the difference between indices in the corresponding set is at least $w - \lceil x/2 \rceil + 1 - (\lceil x/2 \rceil - 1) > w - x$. Therefore

$$\text{cost}(\rho_i^j) > 6\gamma\sqrt{\text{value}(\rho_i^j)} \geq 6\gamma\sqrt{\left(\frac{x}{6}\right)^2} = \gamma x \quad \Rightarrow \quad x \leq \text{cost}(\rho_i^j)/\gamma. \tag{1}$$

We construct the path ρ_i^{j+1} by appending the following three paths A, B, and C:

- As A we take the prefix of ρ_i^j from s_i to v_{i_a}. Note that this prefix contains $a+1$ clustered nodes and hence at most a edges absent in G_{i-1} (by the missing-edge property of $G_{\mathcal{C}}$).
- Let $C_1, C_2 \in \mathcal{C}$ be the clusters containing v_{i_a} and v_{i_b} respectively. We know that in G_{i-1} there is a path from C_1 to C_2 of length at most $\delta_{\rho_i^j}(v_{i_a}, v_{i_b})$, which can be extended to a path B between v_{i_a} and v_{i_b} in G_{i-1} by adding at most 4 edges (by the cluster-diameter property).
- As C we take the suffix of ρ_i^j from v_{i_b} to t_i, which contains at most $w - b$ edges absent in G_{i-1} by the same argument as above.

Observe that ρ_i^{j+1} contains at most $a+(w-b)=x$ edges absent in G_{i-1}, hence by (1) we ensure Invariant (iii). Moreover the length of ρ_i^{j+1} is at most the length of ρ_i^j plus 4, which ensures Invariant (i). In order to ensure Invariant (ii), as long as there exists a cluster $C \in \mathcal{C}$ containing at least 4 vertices of ρ_i^{j+1} we let u and v be two such vertices closest to s_i and t_i on ρ_i^{j+1} respectively and replace the subpath of ρ_i^{j+1} between u and v (which is of length at least three) by a path of length at most two in G_{i-1} (which exists by the cluster-diameter property).

Observe that by Invariant (iii) we have $\mathrm{cost}(\rho_i^k) \le 2/3$, hence $\mathrm{cost}(\rho_i^k)=0$ which ensures that we buy a path ρ_i^j for some $j \le k$. Finally, as our spanner G_s we take $G_s := G_N$.

▶ **Lemma 13.** *For each $(s_i,t_i) \in \mathcal{P}$, $\delta_{G_s}(s_i,t_i) \le \delta_G(s_i,t_i)+4k$.*

Proof. From the above discussion, G_s contains at least one path ρ_i^j between s_i and t_i for some $0 \le j \le k$. The claim follows by Invariant (i). ◀

▶ **Lemma 14.** *For β such that $n^\beta = (6n^{1/k}\sqrt{(4k+5)N})^{k/(2k+1)}$ the size of G_s is $O(n^{1+1/(2k+1)}(\sqrt{(4k+5)N})^{k/(2k+1)})$.*

Proof. The clustering phase gives $O(n^{1+\beta})$ edges, which matches the desired bound on G_s. Let $0 \le j_i \le k$ be the index of a path $\rho_i^{j_i}$ bought for a given i. We claim, that any pair of clusters contributes to $\mathrm{value}(\rho_i^{j_i})$ of at most $(4k+5)$ bought paths. Observe, that if a pair of clusters C_1, C_2 contributes to $\mathrm{value}(\rho_i^{j_i})$, then when $\rho_i^{j_i}$ is bought we have $\delta_{G_i}(C_1,C_2) \le \delta_G(C_1,C_2)+4k+4$, since otherwise the subpath of $\rho_i^{j_i}$ between C_1 and C_2 might be shortened by more than $4k$, contradicting Invariant (i). The total number of edges added in the second phase is upper bounded by

$$\begin{aligned}\sum_{1\le i\le N} \mathrm{cost}(\rho_i^{j_i}) &\le \sum_{1\le i\le N} 6\gamma\sqrt{\mathrm{value}(\rho_i^{j_i})} \\ &\overset{\substack{\text{Cauchy-Schwarz}\\ \text{inequality}}}{\le} 6\gamma\sqrt{\sum_{1\le i\le N}\mathrm{value}(\rho_i^{j_i})}\sqrt{N} \\ &\le 6\gamma\sqrt{4k+5}\,n^{1-\beta}\sqrt{N}\,.\end{aligned}$$

By substituting γ and β the claim follows. ◀

Theorem 2 follows from Lemmas 13 and 14

6 Conclusions

We considered a natural extension to the problem of computing a sparse spanner in an undirected unweighted graph. Along with the input graph $G=(V,E)$, a subset $\mathcal{P} \subseteq V \times V$ of relevant pairs of vertices is also given here and we seek a sparse subgraph H of G such that for every pair (u,v) in $\mathcal{P}$, the u-v distance $\delta_H(u,v)$ in the subgraph is close to the u-v distance $\delta_G(u,v)$ in G. We showed sparse subgraphs H where $\delta_H(u,v)$ is a small additive or near-additive stretch away from $\delta_G(u,v)$.

The pairwise preservers in [8] are at the same time more accurate and sparser than our spanners for small enough values of $|\mathcal{P}|$. In particular, in that range of values of $|\mathcal{P}|$ the authors exploit a construction which does not seem to benefit from allowing a larger stretch. The authors also present lower bounds on the size of any preserver, however it is unclear whether those lower bounds extend to the case of pairwise spanners (where distances have to be approximated rather than preserved). Obtaining sparser pairwise spanners for very small $|\mathcal{P}|$, if possible, is an interesting open problem.

References

1 B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Near-linear time construction of sparse neighborhood covers. *SIAM Journal on Computing*, 28(1):263-277, 1998.

2 B. Awerbuch and D. Peleg. Routing with polynomial communication-space trade-off. *SIAM Journal on Discrete Math.*, 5(2):151-162, 1992.

3 S. Baswana and T. Kavitha. Faster algorithms for all-pairs approximate shortest paths in undirected graphs. *SIAM Journal on Computing*, 39(7):2865-2896, 2010.

4 S. Baswana, T. Kavitha, K. Mehlhorn, S. Pettie. Additive Spanners and (α, β)-Spanners. *ACM Transactions on Algorithms*, 7(1): 5, 2010.

5 S. Baswana and S. Sen. A simple linear time algorithm for computing a $(2k-1)$-spanner of $O(n^{1+1/k})$ size in weighted graphs. In *Proc. 30th Int. Colloq. on Automata, Languages, and Programming (ICALP)*, pages 384-396, 2003.

6 S. Baswana and S. Sen. Approximate distance oracles for unweighted graphs in $O(n^2 \log n)$ time. In *Proc. 15th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 271-280, 2004.

7 E. Cohen. Fast algorithms for constructing t-spanners and paths of stretch t. In *Proc. 34th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 648-658, 1993.

8 D. Coppersmith and M. Elkin. Sparse source-wise and pair-wise distance preservers. In *Proc. 16th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 660-669, 2005.

9 L. J. Cowen. Compact routing with minimum stretch. *Journal of Algorithms*, 28:170-183, 2001.

10 L. J. Cowen and C. G. Wagner. Compact roundtrip routing in directed networks. *Journal of Algorithms*, 50(1):79-95, 2004.

11 D. Dor, S. Halperin, and U. Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740-1759, 2004.

12 M. Elkin. Computing almost shortest paths. In *ACM Transactions on Algorithms*, 1(2):283-323, 2005.

13 M. Elkin and D. Peleg. $(1+\epsilon, \beta)$-spanner construction for general graphs. *SIAM Journal on Computing*, 33(3):608-631, 2004.

14 C. Gavoille, D. Peleg, S. Perennes, and R. Raz. Distance labeling in graphs. In *Proc. 12th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 210-219, 2001.

15 S. Halperin and U. Zwick. Unpublished result, 1996.

16 D. Peleg. Proximity-preserving labeling schemes. *Journal of Graph Theory*, 33(3):167-176, 2000.

17 D. Peleg and A. A. Schaffer. Graph Spanners. *Journal of Graph Theory*, 13:99-116, 1989.

18 D. Peleg and J. D. Ullman. An optimal synchronizer for the hypercube. *SIAM Journal on Computing*, 18:740-747, 1989.

19 L. Roditty and U. Zwick. On dynamic shortest paths problems. In *Proc. 12th Annual European Symposium on Algorithms (ESA)*, pages 580-591, 2004.

20 L. Roditty, M. Thorup, and U. Zwick. Deterministic constructions of approximate distance oracles and spanners. In *Proc. 32nd Int. Colloq. on Automata, Languages, and Programming (ICALP)*, pages 261-272, 2005.

21 M. Thorup and U. Zwick. Approximate Distance Oracles. *Journal of the ACM*, 52(1):1-24, 2005.

22 M. Thorup and U. Zwick. Spanners and emulators with sublinear distance errors. In *Proc. 17th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 802-809, 2006.

Excluded vertex-minors for graphs of linear rank-width at most k

Jisu Jeong[1], O-joung Kwon[2], and Sang-il Oum*[3]

1,2,3Department of Mathematical Sciences, KAIST
291 Daehak-ro Yuseong-gu Daejeon, 305-701 South Korea

1 jjisu@kaist.ac.kr
2 ilkof@kaist.ac.kr
3 sangil@kaist.edu

Abstract

Linear rank-width is a graph width parameter, which is a variation of rank-width by restricting its tree to a caterpillar. As a corollary of known theorems, for each k, there is a finite set $\mathcal{O}_k$ of graphs such that a graph G has linear rank-width at most k if and only if no vertex-minor of G is isomorphic to a graph in $\mathcal{O}_k$. However, no attempts have been made to bound the number of graphs in $\mathcal{O}_k$ for $k \geq 2$. We construct, for each k, $2^{\Omega(3^k)}$ pairwise locally non-equivalent graphs that are excluded vertex-minors for graphs of linear rank-width at most k. Therefore the number of graphs in $\mathcal{O}_k$ is at least double exponential.

1998 ACM Subject Classification G.2.1 Combinatorics, G.2.2 Graph Theory

Keywords and phrases rank-width, linear rank-width, vertex-minor, well-quasi-ordering

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.221

1 Introduction

Linear rank-width is a width parameter of graphs motivated by rank-width of graphs by Oum and Seymour [11]. A vertex-minor relation is a graph containment relation such that rank-width and linear rank-width cannot increase when taking vertex-minors of a graph. Two graphs G, H are called *locally equivalent* if H is a vertex-minor of G and $|V(H)| = |V(G)|$. The definitions can be found in Section 2.

Oum [10] proved that for every infinite sequence G_1, $G_2, \ldots$ of graphs of bounded rank-width, there exist $i < j$ such that G_i is isomorphic to a vertex-minor of G_j. As a corollary, we immediately obtain the following theorem.

▶ **Theorem 1** (Oum [10]). *For every vertex-minor closed class $\mathcal{C}$ of graphs of bounded rank-width, there is a finite list of graphs G_1, $G_2, \ldots, G_m$ such that a graph is in $\mathcal{C}$ if and only if it does not have a vertex-minor isomorphic to G_i for some i.*

Because the rank-width of a graph is less than or equal to the linear rank-width of the graph, we deduce the following.

▶ **Corollary 2.** *For a fixed k, there is a finite set $\mathcal{O}_k$ of graphs $G_1, G_2, \ldots, G_m$ such that a graph has linear rank-width at most k if and only if it does not have a vertex-minor isomorphic to G_i for some $i \in \{1, 2, \ldots, m\}$.*

* Supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2012-0004119).

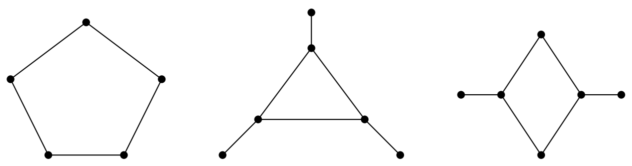


Figure 1 Graphs in $\mathcal{O}_1$.

However, Theorem 1 does not produce an explicit upper or lower bound on the number of graphs in $\mathcal{O}_k$ for Corollary 2. We aim to provide a lower bound on $|\mathcal{O}_k|$.

Our main result is the following.

▶ **Theorem 3.** *Let $k \geq 2$ be an integer. Then $|\mathcal{O}_k| \geq 2^{\Omega(3^k)}$. In other words, there are at least $2^{\Omega(3^k)}$ pairwise locally non-equivalent graphs that are vertex-minor minimal with the property that they have linear rank-width larger than k.*

When $\mathcal{C}$ is the set of all graphs having rank-width at most k, Theorem 1 implies that there are finitely many graphs $G_1, G_2, \ldots, G_m$ such that a graph has rank-width at most k if and only if it has no vertex-minor isomorphic to G_i for some i. Again Theorem 1 does not provide a lower or upper bound on m for graphs of rank-width at most k. However, for the upper bound, Oum [9] proved that $|V(G_i)| \leq (6^{k+1} - 1)/5$ for each i. No analogous result is known for linear rank-width.

Characterizing graphs of linear rank-width at most k in terms of forbidden vertex-minors seems hard. So far only 1 case is known. For $k = 1$, Adler, Farley, and Proskurowski [1] characterized the graphs of linear rank-width at most 1 by a set $\mathcal{O}_1$ of three graphs in Figure 1. A structural characterization of graphs of linear rank-width 1 was given by Ganian [6].

There have been similar results on the number of forbidden minors for various graph width parameters; for instance, path-width [12], linear-width [13], tree-width [8], branch-width [7], tree-depth [5].

The paper is organized as follows. We present the definitions of linear rank-width and vertex-minor. In Section 3, we construct a set Δ_k of graphs for every non-negative integer k, and prove that every graph in Δ_k has linear rank-width $k+1$ but every proper vertex-minor has linear rank-width at most k. Roughly speaking, $\Delta_0 = \{K_2\}$ and for $k \geq 1$, the set Δ_k consists of all graphs obtained from a disjoint union of three graphs in Δ_{k-1} by connecting them with a triangle. In Section 4, we show that no two graphs in Δ_k are locally equivalent. At last, we show that the size of Δ_k is $2^{\Omega(3^k)}$ in Section 5, and we conclude that $|\mathcal{O}_k| \geq 2^{\Omega(3^k)}$.

2 Preliminaries

In this paper, graphs have no loops and parallel edges. Let G be a graph. For $S \subseteq V(G)$, $G[S]$ denotes the subgraph of G induced on S. For $S \subseteq V(G)$, $N_G(S)$ denotes the set of vertices of $V(G) \setminus S$ adjacent to a vertex in S. And for $v \in V(G)$, we let $N_G(v) = N_G(\{v\})$. A vertex v in G is a *leaf* if $|N_G(v)| = 1$. A graph G is a *star* if G is isomorphic to $K_{1,n}$ for some $n \geq 1$.

For an $X \times Y$ matrix M and subsets $A \subseteq X$ and $B \subseteq Y$, $M[A, B]$ denotes the $A \times B$ submatrix $(m_{i,j})_{i \in A, j \in B}$ of M. If $A = B$, then $M[A] = M[A, A]$ is called a *principal submatrix* of M.

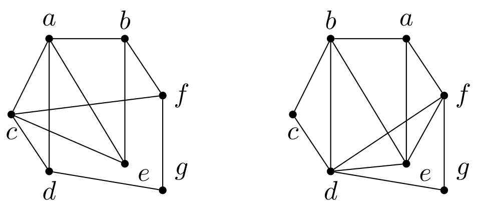


Figure 2 Pivoting an edge ab.

Vertex-minors.

The *local complementation* at a vertex v of a graph $G = (V, E)$ is an operation to obtain a graph $G * v$ from G by replacing the subgraph $G[N_G(v)]$ with the complementary subgraph of $G[N_G(v)]$. The graph obtained from G by *pivoting* an edge uv is defined by $G \wedge uv = G*u*v*u$.

To see how we obtain the resulting graph by pivoting an edge uv, let $V_1 = N_G(u) \cap N_G(v)$, $V_2 = N_G(u) \backslash N_G(v) \backslash \{v\}$, and $V_3 = N_G(v) \backslash N_G(u) \backslash \{u\}$. One can easily verify that $G \wedge uv$ is identical to the graph obtained from G by complementing adjacency of vertices between distinct sets V_i and V_j, and swapping the vertices u and v [9]. See Figure 2 for an example.

A graph H is a *vertex-minor* of G if H can be obtained from G by applying a sequence of vertex deletions and local complementations. A graph H is *locally equivalent* to G if H can be obtained from G by applying a sequence of local complementations.

A vertex-minor H of G is *elementary* if $|V(H)| = |V(G)| - 1$. A vertex-minor H of G is *proper* if $|V(H)| < |V(G)|$. A graph G is an *excluded vertex-minor* for a vertex-minor closed set $\mathcal{C}$ of graphs if $G \notin \mathcal{C}$ and $H \in \mathcal{C}$ for every proper vertex-minor H of G.

Linear rank-width.

The adjacency matrix of a graph G, which is a $(0, 1)$-matrix over the binary field, will be denoted by $A(G)$. The *cut-rank* function $\text{cutrk}_G : 2^V \to \mathbb{Z}$ of a graph $G = (V, E)$ is defined by

$$\text{cutrk}_G(X) = \text{rank}(A(G)[X, V \backslash X]).$$

A *linear layout* L of G is a sequence $(v_1, v_2, \ldots, v_{|V(G)|})$ of $V(G)$. For a linear layout L of G and $a, b \in V(G)$, we denote $a \leq_L b$ if $a = b$ or a appears before b in L. For two sequences $L_1 = (v_1, v_2, \ldots, v_n)$ and $L_2 = (w_1, w_2, \ldots, w_m)$, we define $L_1 \oplus L_2 = (v_1, v_2, \ldots, v_n, w_1, w_2, \ldots, w_m)$.

The *width* of a linear layout L in G, denoted by $\text{lrw}_L(G)$, is defined as the maximum over all $\text{cutrk}_G(\{w : w \leq_L v\})$ for $v \in V(G)$. The *linear rank-width* of G, denoted by $\text{lrw}(G)$, is the minimum width of all linear layouts of G. The next proposition shows the relation between the cut-rank function and local complementation.

▶ Proposition 4 (Oum [9]). Let G be a graph and $v \in V(G)$. Then for every $X \subseteq V(G)$,

$$\text{cutrk}_G(X) = \text{cutrk}_{G*v}(X).$$

By Proposition 4, $\text{lrw}(G) = \text{lrw}(G * v)$ for every $v \in V(G)$. Thus, we immediately obtain that if H is locally equivalent to G, then $\text{lrw}(H) = \text{lrw}(G)$. And if a graph H is a vertex-minor of a graph G, then $\text{lrw}(H) \leq \text{lrw}(G)$.

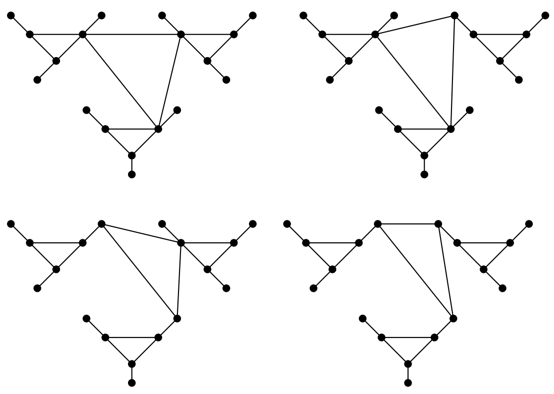


Figure 3 All graphs in Δ_2.

3 Excluded vertex-minors for graphs of bounded linear rank-width

To prove Theorem 3, for each non-negative integer k, we construct a set Δ_k of graphs such that every graph in Δ_k has linear rank-width $k+1$ but every proper vertex-minor has linear rank-width at most k.

A *delta composition* G of graphs G_1, G_2, and G_3 is a graph obtained from the disjoint union of G_1, G_2, and G_3 by adding a triangle $v_1v_2v_3$ where $v_i \in V(G_i)$ for $i = 1, 2, 3$. We call $v_1v_2v_3$ the *main triangle* of G. For a non-negative integer k, we define Δ_k as follows.

1. $\Delta_0 = \{K_2\}$.
2. For $i \geq 1$, Δ_i is the set of all delta compositions of 3 graphs in Δ_{i-1}.

The main theorem of this section is as follows.

▶ **Theorem 5.** *Let k be a non-negative integer. Every graph in Δ_k is an excluded vertex-minor for graphs of linear rank-width at most k.*

First, we prove that every graph in Δ_k has linear rank-width $k+1$.

▶ Proposition 6. Let k be a non-negative integer and $G \in \Delta_k$. Then G has linear rank-width $k+1$. Moreover, for $w \in V(G)$, there is a linear layout of G having width $k+1$ such that the first vertex of the linear layout is w.

Proof. We use induction on k. If $k = 0$, then $G = K_2$. If $V(G) = \{x, y\}$, then both (x, y) and (y, x) are linear layouts of G having width 1. Hence, the statements are true. We may assume that $k \geq 1$. Since $G \in \Delta_k$, G is a delta composition of G_1, G_2, and G_3 in Δ_{k-1}. Let $v_1v_2v_3$ be the main triangle of G such that $v_i \in V(G_i)$ for $i = 1, 2, 3$.

We first show that $\operatorname{lrw}(G) \geq k+1$. Suppose that G has linear rank-width at most k. Since $G_1 \in \Delta_{k-1}$, by induction hypothesis, G_1 has linear rank-width k. Since $\operatorname{lrw}(G) \geq \operatorname{lrw}(G_1) = k$, G has linear rank-width k. Let L be a linear layout of G having width k. And for a vertex v in G, we define $S_v = \{x \in V(G) : x \leq_L v\}$ and $T_v = V(G) \setminus S_v$.

Let a and b be the first and the last vertices in L such that $\operatorname{cutrk}_G(S_a) = \operatorname{cutrk}_G(S_b) = k$. Without loss of generality, we may assume that $\{a, b\} \subseteq V(G_2) \cup V(G_3)$. We want to show that G_1 has linear rank-width at most $k-1$. If it is true, then we obtain a contradiction because $\operatorname{lrw}(G_1) = k$. Let L_{G_1} be the subsequence of L whose elements are the vertices of G_1.

We claim that L_{G_1} is a linear layout of G_1 having width at most $k-1$. Let $v \in V(G_1)$. It is sufficient to show that $\operatorname{cutrk}_{G_1}(S_v \cap V(G_1)) \leq k-1$. Note that $v \neq a$ and $v \neq b$. If $v <_L a$ or $v >_L b$, then

$$\begin{aligned}\operatorname{cutrk}_{G_1}(S_v \cap V(G_1)) &\leq \operatorname{cutrk}_G(S_v)\\ &\leq k-1.\end{aligned}$$

So we may assume that $a <_L v <_L b$. Note that one of $S_v \cap V(G_1)$ and $T_v \cap V(G_1)$ does not have a neighbor in $G \setminus V(G_1)$ because v_1 is the unique vertex in G_1 which has a neighbor in $G \setminus V(G_1)$. And since $G[V(G_2) \cup V(G_3)]$ is connected and $a \in S_v$ and $b \notin S_v$, there is an edge u_1u_2 in $G \setminus V(G_1)$ such that $u_1 \in S_v$ and $u_2 \notin S_v$. So $A(G)[S_v \setminus V(G_1), T_v \setminus V(G_1)]$ is a non-zero matrix. Depending on whether $v_1 \in S_v \cap V(G_1)$ or $v_1 \in T_v \cap V(G_1)$,

$$\begin{aligned}\operatorname{cutrk}_G(S_v) &= \operatorname{rank}\left(\begin{array}{c|c|c} & T_v \cap V(G_1) & T_v \setminus V(G_1)\\ S_v \cap V(G_1) & * & 0\\ \hline S_v \setminus V(G_1) & * & * \end{array}\right)\\ &\geq \operatorname{rank}\left(A(G)[S_v \cap V(G_1), T_v \cap V(G_1)]\right) + \operatorname{rank}\left(A(G)[S_v \setminus V(G_1), T_v \setminus V(G_1)]\right),\end{aligned}$$

or

$$\begin{aligned}\operatorname{cutrk}_G(S_v) &= \operatorname{rank}\left(\begin{array}{c|c|c} & T_v \cap V(G_1) & T_v \setminus V(G_1)\\ S_v \cap V(G_1) & * & *\\ \hline S_v \setminus V(G_1) & 0 & * \end{array}\right)\\ &\geq \operatorname{rank}\left(A(G)[S_v \cap V(G_1), T_v \cap V(G_1)]\right) + \operatorname{rank}\left(A(G)[S_v \setminus V(G_1), T_v \setminus V(G_1)]\right),\end{aligned}$$

respectively. Thus, we have

$$\begin{aligned}\operatorname{cutrk}_{G_1}(S_v \cap V(G_1)) &= \operatorname{rank}\left(A(G)[S_v \cap V(G_1), T_v \cap V(G_1)]\right)\\ &\leq \operatorname{cutrk}_G(S_c) - \operatorname{rank}\left(A(G)[S_v \setminus V(G_1), T_v \setminus V(G_1)]\right)\\ &\leq \operatorname{cutrk}_G(S_v) - 1 \leq k-1.\end{aligned}$$

So L_{G_1} is a linear layout of G_1 having width at most $k-1$, which is a contradiction. Hence, $\operatorname{lrw}(G) \geq k+1$.

Now we show that there is a linear layout of G having width $k+1$ with a given starting vertex. Let $v \in V(G)$. Without loss of generality, we assume that $v \in V(G_1)$. By induction hypothesis, there is a linear layout L_1 of G_1 having width k such that the first vertex of L_1 is v. And, for $j = 2, 3$, there is a linear layout L_j of G_j having width k such that the first vertex of L_j is v_j. It is easy to check that $L_1 \oplus L_2 \oplus L_3$ is a linear layout of G having width at most $k+1$. Since this linear layout starts at v, we conclude the result. ◀

Of course, for $v \in V(G)$, there is also a linear layout having width $k+1$ such that the last vertex of the linear layout is v. Let $v \in V(G)$. A vertex w, $w \neq v$, in G is a *twin* of v if $N_G(w) \setminus v = N_G(v) \setminus w$. A twin w of v is a *false twin* if w is not adjacent to v. And a twin w of v is a *true twin* if w is adjacent to v.

Now we prove that every elementary vertex-minor of G in Δ_k has linear rank-width k. To prove it, we will use the following lemmata.

▶ **Lemma 7** (Bouchet [2]). *Let G be a graph, $v \in V(G)$, and H be a vertex-minor of G such that $V(G) \setminus V(H) = \{v\}$. If w is an arbitrary neighbor of v, then H is locally equivalent to either $G \setminus v$, $G * v \setminus v$, or $G \wedge vw \setminus v$.*

▶ **Lemma 8** (Oum [9]). *Let G be a graph and $vv_1, vv_2 \in E(G)$. Then $v_1v_2 \in E(G \wedge vv_1)$ and $G \wedge vv_1 \wedge v_1v_2 = G \wedge vv_2$.*

▶ **Lemma 9.** *Let k be a positive integer. Let G_1, $G_2 \in \Delta_{k-1}$, and let G_3 be a graph having linear rank-width at most $k-1$. Then every delta composition of G_1, G_2, and G_3 has linear rank-width k. Also, if a graph is obtained from the disjoint union of G_1 and G_2 by adding an edge w_1w_2 where $w_1 \in V(G_1)$ and $w_2 \in V(G_2)$, then it has linear rank-width k.*

▶ **Lemma 10.** *Let k be a non-negative integer. Let $G \in \Delta_k$, $v \in V(G)$, and H be a graph obtained from G by adding a twin w of v. Then there is a linear layout L of H having width $k+1$ such that the first vertex of L is v and the last vertex of L is w.*

We are ready to prove the main combinatorial result in this paper.

▶ Proposition 11. Let k be a non-negative integer and $G \in \Delta_k$. Then every elementary vertex-minor of G has linear rank-width k.

Proof. Note that for $v \in V(G)$ and $S \subseteq V(G)$, $\mathrm{cutrk}_{G \setminus v}(S \setminus v) \geq \mathrm{cutrk}_G(S) - 1$ because exactly one column or one row of $A(G)[S, V(G) \setminus S]$ is removed. Thus by Proposition 6, if H is an elementary vertex-minor of G, then $\mathrm{lrw}(H) \geq \mathrm{lrw}(G) - 1 = (k+1) - 1 = k$. Therefore, it is sufficient to prove that every elementary vertex-minor of G in Δ_k has linear rank-width at most k.

We use induction on k. If $k = 0$, then $G = K_2$ and every elementary vertex-minor of G is isomorphic to K_1, so it has linear rank-width 0. We assume that $k \geq 1$. Since $G \in \Delta_k$, G is a delta composition of G_1, G_2, and G_3 in Δ_{k-1}. Let $v_1v_2v_3$ be the main triangle of G such that $v_i \in V(G_i)$ for $i = 1, 2, 3$. Let H be an elementary vertex-minor of G and $V(G) \setminus V(H) = \{v\}$. By Lemma 7, for a neighbor w of v, H is locally equivalent to one of three graphs $G \setminus v$, $G * v \setminus v$, and $G \wedge vw \setminus v$. Without loss of generality, we may assume that $v \in V(G_1)$. Since $G_1 \in \Delta_{k-1}$, by induction hypothesis, $G_1 \setminus v$ has linear rank-width at most $k-1$. Thus, by Lemma 9, $G \setminus v$ has linear rank-width k. What remains to be proved is that for a neighbor w of v, $G * v \setminus v$ and $G \wedge vw \setminus v$ have linear rank-width at most k.

First, suppose that $v \neq v_1$. If $N_G(v) = \{v_1\}$, then $G * v \setminus v = G \setminus v$ and $G \wedge vv_1 \setminus v$ is isomorphic to $G \setminus v_1$. Therefore, by Lemma 9, they have linear rank-width k. If v has a neighbor w other than v_1, then

$$(G*v)[V(G_2) \cup V(G_3) \cup \{v_1\}] = (G \wedge vw)[V(G_2) \cup V(G_3) \cup \{v_1\}] = G[V(G_2) \cup V(G_3) \cup \{v_1\}].$$

Hence, both $G * v \setminus v$ and $G \wedge vw \setminus v$ are delta compositions of two graphs in Δ_{k-1} and one graph having linear rank-width at most $k-1$. Thus, by Lemma 9, they have linear rank-width k.

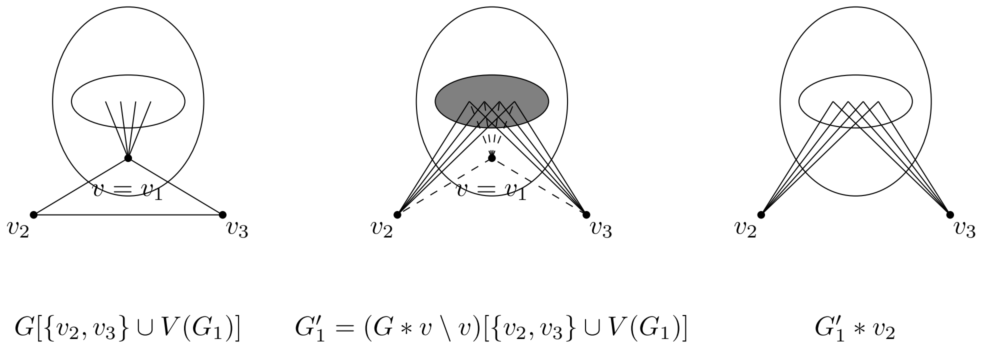

$v = v_1$
v_2
v_3
$v = v_1$
v_2
v_3
v_2
v_3
$G[\{v_2, v_3\} \cup V(G_1)]$
$G'_1 = (G * v \setminus v)[\{v_2, v_3\} \cup V(G_1)]$
$G'_1 * v_2$

Figure 4 The case $G * v \setminus v$ where $v = v_1$.

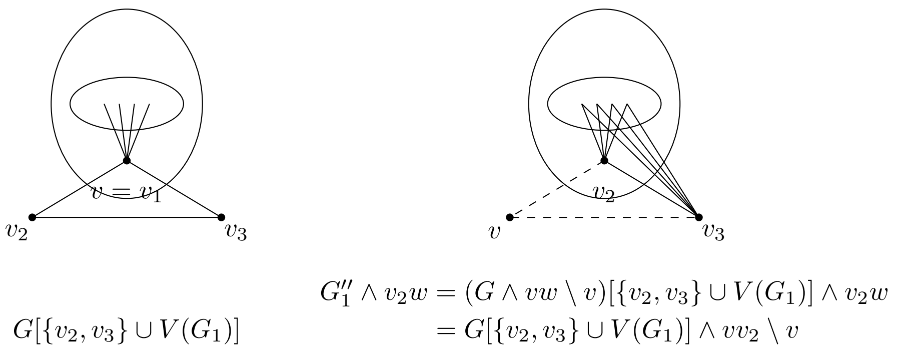

$v = v_1$
v_2
v_3
v_2
v
v_3
$G''_1 \wedge v_2w = (G \wedge vw \setminus v)[\{v_2, v_3\} \cup V(G_1)] \wedge v_2w$
$= G[\{v_2, v_3\} \cup V(G_1)] \wedge vv_2 \setminus v$
$G[\{v_2, v_3\} \cup V(G_1)]$

Figure 5 The case $G \wedge vw \setminus v$ where $v = v_1$.

Now we consider $v = v_1$. Let w be a neighbor of v in G_1. By Proposition 6, there is a linear layout L_{G_2} of G_2 having width k such that the end vertex of L_{G_2} is v_2, and there is a linear layout L_{G_3} of G_3 having width k such that the first vertex of L_{G_3} is v_3. We denote $G'_1 = (G * v \setminus v)[\{v_2, v_3\} \cup V(G_1)]$ and $G''_1 = (G \wedge vw \setminus v)[\{v_2, v_3\} \cup V(G_1)]$.

We first show that $G * v \setminus v$ has linear rank-width at most k. To prove it, we will find a linear layout L' of G'_1 having width k such that the first vertex of L' is v_2 and the last vertex of L' is v_3. In Figure 4, we can observe that $N_{G_1}(v) = N_{G'_1 * v_2}(v_2) = N_{G'_1 * v_2}(v_3)$ and $A(G)[N_{G_1}(v)] = A(G'_1 * v_2)[N_{G_1}(v)]$. Hence, the graph $G'_1 * v_2$ is isomorphic to the graph obtained from G_1 by adding a false twin of v. By Proposition 10, there is a linear layout L' of $G'_1 * v_2$ having width k such that the first vertex of L' is v_2 and the last vertex of L' is v_3. Let L_{G_1} be the sequence obtained from L' by deleting v_2 and v_3.

We show that $L = L_{G_2} \oplus L_{G_1} \oplus L_{G_3}$ is a linear layout of $G * v \setminus v$ having width at most k. If $x \in V(G_2) \cup V(G_3)$, then clearly $\mathrm{cutrk}_{G*v\setminus v}(\{y : y \le_L x\}) \le k$. If $x \in V(G_1) \setminus v$, then by Proposition 4,

$$\begin{aligned}\mathrm{cutrk}_{G*v\setminus v}(\{y : y \le_L x\}) &= \mathrm{cutrk}_{G'_1}(\{y : y \le_{L'} x\}) \\ &= \mathrm{cutrk}_{G'_1 * v_2}(\{y : y \le_{L'} x\}) \le k.\end{aligned}$$

Therefore, $G * v \setminus v$ has linear rank-width at most k.

Now we show that $G \wedge vw \setminus v$ has linear rank-width at most k. By the same argument in the previous case, it is sufficient to prove that there is a linear layout L'' of G''_1 having width k such that the first vertex is v_2 and the last vertex is v_3. We claim that $G''_1 \wedge v_2w = G[\{v_2, v_3\} \cup V(G_1)] \wedge vv_2 \setminus v$. Note that

$$\begin{aligned}G''_1 \wedge v_2w &= (G \wedge vw \setminus v)[\{v_2, v_3\} \cup V(G_1)] \wedge v_2w \\ &= G[\{v_2, v_3\} \cup V(G_1)] \wedge vw \setminus v \wedge v_2w \\ &= G[\{v_2, v_3\} \cup V(G_1)] \wedge vw \wedge v_2w \setminus v.\end{aligned}$$

And by Lemma 8,

$$G[\{v_2, v_3\} \cup V(G_1)] \wedge vw \wedge v_2w \setminus v = G[\{v_2, v_3\} \cup V(G_1)] \wedge vv_2 \setminus v.$$

In Figure 5, we can observe that $G''_1 \wedge v_2w$ is isomorphic to the graph obtained from G_1 by adding a true twin of v. Thus, by Proposition 10, there is a linear layout L'' of $G''_1 \wedge v_2w$ having width k such that the first vertex of L'' is v_2 and the last vertex of L'' is v_3. Also, for $x \in V(G_1) \setminus v$,

$$\mathrm{cutrk}_{G''_1}(\{y : y \le_{L''} x\}) = \mathrm{cutrk}_{G''_1 \wedge vv_2}(\{y : y \le_{L''} x\}) \le k.$$

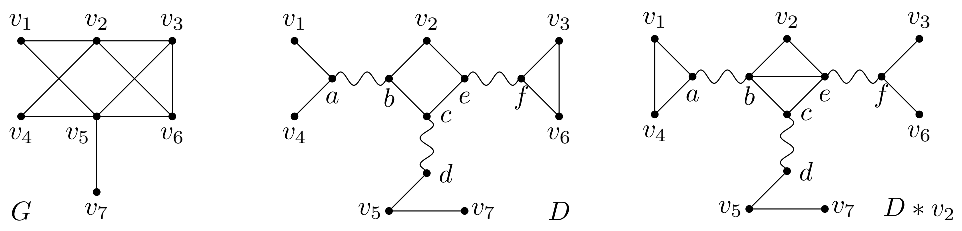


Figure 6 A split-decomposition D of a graph G, and $D * v_2$. The marked edges of D are depicted as wavy edges, and the desendants of the vertex v_2 in D is a and f. Note that $D * v_2$ is a split decomposition of $G * v_2$.

Therefore, we conclude that $G \wedge vw \setminus v$ has linear rank-width at most k. ◀

Proof of Theorem 5. Let $G \in \Delta_k$. By Proposition 6, G has linear rank-width $k + 1$. And by Proposition 11, every elementary vertex-minor of G has linear rank-width k. So every proper vertex-minor of G has linear rank-width at most k. Therefore, G is an excluded vertex-minor for graphs of linear rank-width at most k. ◀

4 No two graphs in Δ_k are locally equivalent.

In this section, we show that no two graphs in Δ_k are locally equivalent.

▶ **Theorem 12.** *Let k be a non-negative integer and $G, H \in \Delta_k$. If G and H are locally equivalent, then G and H are isomorphic.*

To prove it, we will use the canonical split-decompositions of graphs in Δ_k.

Split-decomposition.

Let G be a graph. A partition (A, B) of $V(G)$ is a *split* if $|A| \geq 2$, $|B| \geq 2$, and for every $v \in N_G(B)$ and $w \in N_G(A)$, $vw \in E(G)$. If G has no split and $|V(G)| \geq 5$, then we call G a *prime graph.* If G has a split (A, B), then we define a graph G', called a *simple decomposition* of G, as the graph obtained from G by deleting all edges between $N_G(A)$ and $N_G(B)$, and adding two vertices w_1,w_2 and adding edges $\{w_1w_2\} \cup \{vw_1 : v \in N_G(B)\} \cup \{w_2v : v \in N_G(A)\}$. We call w_1w_2 a *marked edge* of G'. A graph is a *marked graph* if it has marked edges, and for a marked graph D, we define $M(D)$ as the set of marked edges of D. A *split-decomposition* of G is recursively defined to be either G or a marked graph obtained from a split-decomposition D by replacing a component H of $D \setminus M(D)$ with a simple decomposition of H. Two components C_1 and C_2 of $D \setminus M(D)$ are *neighbors* if there exist $v_1 \in V(C_1)$, $v_2 \in V(C_2)$ such that $v_1v_2 \in M(D)$. A split-decompositon D of a graph is *canonical* if it satisfies the following:

(i) each component of $D \setminus M(D)$ is either a prime graph or a star or a complete graph,
(ii) no two complete components are neighbors,
(iii) if two star components are neighbors, then two ends of the marked edge are both centers or both leaves of each components.

Two split-decompositions D_1 and D_2 of a graph G are *equivalent* if there is a graph isomorphism f from D_1 to D_2 such that f preserves the marked edges and $f|_{V(G)}$ is an identity function. We need the following result.

▶ **Lemma 13** (Cunningham [4])**.** *Canonical split-decompositions of a graph are equivalent.*

Let D be the canonical split-decomposition of G and $C(D) = \{C_1, C_2, \dots, C_n\}$ be the components of $D \setminus M(D)$. A tree T_G is a *canonical tree* of G if $V(T_G) = \{v_{C_1}, v_{C_2}, \dots, v_{C_n}\}$ and v_{C_i} is adjacent to v_{C_j} if and only if two components C_i and C_j are neighbors in D. We call f the *canonical map* from T_G to D if it is the bijection from $V(T_G)$ to $C(D)$ such that $f(v_{C_k}) = C_k$.

For $v \in V(G) \subseteq V(D)$, a vertex w in D is a *descendant* of v if either $w = v$ or w is the end of a path starting from v, whose successive edges are alternatively non-marked and marked edges, and the last edge is marked. Note that each component of $D \setminus M(D)$ has at most 1 descendant of a vertex because every marked edge in D is a cut-edge. For $v \in V(G)$, we define $D * v$ as the marked graph obtained from D by replacing each component H of $D \setminus M(D)$ having a descendant w of v by $H * w$.

▶ **Lemma 14** (Bouchet [3])**.** *If D is a canonical split-decomposition of a graph G and $v \in V(G)$, then $D * v$ is a canonical split-decomposition of the graph $G * v$.*

By Lemma 14, if G and H are locally equivalent, then G and H have isomorphic canonical trees. Hence, it is sufficient to prove that for $G, H \in \Delta_k$, if G and H have isomorphic canonical trees, then G is isomorphic to H. To show this, we explicitly describe the canonical decompositions of graphs in Δ_k.

Clearly, K_2 has itself as a canonical split-decomposition. Let $k \geq 1$ and $G \in \Delta_k$. Note that for a non-leaf vertex v in G, v is incident with exactly one cut-edge and meets at least one triangles. For a non-leaf vertex v in G, let l_v be the star on the vertex set $V(l_v) = \{v, a^v, b^v_{C_1}, b^v_{C_2}, \dots, b^v_{C_m}\}$ with the center v, where v is incident with a cut-edge e and meets trianges $C_1, C_2, \dots, C_m$. And for each triangle C in G, let t_C be the triangle on the vertex set $\{d^a_C, d^b_C, d^c_C\}$ where $V(C) = \{a, b, c\}$. We define the graph D_G as the graph obtained from the disjoint union of all graphs in $\{l_v : v \text{ is a non-leaf vertex in } G\} \cup \{t_C : C \text{ is a triangle in } G\}$ by adding the marked edge set $M(D_G)$ which consists of

(i) $b^v_C d^v_C$ if v meets a triangle C,
(ii) $a^v a^w$ if vw is a cut-edge of G and both v and w are not leaves of G.

We can verify that the marked graph D_G with $M(D_G)$ of the third graph G in Figure 3 is the first graph in Figure 7. In general, we can show that for $G \in \Delta_k$, D_G with the marked edge set $M(D_G)$ is indeed a canonical split-decomposition of G.

▶ **Lemma 15.** *Let k be a non-negative integer and $G \in \Delta_k$. The graph D_G is a canonical split-decomposition of G with the set $M(D_G)$ of marked edges.*

We can observe the following.

▶ **Lemma 16.** *Let k be a non-negative integer and $G \in \Delta_k$. Let T_G be a canonical tree of G and f be the canonical map from T_G to D_G. Let B be the set of vertices of T_G mapped by f to a complete graph. Then the following are true.*

(i) If $v \in B$, then $N_{T_G}(v) \cap B = \emptyset$ and $|N_{T_G}(v)| = 3$.
(ii) Every component of $T_G[V(T_G) \setminus B]$ has at most 2 vertices.
(iii) If $w \in V(T_G) \setminus B$, then the component $f(w)$ is a star, and the center of $f(w)$ is a non-leaf vertex in G, say u. Suppose that u meets m triangles in G. Then u is adjacent with $m + 1$ vertices in $f(w)$.

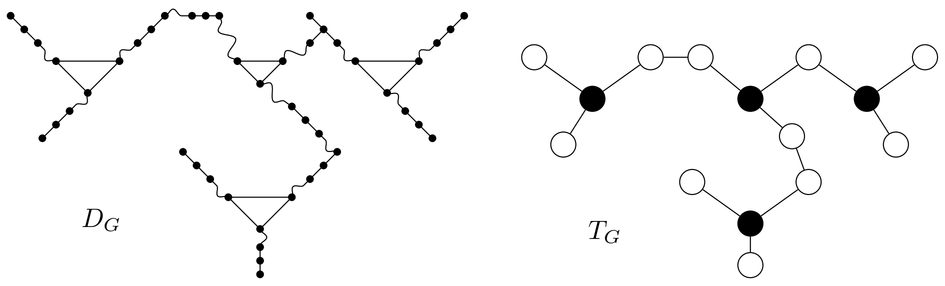


Figure 7 The canonical split-decomposition D_G and the canonical tree T_G of the third graph G in Figure 3. The black vertices in T_G are the vertices mapped by the canonical map to a triangle of D_G.

▶ Proposition 17. Let k be a non-negative integer and $G, H \in \Delta_k$. If G, H have isomorphic canonical trees, then G is isomorphic to H.

Proof. Let T be a canonical tree of both G and H. Let f_G be the canonical map from T to D_G, and let B_G be the set of vertices mapped by f_G to a complete graph of D_G. Similarly, let f_H be the canonical map from T to D_H, and let B_H be the set of vertices mapped by f_H to a complete graph in D_H.

We first show that $B_G = B_H$. Suppose that $B_G \neq B_H$. Since G and H have the same number of triangles, $|B_G| = |B_H|$. So we can choose $v_1 \in B_G \setminus B_H$ and a maximal path $P = v_1v_2 \dots v_n$ in T such that

(i) P contains vertices from B_G and from $V(T) \setminus B_G$, alternatively, and
(ii) P also contains vertices from $V(T) \setminus B_H$ and from B_H, alternatively.

Suppose v_n is not a leaf. By the symmetry, we assume that $v_n \in B_G$ and $v_n \in V(T) \setminus B_H$. Since $v_n \in B_G$, by Lemma 16, v_n has 3 neighbors in T, which are contained in $V(T) \setminus B_G$. And since $v_n \in V(T) \setminus B_H$, by Lemma 16, v_n has at most 1 neighbor of $V(T) \setminus B_H$. Hence, there exists a vertex in $(N_T(v_n) \setminus V(P)) \cap B_H$, say v_{n+1}. Thus, $v_{n+1} \in V(T) \setminus B_G$ and $v_{n+1} \in B_H$, and $v_1v_2, \dots, v_nv_{n+1}$ is also a path in T satisfying (i) and (ii). It contradicts to the maximality of P. Thus, v_n is a leaf in T. But if v_n is a leaf in T, neither $f_G(v_n)$ nor $f_H(v_n)$ is a triangle, so it is a contradiction. Therefore, $B_G = B_H$, and we call this set B.

Clearly, for $v \in B$, $f_G(v)$ and $f_H(v)$ are triangles. And by Lemma 16, for $v \in V(T) \setminus B$, the components $f_G(v)$ and $f_H(v)$ are uniquely determined by the neighbors of v in T_G. Therefore, the graphs D_G and D_H are isomorphic, and G is isomorphic to H. ◀

Proof of Theorem 12. Since G and H are locally equivalent, there is a sequence $v_1, v_2, \dots v_m$ of $V(G)$ such that $G * v_1 * v_2 \dots * v_m = H$. By Lemma 14, G and $G * v_1 * v_2 \dots * v_m$ have isomorphic canonical trees. And since $G * v_1 * v_2 \dots * v_m = H$, by Lemma 13, $G * v_1 * v_2 \dots * v_m$ and H have isomorphic canonical trees. Thus G and H have isomorphic canonical trees. Since $G, H \in \Delta_k$, by Proposition 17, G is isomorphic to H. ◀

5 The size of Δ_k is $2^{\Omega(3^k)}$

In this section, we determine the number of graphs in Δ_k for each non-negative integer k. The main theorem of this section is as follows.

▶ **Theorem 18.** *Let $k \geq 2$ be an integer. The size of Δ_k is $2^{\Omega(3^k)}$.*

For graphs G, G' and $v \in V(G)$ and $v' \in V(G')$, two pairs (G, v) and (G', v') are *isomorphic* if there exists a graph isomorphism ϕ from G to G' such that $\phi(v) = v'$. To prove Theorem 18, for a positive integer k, we partition Δ_k into A_k, B_k, and C_k as follows:

(i) $G \in A_k$ if (G_1, v_1), (G_2, v_2), and (G_3, v_3) are isomorphic,
(ii) $G \in B_k$ if only two of (G_1, v_1), (G_2, v_2), (G_3, v_3) are isomorphic,
(iii) $G \in C_k$ otherwise,

where G is a delta composition of G_1, G_2, and G_3 in Δ_{k-1}, and $v_1v_2v_3$ is the main triangle of G such that $v_i \in V(G_i)$ for $i = 1, 2, 3$. If p_k is the number of non-isomorphic pairs (G, v) where $G \in \Delta_k$ and $v \in V(G)$, we can easily verify that

$$|A_k| = p_{k-1}, \quad |B_k| = p_{k-1}(p_{k-1} - 1), \quad |C_k| = \frac{1}{6}p_{k-1}(p_{k-1} - 1)(p_{k-1} - 2).$$

We will give a lower bound of p_k from $|A_k|$, $|B_k|$, $|C_k|$, and obtain a recurrence relation.

For a graph G and $v, w \in V(G)$, we denote $v \simeq_G w$ if (G, v) and (G, w) are isomorphic. We consider the equivalent classes $V(G)/\simeq_G$. We denote $[v]$ as an element of $V(G)/\simeq_G$. For a non-negative integer k, let $P_k = \{(G, [v]) : G \in \Delta_k, [v] \in V(G)/\simeq_G\}$ and $p_k = |P_k|$. Then p_k is exactly the number of all non-isomorphic pairs (G, v) where $G \in \Delta_k$ and $v \in V(G)$. It is obvious that $p_0 = 1$, $p_1 = 2$. And we can see that $p_2 = 24$ in Figure 3. We need the following lemma.

▶ **Lemma 19.** *Let k be a positive integer and $G \in \Delta_k$.*

1. *If $G \in A_k$, then $|V(G)/\simeq_G| \geq 2^k$.*
2. *If $G \in B_k$, then $|V(G)/\simeq_G| \geq 2 \cdot 2^k$.*
3. *If $G \in C_k$, then $|V(G)/\simeq_G| \geq 3 \cdot 2^k$.*

Proof of Theorem 18. By Lemma 19,

$$p_k = \sum_{G \in A_k \cup B_k \cup C_k} |V(G)/\simeq_G| \geq 2^k|A_k| + 2 \cdot 2^k|B_k| + 3 \cdot 2^k|C_k|.$$

Since $|A_k| = p_{k-1}$, $|B_k| = p_{k-1}(p_{k-1} - 1)$ and $|C_k| = \frac{1}{6}p_{k-1}(p_{k-1} - 1)(p_{k-1} - 2)$, we obtain the following recurrence relation,

$$\begin{aligned} |A_{k+1}| = p_k &\geq 2^k|A_k| + 2 \cdot 2^k|B_k| + 3 \cdot 2^k|C_k| \\ &\geq 2^{k-1}|A_k|^3 \end{aligned}$$

and $|A_2| = 2$.

This means $|A_k| = 2^{\Omega(3^k)}$ for $k \geq 3$. Because $|\Delta_2| = 4$ and $|\Delta_k| \geq |A_k| = 2^{\Omega(3^k)}$ for $k \geq 3$, we conclude that $|\Delta_k| = 2^{\Omega(3^k)}$ for $k \geq 2$. ◀

Proof of Theorem 3. By Theorems 5 and 12, $|\mathcal{O}_k| \geq |\Delta_k|$. And by Theorem 18, $|\Delta_k| \geq 2^{\Omega(3^k)}$. Therefore, $|\mathcal{O}_k| \geq 2^{\Omega(3^k)}$. ◀

References

1 Isolde Adler, Arthur M. Farley, and Andrzej Proskurowski. Obstructions for linear rankwidth at most 1. *arXiv:1106.2533*, 2011.

2 André Bouchet. Graphic presentations of isotropic systems. *J. Combin. Theory Ser. B*, 45(1):58–76, 1988.

3 André Bouchet. Transforming trees by successive local complementations. *J. Graph Theory*, 12(2):195–207, 1988.

4 William H. Cunningham. Decomposition of directed graphs. *SIAM J. Algebraic Discrete Methods*, 3(2):214–228, 1982.

5 Zdeněk Dvořák, Archontia C. Giannopoulou, and Dimitrios M. Thilikos. Forbidden graphs for tree-depth. *European J. Combin.*, 33(5):969–979, 2012.

6 Robert Ganian. Thread graphs, linear rank-width and their algorithmic applications. In *Combinatorial Algorithms*, volume 6460 of *Lecture Notes in Comput. Sci.*, pages 38–42. Springer, 2011.

7 J. F. Geelen, A. M. H. Gerards, N. Robertson, and G. P. Whittle. On the excluded minors for the matroids of branch-width k. *J. Combin. Theory Ser. B*, 88(2):261–265, 2003.

8 Arvind Gupta, Damon Kaller, and Thomas Shermer. On the complements of partial k-trees. In *Automata, languages and programming (Prague, 1999)*, volume 1644 of *Lecture Notes in Comput. Sci.*, pages 382–391. Springer, Berlin, 1999.

9 Sang-il Oum. Rank-width and vertex-minors. *J. Combin. Theory Ser. B*, 95(1):79–100, 2005.

10 Sang-il Oum. Rank-width and well-quasi-ordering. *SIAM J. Discrete Math.*, 22(2):666–682, 2008.

11 Sang-il Oum and Paul Seymour. Approximating clique-width and branch-width. *J. Combin. Theory Ser. B*, 96(4):514–528, 2006.

12 Atsushi Takahashi, Shuichi Ueno, and Yoji Kajitani. Minimal acyclic forbidden minors for the family of graphs with bounded path-width. *Discrete Math.*, 127(1-3):293–304, 1994. Graph theory and applications (Hakone, 1990).

13 Dimitrios M. Thilikos. Algorithms and obstructions for linear-width and related search parameters. *Discrete Appl. Math.*, 105(1-3):239–271, 2000.

Recompression: a simple and powerful technique for word equations*

Artur Jeż

Max Planck Institute für Informatik,
Campus E1 4, DE-66123 Saarbrücken, Germany
Institute of Computer Science, University of Wrocław,
ul. Joliot-Curie 15, PL-50-383 Wrocław, Poland
aje@cs.uni.wroc.pl

Abstract

We present an application of a local recompression technique, previously developed by the author in the context of compressed membership problems and compressed pattern matching, to word equations. The technique is based on local modification of variables (replacing X by aX or Xa) and replacement of pairs of letters appearing in the equation by a 'fresh' letter, which can be seen as a bottom-up *compression* of the solution of the given word equation, to be more specific, building an SLP (Straight-Line Programme) for the solution of the word equation.

Using this technique we give new self-contained proofs of many known results for word equations: the presented nondeterministic algorithm runs in $\mathcal{O}(n \log n)$ space and in time polynomial in $\log N$ and n, where N is the size of the length-minimal solution of the word equation. It can be easily generalised to a generator of all solutions of the word equation. A further analysis of the algorithm yields a doubly exponential upper bound on the size of the length-minimal solution. The presented algorithm does not use exponential bound on the exponent of periodicity. Conversely, the analysis of the algorithm yields a new proof of the exponential bound on exponent of periodicity. For $\mathcal{O}(1)$ variables with arbitrary many appearances it works in linear space.

1998 ACM Subject Classification F.4.3 Formal Languages: Decision problems, F.4.2 Grammars and Other Rewriting Systems, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Word equations, exponent of periodicity, string unification

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.233

1 Introduction

The problem of *word equations* is one of the most intriguing in computer science: given words U and V, consisting of letters (from Γ) and variables (from $\mathcal{X}$) we are to check the *satisfiability*, i.e. decide, whether there is a substitution for variables that turns this formal equation into an equality of strings of letters. It is useful to think of a solution S as a homomorphism $S : \Gamma \cup \mathcal{X} \to \Gamma^*$, which is an identity on Γ. In the more general problem of *solving* the equation, we are to give representation of all solutions of the equation.

The satisfiability problem was solved by Makanin [10] (a more accessible presentation is due to Diekert [2]). The proposed algorithm MakSAT transforms equations and large part of Makanin's work consists of proving that this procedure in fact terminates. While terminating, MakSAT complexity is high and it was gradually improved: by Jaffar and independently Schulz to 4-NEXPTIME [4, 17] Kościelski and Pacholski to 3-NEXPTIME [7], by Diekert to

* This work was partially supported by NCN grant number DEC-2011/01/D/ST6/07164, 2011–2014.

2-EXPSPACE (unpublished, see notes in [2]) and by Gutiérrez to EXPSPACE [3]. On the other hand, only a (simple) NP-hardness is known and it is widely believed that indeed this problem is in NP.

One of the key factors in the proof of termination, as well in later estimations of the complexity of the algorithm, was the upper bound on *exponent of periodicity* of the solution: the exponent of periodicity of a word w is the largest p such that $w = w_1 u^p w_2$ for some $u \neq \epsilon$. The original proof of Makanin gave a doubly exponential bound on the exponent of periodicity of any length-minimal solution of word equations. Later it was shown by Kościelski and Pacholski that it is at most exponential and that this bound is tight [7].

A major independent step in the field was done by Plandowski and Rytter [15], who for the first time applied the *compression* to the solutions of the word equations. They showed that LZ-compressed representation of length-minimal (of size N) is $\mathcal{O}(\mathsf{poly}(\log N, n))$-size. This yielded a new, very simple to both state and analyse, algorithm for word equations satisfiability, which works in (nondeterministic) polynomial in terms of $\log N$ and n. Unfortunately, at that time the only bound on N followed from the original Makanin's algorithm, and it was triply exponential; this gave a 2-NEXPTIME algorithm.

Later, Plandowski gave a doubly-exponential upper bound on the size of the minimal solution [12], which immediately yielded a NEXPTIME algorithm PlaSAT2EXP. This upper bound was obtained by an analysis of the minimal solution using so-called $\mathcal{D}$-factorisations.

Soon after an algorithm PlaSAT with a PSPACE upper-bound was given by Plandowski [13]. This algorithm starts with a trivial equation $e = e$ and has a set of operations that can be performed on the equation. The rewriting rules are simple and the algorithm is easy to understand, moreover it is obvious that they preserve satisfiability. However, the proof of completeness (i.e. that they properly generate all satisfiable equations) is involved. It is based on usage of exponential expressions, which can be seen as a very simple compression, and on indexed factorisations of words, which extend $\mathcal{D}$-factorisations.

All mentioned algorithms check satisfiability and can be modified to return some solution of the word equation, but they do not *solve* it in the sense that they do not provide a representation of all solutions. This was fully resolved by Plandowski [14], who gave an algorithm PlaSolve, which runs in PSPACEand generates a compact representation of all finite solutions of a word equation. This algorithm uses an improved version of PlaSAT, the PlaSATimp, as subprocedure. The representation of the solutions is a directed multigraph, whose nodes are labelled with expressions and edges define substitutions for constants and variables. Such representation reduces many properties of word equations to reachability in graphs (which were exponentially larger), for instance the problem of finiteness of set of solutions is shown to be in PSPACE.

Our contribution

We present an application of a simple technique of *local recompression* developed by the author and successfully applied to problems related with compressed data [5, 6]. The idea of the technique is easily explained in terms of solutions of the equations rather than the equations themselves: consider a solution $S(U) = S(V)$ of the equation $U = V$. In one phase we first list all pairs of different letters ab that appear as substrings in $S(U)$ and $S(V)$. For a fixed pair ab of this kind we greedily replace all appearances of ab in $S(U)$ and $S(V)$ by a 'fresh' letter c. (A slightly more complicated action is performed for pairs aa, for now we ignore this case to streamline the presentation of the main idea). There are possible conflicts between such replacements for different types of pairs (consider string aba, in which we try to replace both pairs ab and ba), we resolve them by introducing some arbitrary order on types of pairs and performing the replacement for one type of pair at a time, according to the order. When all such pairs are replaced, we obtain another equal pair of strings $S'(U')$ and $S'(V')$ (note that the equation $U = V$ may have changed, say into $U' = V'$). Then we iterate the process. In each phase the strings are shortened by a constant factor, hence after $\mathcal{O}(\log N)$ phases we obtain constant-length strings. The original equation is solvable if and only if the obtained strings are the same.

The problematic part is that the operations are performed on the solutions, which can be large. If we simply guess the solution and then perform the compressions, the running time is polynomial in N. Instead we perform the compression directly on the equation (the *recompression*): the pairs ab appearing in the solution are identified using only the equation and the compression of the solution is two-fold: the pairs ab from U and V are replaced explicitly and the pairs fully within some $S(X)$ are replaced implicitly, by changing S (which is not stored). However, not all pairs of letters can be compressed in this way, as some of them appear on the 'crossing' between a variable and a constant: consider for instance $S(X) = ab$, a string of symbols Xc and a compression of a pair bc. This is resolved by *local decompression*: when trying to compress the pair bc in the example above we first replace X by Xb (implicitly changing $S(X)$ from ab to a), obtaining the string of symbols Xbc, in which the pair bc can be easily compressed.

▶ **Example 1.** Consider an equation $aXca = abYa$ with a solution $S(X) = baba$ and $S(Y) = abac$. In the first phase, the algorithm wants to compress the pairs ab, ca, ac, ba, say in this order. To compress ab, it replaces X with bX, thus changing the substitution into $S(X) = aba$. After compression we obtain equation $a'Xca = a'Ya$. Notice, that this implicitly changed to solution into $S(X) = a'a$ and $S(Y) = a'ac$. To compress ca (into c'), we replace Y by Yc, thus implicitly changing the substitution into $S(Y) = a'a$. Then, we obtain the equation $a'Xc' = a'Yc'$ with a solution $S(X) = a'a$ and $S(Y) = a'a$. The remaining pairs no longer appear in the equations, and so we can proceed to the next phase.

Using the technique of local recompression we give a (nondeterministic) algorithm for testing satisfiability of word equations that works in time $\mathcal{O}(\log N \mathsf{poly}(n))$ and in $\mathcal{O}(n \log n)$ space; PlaSAT stored equation of quadratic length and used cubic space for auxiliary computation. Furthermore, for $\mathcal{O}(1)$ variables a more careful analysis yields that the space consumption (calculated in bits) is $\mathcal{O}(n)$, thus showing that this case is context-sensitive.

The presented algorithm and its analysis are stand-alone, as they do not assume any (non-trivial) properties of the solutions of word equations. To the contrary, the presented algorithm supplies an easy proof of doubly-exponential upper bound of Plandowski [12] on lengths of length-minimal solutions as well as giving a new proof of exponential exponent of periodicity (though slightly weaker than the one presented by Kościelski and Pacholski [7]).

The presented method can be used as a subprocedure in an algorithm generating a representation of all solutions, similarly as PlaSATimp in PlaSolve. The representation provided by our algorithm is similar to representation provided by PlaSolve, i.e. a directed multigraph with edges representing substitutions.

Comparison with previous approaches to word equations

The presented method and the obtained algorithm is independent from all previously known algorithms for word equations, i.e. from original MakSAT and its variants, from PlaRytSAT (and its variant PlaSAT2EXP), from PSPACE algorithm PlaSAT as well as its modification PlaSATimp. It can be somehow compared with the LZ-based PlaRytSAT [15]. The key difference is that Plandowski and Rytter showed that a length-minimal solution has a short

LZ-representation and then explicitly guessed and verified it. Thus their solution is 'global' and based on solutions' properties. The novelty of the here proposed method is that it does not use properties of the solutions and that it is very 'local', as it does not try to build the solution in one go and modifies the equations and variables locally.

The presented algorithm uses a limited variant of exponent of periodicity, in which the strings in question are repetitions of the same letter. In such a case the bound on such restricted exponent of periodicity easily reduces to a bound on minimal solutions of a system of linear Diophantine equations, which are well known. This again makes the presented algorithm somehow similar to PlaRytSAT, which does not use the exponent of periodicity bound.

Related techniques

While the presented method of recompression is relatively new, some of its ideas and inspirations go quite back. It was developed in order to deal with fully compressed membership problem for NFA and the previous work on this topic by Mathissen and Lohrey [9] already implemented the idea of replacing strings with fresh letters as well as modifications of the instance so that this is possible. Furthermore they treated maximal blocks of a single letter in a proper way. However, the replacement was not iterated, and the newly introduced letters could not be further compressed.

The idea of replacing short strings by a fresh letter and iterating this procedure was used by Mehlhorn et. al [11] in their work on data structure for equality tests for dynamic strings (cf. also an improved implementation of a similar data structure by Alstrup, Brodal and Rauhe [1]). In particular their method can be straightforwardly applied to equality testing for SLPs, yielding a nearly quadratic algorithm (as observed by Gawrychowski). However, the inside technical details of the construction make it problematic to extend: while this method can be used to build 'canonical' SLPs for the text and the pattern, there is no apparent way to control how these SLPs look like and how do they encode the strings.

A similar technique, based on replacement of pairs and blocks of the same letter was proposed by Sakamoto [16] in the context of constructing a smallest SLP generating a given word. His algorithm was inspired by a practical grammar-based compression algorithm RePair [8]. It possessed the important features of the method: iterated replacement, and ignoring letters recently introduced. However, the analysis that stressed the modification of the variables (nonterminals, in there considered case of grammars) was not introduced and it was done in a different way.

2 Main notions and techniques: local compression

By Γ we denote the set of letters appearing in the equation $U = V$ or are used for representation of compressed strings, $\mathcal{X}$ denotes a set of variables. The equation is written as $U = V$, where $U, V \in (\Gamma \cup \mathcal{X})^*$. By $|U|$, $|V|$ we denote the length of U and V, n denotes the length of the input equation, n_v is the number of appearances of variables in the input.

A *substitution* is a morphism $S : \mathcal{X} \cup \Gamma \to \Gamma^*$, such that $S(a) = a$ for every $a \in \Gamma$, substitution is naturally extended to $(\mathcal{X} \cup \Gamma)^*$. A *solution* of an equation $U = V$ is a substitution S, such that $S(U) = S(V)$; a solution S is a *length-minimal*, if for every solution S' it holds that $|S(U)| \leq |S'(U)|$.

Operations

In essence, the presented technique is based on performing two operations on $S(U)$ and $S(V)$, the first one is *pair compression of* ab: For two different letters ab appearing in $S(U)$ replace each of ab in $S(U)$ and $S(V)$ by a fresh letter c.

The compression of pair aa is ambiguous, we compress maximal blocks instead: For a letter $a \in \Gamma$ we say that a^ℓ is a a's *maximal block* of length ℓ (for a solution S), if a^ℓ appears in $S(U)$ and this appearance cannot be extended by a letter a to the right, neither to the left. We refer to *a's ℓ-block* for shortness. The second operation is *block compression for a*: For a letter a appearing in $S(U)$ and for each maximal block a^ℓ replace all a^ℓs in $S(U)$ and $S(V)$ by a fresh letter a_ℓ.

The lengths of the maximal blocks can be upper bounded using the well-known exponential bound on exponent of periodicity:

▶ **Lemma 2** (Exponent of periodicity bound [7]). *If solution S is length-minimal and w^ℓ for $w \neq \epsilon$ is a substring of $S(U)$, then $\ell \leq 2^{cn}$ for some constant $0 < c < 2$.*

▶ Remark. WordEqSat introduces new letters to the instance, replacing pairs of letters or maximal blocks of one letter. We insist that these new symbols are called and treated as letters. On the other hand, we can think of them as non-terminals of a context-free grammar: if c replaced ab, then this corresponds to a production $c \to ab$, similarly, $a_\ell \to a^\ell$. In this way we can think that WordEqSat builds a context-free grammar generating $S(U)$ as a unique word in the language.

Types of pairs and blocks

Both pair compression and block compression shorten $S(U)$ (and $S(V)$). On the other hand, sometimes it is hard to perform these operations: for instance, if we are to compress a pair ab and aX appears in U, moreover, $S(X)$ begins with b, then the compression is problematic, as we need to somehow modify $S(X)$. The following definition allows distinguishing between pairs (blocks) that are easy to compress and those that are not.

▶ **Definition 3** (cf. [5, 6]). Given an equation $U = V$ and a substitution S and a substring $u \in \Gamma^+$ of $S(U)$ (or $S(V)$) we say that this appearance of u is *explicit*, if it comes from substring u of U (or V, respectively); *implicit*, if it comes (wholly) from $S(X)$ for some variable X; *crossing* otherwise. A string u is *crossing* (with respect to S) if it has a crossing appearance and non-crossing otherwise.

We say that a pair of ab is a *crossing pair* (with respect to S), if ab has a crossing appearance. Otherwise, a pair is *non-crossing* (with respect to S). Unless explicitly stated, we consider crossing/non-crossing pairs ab in which $a \neq b$. Similarly, a letter $a \in \Gamma$ has a *crossing block* (with respect to S), if there is a block of a which has a crossing appearance.

Compression of noncrossing pairs is easy, so is block compression when a has no crossing block. In other cases, the compression seems difficult.

We say that a^ℓ is *visible* in S, if there is an appearance of the a's ℓ-block that is explicit or crossing or it is a prefix or suffix of some $S(X)$.

▶ **Lemma 4** (cf. [15, Lemma 6]). *Let S be a length-minimal solution of $U = V$.*

- *If ab is a substring of $S(U)$, where $a \neq b$, then ab is an explicit pair or a crossing pair.*
- *If a^k is a maximal block in $S(U)$ then a has an explicit appearance in U or V and there is a visible appearance of a^k.*

Compression of noncrossing pairs and blocks

Intuitively, when ab is non-crossing, each of its appearance in $S(U)$ is either explicit or implicit. Thus, to perform the pair compression of ab on $S(U)$ it is enough to separately replace each explicit pair ab in U and change each ab in $S(X)$ for each variable X. The latter is of course done implicitly (as $S(X)$ is not written down anywhere).

Algorithm 1 PairCompNCr(a, b) Pair compression for a non-crossing pair

```
1: let $c \in \Gamma$ be an unused letter
2: replace each explicit $ab$ in $U$ and $V$ by $c$
```

Similarly when none block of a has a crossing appearance, the a's blocks compression consists simply of replacing explicit a blocks.

Algorithm 2 BlockCompNCr(a) Block compression for a letter a with no crossing block

```
1: for each explicit $a$'s $\ell$-block appearing in $U$ or $V$ do
2:     let $a_\ell \in \Gamma$ be an unused letter
3:     replace every explicit $a$'s $\ell$-block appearing in $U$ or $V$ by $a_\ell$
```

Preserving satisfiability and unsatisfiability

We say that a nondeterministic procedure *preserves unsatisfiability*, when given a unsatisfiable word equation $U = V$ it cannot transform it to a satisfiable one, regardless of the nondeterministic choices; such a procedure *preserves satisfiability*, if given a satisfiable equation $U = V$ for some nondeterministic choices it returns a satisfiable equation $U' = V'$.

A procedure that preserves satisfiability *implements pair compression of a pair ab* for a solution S of an equation $U = V$ for some nondeterministic choices it returns equation $U' = V'$ with a solution S', such that $S'(U')$ is obtained from $S(U)$ by replacing each ab by c; similarly we say that a procedure implements blocks compression for a.

▶ **Lemma 5.** PairCompNCr(a, b) *preserves the unsatisfiability; for each solution S, if ab is a non-crossing pair for S in an equation $U = V$ then it preserves satisfiability and implements the pair compression of ab, .*

BlockCompNCr(a) *preserves unsatisfiability; for each solution S, if a has no crossing blocks in $U = V$ (for S) it preserves satisfiability and implements a's block compression.*

Crossing pairs and blocks compression

The presented algorithms cannot be directly applied to crossing pairs or to compression of a's blocks that have crossing appearances. To fix this, we modify the instance: if a pair ab is crossing because there is a variable X such that $S(X) = bw$ for some word w and a is to the left of X, it is enough to *left-pop* b from $S(X)$: we replace each X with bX and implicitly change S, so that $S(X) = w$; similar action is applied to variables Y ending with a and with b to the right (*right-popping* a from $S(X)$). Afterwards, ab is non-crossing with respect to S.

This idea can be employed much more efficiently: consider a partition of Γ into Γ_ℓ and Γ_r. The 'left-popping' from each variable a letter from Γ_r and 'right-popping' a letter from Γ_ℓ guarantees that each pair $ab \in \Gamma_\ell\Gamma_r$ is non-crossing. Since pairs from $\Gamma_\ell\Gamma_r$ do not overlap, after the preprocessing they can be compressed in parallel.

Algorithm 3 Pop(Γ_ℓ, Γ_r)

```
1: for $X \in \mathcal{X}$ do
2:     let $b$ be the first letter of $S(X)$                         ▷ Guess
3:     if $b \in \Gamma_r$ then
4:         replace each $X$ in $U$ and $V$ by $bX$   ▷ Implicitly change $S(X) = bw$ to $S(X) = w$
5:         if $S(X) = \epsilon$ then                                 ▷ Guess
6:             remove $X$ from the equation
7:                                  ▷ Perform a symmetric action for the last letter
```

▶ **Lemma 6.** Pop(Γ_ℓ, Γ_r) *preserves satisfiability and unsatisfiability.*

Furthermore, if S is a solution of $U = V$ then for all nondeterministic choices the obtained $U' = V'$ has a solution S' such that $S'(U') = S(U)$ and for some nondeterministic choices each pair ab from $\Gamma_\ell\Gamma_r$ is non-crossing (with regard to S').

Algorithm 4 PairComp(Γ_ℓ, Γ_r)

```
1: run Pop$(\Gamma_\ell, \Gamma_r)$
2: for $ab \in \Gamma_\ell\Gamma_r$ do
3:     run PairCompNCr$(a, b)$
```

▶ **Lemma 7.** PairComp(Γ_ℓ, Γ_r) *preserves satisfiability and unsatisfiability and for each solution it implements the pair compression of each pair $ab \in \Gamma_\ell\Gamma_r$.*

The problems with crossing blocks can be solved in a similar fashion: a has a crossing block, if aa is a crossing pair. So we 'left-pop' a from X until the first letter of $S(X)$ is different than a, we do the same with the ending letter b. This can be alternatively seen as removing the whole a-prefix (b-suffix, respectively) from X: suppose that $S(X) = a^\ell w b^r$, where w does not start with a nor end with b. Then we replace each X by $a^\ell X b^r$ implicitly changing the solution to $S(X) = w$. This eliminates crossing blocks with respect to S.

Algorithm 5 CutPrefSuff Cutting prefixes and suffixes

```
1: for $X \in \mathcal{X}$ do
2:     let $a$, $b$ be the first and last letter of $S(X)$
3:     guess $\ell_X \geq 1$, $r_X \geq 0$   ▷ $S(X) = a^{\ell_X} w b^{r_X}$, $w$ does not begin with $a$ nor end with $b$
4:     replace each $X$ in $U$ and $V$ by $a^{\ell_X} X b^{r_X}$   ▷ $a^{\ell_X}$, $b^{r_X}$ is stored in a compressed form
5:                              ▷ implicitly change $S(X) = a^{\ell_X} w b^{r_X}$ to $S(X) = w$
6:     if $S(X) = \epsilon$ then                                     ▷ Guess
7:         remove $X$ from the equation
```

▶ **Lemma 8.** CutPrefSuff *preserves unsatisfiability and satisfiability. For a solution S of $U = V$ and appropriate nondeterministic choices it returns an equation $U' = V'$ that has a solution S' such that $S(U) = S'(U')$ and $U' = V'$ has no crossing blocks with respect to S'.*

The procedure CutPrefSuff allows defining a procedure BlockComp that compresses maximal blocks of all letters, regardless of whether they have crossing blocks or not.

Algorithm 6 BlockComp

```
1: run CutPrefSuff
2: for each letter $a \in \Gamma$ do
3:     BlockCompNCr($a$)
```

▶ **Lemma 9.** BlockComp *preserves unsatisfiability and satisfiability and implements the block compression (for all letters a).*

3 Main algorithm, its time and space consumption, solutions' size

Now, the algorithm for testing satisfiability of word equations can be conveniently stated. We refer to one iteration of the main loop in WordEqSat as one *phase*.

Algorithm 7 WordEqSat Checking the satisfiability of a word equation

```
1: while $|U| > 1$ or $|V| > 1$ do
2:     run BlockComp
3:     $L \leftarrow$ the set of letters present in $U$ or $V$
4:     for $i \leftarrow 1..2$ do                    ▷ One to compress the equation, one the solution
5:         guess partition of $L$ into $L_1$ and $L_2$, run PairComp($L_1, L_2$)
6: Solve the problem naively                   ▷ With sides of length 1, the problem is trivial
```

The somehow peculiar double iteration of line 5 is for technical reasons: in one of the iterations we make sure that the solution is compressed, in the other that the equation representation is compressed, see Lemma 11.

▶ **Theorem 10.** WordEqSat *nondeterministically verifies the satisfiability of word equations. It can verify an existence of a length-minimal solution of length N in $\mathcal{O}(\mathsf{poly}(n) \log N)$ time and $\mathcal{O}(n \log n)$ space. For appropriate choices, the stored equation has length $\mathcal{O}(n)$.*

The correctness of WordEqSat follows from the fact that its subprocedures preserve satisfiability and unsatisfiability, which was shown in the previous section.

The lemma below formally states the idea that the solutions are compressed and it is a technical foundation for proofs of space and time consumption bounds.

▶ **Lemma 11.** *Let $U = V$ has a solution S. For appropriate nondeterministic choices the equation $U' = V'$ obtained at the end of the phase has a solution S' such that i) at least $1/6$ of letters in U or V are compressed in U' or V'; ii) at least $1/6$ of letters in $S(U)$ are compressed in $S'(U')$.*

Proof. We show the claim for $S(U)$ and then comment how the proof applies to U. Divide $S(U)$ into three-letter segments. We prove that for some partition into L_1 and L_2, in at least halve of these segments one of the letters in them is compressed, which shows the claim. Consider any such segment, let it be abc. If any of those letters is the same as its neighbouring letters, then by Lemma 9 for some nondeterministic choices BlockComp implements the compression of blocks, and so this letter is compressed and we are done.

So suppose that none of these letters is the same as its neighbouring letters, in particular, they are not compressed by BlockComp. Consider a random partition of L into L_1 and L_2, each letter goes to one part with probability $1/2$. There is a compression inside abc if $ab \in L_1L_2$ or $bc \in L_1L_2$. Each of those events has probability $1/4$ and they are disjoint, hence the compression appears with probability at least $1/2$. So regardless of the case, with probability $1/2$ at least one of letters in abc is compressed. There are $|S(U)|/3$ three-letter segments. The expected number of segments in which at least one letter is compressed is thus at least $|S(U)|/6$, so for some partition at least $|S(U)|/6$ letters are compressed.

Concerning U, the analysis is similar: we consider segments in U and V instead of $S(U)$ and $S(V)$. This corresponds to the second partition of L to L_1 and L_2. ◀

Lemma 11 is enough to show the bound on used memory: on one hand the new letters are introduced to the equations by Pop and CutPrefSuff, and their total number is $\mathcal{O}(n_v)$ per phase. This increases the lengths of U and V by $\mathcal{O}(n_v)$. On the other hand U and V are shortened by a constant factor; this yields a linear bound on $|U'|$ and $|V'|$. Moreover, Lemma 11 yields that for some choices there are $\mathcal{O}(\log N)$ phases.

4 Other results

4.1 Theoretical properties

Using the approach of recompression we give (alternative and often simpler) proofs of a doubly-exponential bound on the size of the length-minimal solution and an exponential bound on the periodicity bound.

4.1.1 Double exponential bound on minimal solutions

The running time of WordEqSat is polynomial in n and $\log N$ and it is easy to also *lower-bound* it in terms of $\log N$. On the other hand the length of the stored equations is $\mathcal{O}(n)$, which yields that there are exponentially (in n) many different configurations. Comparing those two bounds yields a doubly exponential bound on N.

4.1.2 Exponential bound on exponent of periodicity

For a word w the *exponent of periodicity* $\text{per}(w)$ is the maximal k such that u^k is a substring of w, for some $u \in \Gamma^+$; Γ-*exponent of periodicity* $\text{per}_\Gamma(w)$ restricts the choice of u to Γ. This notion is naturally transferred to equations: For an equation $U = V$, define the exponent of periodicity as $\max_S [\text{per}(S(U))]$, where the maximum is taken over all length-minimal solutions S of $U = V$; define the Γ-*exponent of periodicity* of $U = V$ in a similar way.

An exponential upper bound on Γ-exponent of periodicity of an equation is easy to obtain. Fix a solution S, for each variable X define ℓ_X (r_X): the length of maximal prefix (suffix, respectively) of $S(X)$ that is a block of the same letter. From Lemma 4 it follows that each length of maximal block can be expressed in terms of $\{\ell_X, r_X\}_{X \in \mathcal{X}}$ and constants. We define a different solution S' which is obtained by altering $\{\ell_X, r_X\}_{X \in \mathcal{X}}$. To guarantee that such a S' is indeed a solution we require that if two maximal blocks in $S(U)$ are of the same length, they have the same length in $S'(U)$ as well. Such a condition boils down to an equality of two expressions using $\{\ell_X, r_X\}_{X \in \mathcal{X}}$ and constants. When we treat $\{\ell_X, r_X\}_{X \in \mathcal{X}}$ as variables, we obtain a system of linear Diophantine equations. Each solution of this system defines a solution of $U = V$, in particular, length-minimal solutions come from minimal (in an appropriate sense) solutions of such Diophantine systems, which are known to be at most exponential.

To show a similar bound in the case of exponent of periodicity, we investigate, how it can be changed by subprocedures of WordEqSat. On one hand, if the exponent of periodicity is equal to Γ-exponent of periodicity then it is at most exponential. We show that when it is not then each subprocedure of WordEqSat can modify it at most by a constant. Hence, the upper bound on the exponent of periodicity is at most the sum of number of subprocedures and the Γ-exponent of periodicity, which are both at most exponential.

4.2 Linear space for $\mathcal{O}(1)$ variables

The length of the word equation kept by WordEqSat is linear. However, the letters in this equation can be all different, even if the input equation is over two letters. Hence the (nondeterministic) space usage is $\mathcal{O}(n \log n)$ bits. For $\mathcal{O}(1)$ variables (and unbounded number of appearances) we improve the bit consumption to only constant larger than the input.

The main obstacle is the encoding of letters introduced by WordEqSat, we give a more suitable one. Consider string of explicit letters between two consecutive variables X and Y in $U = V$. During WordEqSat the XwY will be changed to $Xw^{(1)}Y$, $Xw^{(2)}Y$, ... Each $w^{(i)}$ can be partitioned into 3 substrings $x^{(i)}v^{(i)}y^{(i)}$, where the letters in $v^{(i)}$ represent solely the letters from w, while each letter in $x^{(i)}$ ($y^{(i)}$) represent also some letter popped at some point from X (Y, respectively). We encode $v^{(i)}$ using only a constant time more bits than w: roughly, we represent letters as trees and when merging a and b into c, the tree of c has the tree of a as a left subtree and a tree of b as a right subtree.

On the other hand, the letters in $x^{(i)}$ and $y^{(i)}$ depend solely on XwY, so we simply encode them as $(XwY)1$, $(XwY)2$, ..., $(XwY)(|x^{(i)}| + |y^{(i)}|)$, where (XwY) is encoded as the corresponding fragment of the input and the numbers are encoded in binary. Note, that in this way different appearances of the same letter a may get the same code: in such case we collect the codes for a and add the information that they all represent the same letter.

It might be that $|x^{(i)}| + |y^{(i)}|$ is non-constant: WordEqSat guarantees that the length of the whole $|U| + |V|$ is $\mathcal{O}(n)$, but some fragments may become large. However, for $\mathcal{O}(1)$ variables we can enforce that in one phase WordEqSat compresses *each* pair of consecutive letters. In this way a stronger variant of Lemma 11 yields that each of $|x^{(i)}| + |y^{(i)}|$ is $\mathcal{O}(1)$. Let LinWordEqSat denote the such modified WordEqSat.

▶ **Theorem 12.** LinWordEqSat *preserves unsatisfiability and satisfiability. For k variables, it runs in (nondeterministic) $\mathcal{O}(mk^{ck})$ space, for some constant c, where m is the size of the input measured in bits.*

4.3 Representation of all solutions

Plandowski [14] gave an algorithm that generated a finite, graph-like representation of all solutions of a word equations. It is based on the idea that PlaSAT not only preserves satisfiability and unsatisfiability, but it in some sense operates on the solutions: when it transforms $U = V$ to $U' = V'$ then solutions of $U' = V'$ correspond to solutions of $U = V$. Moreover, each solution of $U = V$ can be represented in this way for some $U' = V'$. Hence, all solutions can be represented as a graph as follows: nodes are labelled with equations of the form $U = V$ and a directed edge leads from $U = V$ to $U' = V'$ if for some nondeterministic choices the former equation is transformed into the latter by PlaSAT. Furthermore, the edge describes, how the solutions of $U' = V'$ can be changed into the solutions of $U = V$. In this process PlaSAT is used to establish the existence of the edge and the operations that are associated with this edge. Since WordEqSat runs in PSPACE, such generation of labelled vertices and edges can also be performed in PSPACE, furthermore the description of a vertex (edge) is of polynomial size, so iteration over all vertices (edges, respectively) can be implemented in PSPACE. We describe how to employ WordEqSat in a similar procedure.

▶ **Theorem 13** (cf. [14]). *The graph representation of all solutions of an equation $U = V$ can be constructed in* PSPACE. *The size of the constructed graph is at most exponential.*

Transforming the solutions

As a first step we define precisely what does it mean that a subprocedure transforms the solutions. We use a notion of an *operator*, which is simply a function taking and returning a substitution. Then given a (nondeterministic) procedure transforming the equation $U = V$ we say that this procedure *transforms the solutions*, if based on the nondeterministic choices and the input equation we can define a family of operators $\mathcal{H}$ such that

- for any solution S of $U = V$ there are some nondeterministic choices that lead to an equation $U' = V'$ such that $S = H[S']$ for some solution S' of the equation $U' = V'$ and some operator $H \in \mathcal{H}$;
- for every solution S' of the obtained equation $U' = V'$ and for every operator $H \in \mathcal{H}$ the $H[S']$ is a solution of $U = V$.

Intuitively, in this way all solutions of an equation $U = V$ can be represented using the solutions of the possible outcome equations. We say that that $\mathcal{H}$ is the *corresponding family of inverse operators*. In many cases, $\mathcal{H} = \{H\}$, in such case we call H the *corresponding inverse operator*.

The lemmata showing that subprocedures of WordEqSat preserve satisfiability can be strengthened to show that they in fact transform the solutions and the appropriate family of inverse operators can be given.

Representation of solutions

The set of solutions will be represented by a directed graph $\mathcal{G}$: the vertices of $\mathcal{G}$ are labelled with equations $U = V$ that appear during the run of WordEqSat; furthermore, we require that the length of $U = V$ is at most cn and it has at most n_v appearances of variables, for some constant c. There is an edge from $U = V$ to $U' = V'$ if and only if for some nondeterministic choices WordEqSat transforms $U = V$ to $U' = V'$; such an edge is labelled with the corresponding family of inverse operators, which can be in one of the following forms: H replaces c in each $S(X)$ by ab; H appends (prepends) a^{ℓ_X} to $S(X)$; for each k replace each a_k in all $S(X)$ by a^k. Note that in the last type of operators, the a_k is just a naming convention, a priori we do not know, which letter is going to be replaced. So, the operators in $\mathcal{H}$ need to specify, which letters in $U' = V'$ are being replaced with a blocks and the lengths of the respective blocks. As such lengths are potentially unbounded, $\mathcal{H}$ can be infinite. However, using approach similar to the one presented in Section 4.1.2, the respective lengths can be represented compactly by a system of linear Diophantine equations, and so the description remains polynomial.

In order to generate the graph representation we need to be able to decide whether: a given equation $U = V$ labels a node in $\mathcal{G}$; given two equations $U = V$ and $U' = V'$ and $\mathcal{H}$ whether there is an edge from $U = V$ to $U' = V'$ labelled with $\mathcal{H}$. WordEqSat can be naturally used to answer such queries. Thus the construction of $\mathcal{G}$ is easy: We iterate over all equations $U = V$ of length at most cn, for a fixed $U = V$ we check whether it labels a node in $\mathcal{G}$. If so, we output it. We then perform a similar iteration for edges: for each pair of equations $U = V$ and $U' = V'$ and family of inverse operators $\mathcal{H}$ we verify, whether there is an edge from $U = V$ to $U' = V'$ labelled with $\mathcal{H}$. If so, we output the appropriate edge.

Acknowledgements I would like to thank P. Gawrychowski for initiating my interest in compressed membership problems, which eventually led to this work and for pointing to relevant literature [9, 11]; J. Karhumäki, for his question, whether the techniques of local recompression can be applied to the word equations; W. Plandowski for his comments and suggestions on the manuscript and questions concerning the space consumption that led to linear space algorithm for $\mathcal{O}(1)$ variables; anonymous referees, who pointed out several shortcomings and mistakes.

References

1 Stephen Alstrup, Gerth Stolting Brodal, and Theis Rauhe. Pattern matching in dynamic texts. In *SODA*, pages 819–828, 2000.

2 Volker Diekert. Makanin's algorithm. In M. Lothaire, editor, *Algebraic Combinatorics on Words*, chapter 12, pages 342–390. Cambridge University Press, 2002.

3 Claudio Gutiérrez. Satisfiability of word equations with constants is in exponential space. In *FOCS*, pages 112–119. IEEE Computer Society, 1998.

4 Joxan Jaffar. Minimal and complete word unification. *J. ACM*, 37(1):47–85, 1990.

5 Artur Jeż. Compressed membership for NFA (DFA) with compressed labels is in NP (P). In Christoph Dürr and Thomas Wilke, editors, *STACS*, volume 14 of *LIPIcs*, pages 136–147. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

6 Artur Jeż. Faster fully compressed pattern matching by recompression. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, editors, *ICALP*, volume 7391 of *LNCS*, pages 533–544. Springer, 2012.

7 Antoni Kościelski and Leszek Pacholski. Complexity of Makanin's algorithm. *J. ACM*, 43(4):670–684, 1996.

8 N. Jesper Larsson and Alistair Moffat. Offline dictionary-based compression. In *Data Compression Conference*, pages 296–305. IEEE Computer Society, 1999.

9 Markus Lohrey and Christian Mathissen. Compressed membership in automata with compressed labels. In Alexander S. Kulikov and Nikolay K. Vereshchagin, editors, *CSR*, volume 6651 of *LNCS*, pages 275–288. Springer, 2011.

10 G. S. Makanin. The problem of solvability of equations in a free semigroup. *Matematicheskii Sbornik*, 2(103):147–236, 1977. (in Russian).

11 Kurt Mehlhorn, R. Sundar, and Christian Uhrig. Maintaining dynamic sequences under equality tests in polylogarithmic time. *Algorithmica*, 17(2):183–198, 1997.

12 Wojciech Plandowski. Satisfiability of word equations with constants is in NEXPTIME. In *STOC*, pages 721–725, 1999.

13 Wojciech Plandowski. Satisfiability of word equations with constants is in PSPACE. *J. ACM*, 51(3):483–496, 2004.

14 Wojciech Plandowski. An efficient algorithm for solving word equations. In Jon M. Kleinberg, editor, *STOC*, pages 467–476. ACM, 2006.

15 Wojciech Plandowski and Wojciech Rytter. Application of Lempel-Ziv encodings to the solution of words equations. In Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors, *ICALP*, volume 1443 of *LNCS*, pages 731–742. Springer, 1998.

16 Hiroshi Sakamoto. A fully linear-time approximation algorithm for grammar-based compression. *J. Discrete Algorithms*, 3(2-4):416–430, 2005.

17 Klaus U. Schulz. Makanin's algorithm for word equations — two improvements and a generalization. In Klaus U. Schulz, editor, *IWWERT*, volume 572 of *LNCS*, pages 85–150. Springer, 1990.

Fast Algorithms for Abelian Periods in Words and Greatest Common Divisor Queries

Tomasz Kociumaka[1], **Jakub Radoszewski**[1], **and Wojciech Rytter**[*1,2]

1 **Faculty of Mathematics, Informatics and Mechanics, University of Warsaw, ul. Banacha 2, 02-097 Warsaw, Poland**
`[kociumaka,jrad,rytter]@mimuw.edu.pl`

2 **Faculty of Mathematics and Computer Science, Nicolaus Copernicus University, ul. Chopina 12/18, 87-100 Toruń, Poland**

Abstract

We present efficient algorithms computing all Abelian periods of two types in a word. Regular Abelian periods are computed in $O(n \log\log n)$ randomized time which improves over the best previously known algorithm by almost a factor of n. The other algorithm, for full Abelian periods, works in $O(n)$ time. As a tool we develop an $O(n)$ time construction of a data structure that allows $O(1)$ time $\gcd(i,j)$ queries for all $1 \le i,j \le n$, this is a result of independent interest.

1998 ACM Subject Classification E.1 Data Structures, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Abelian period, greatest common divisor

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.245

1 Introduction

The area of Abelian stringology was initiated by Erdös who posed a question about the smallest alphabet size for which there exists an infinite Abelian-square-free word, see [11]. An example of such a word over five-letter alphabet was given by Pleasants [18] and afterwards the optimal example over four-letter alphabet was shown by Keränen [16]. Quite recently there have been several results on Abelian complexity in words [2, 8, 9, 10] and partial words [3, 4] and on Abelian pattern matching [5, 17]. Abelian periods were first defined and studied by Constantinescu and Ilie [6].

We say that two words are commutatively equivalent, if one can be obtained from the other by permuting its symbols. This relation can be conveniently described using *Parikh vectors*, which show frequency of each symbol of the alphabet in a word: x and y are commutatively equivalent if and only if the Parikh vectors $\mathcal{P}(x)$ and $\mathcal{P}(y)$ are equal.

Let w be a non-empty word of length n over an alphabet $\Sigma = \{1, \dots, m\}$. We assume that $m \le n$, but if m is polynomially bounded, i.e. $m = n^{O(1)}$, the letters of w can be renumbered in $O(n)$ time so that $m \le n$. Let $\mathcal{P}(w)$ be an array such that $\mathcal{P}(w)[c]$ equals to the number of occurrences of the symbol $c \in \Sigma$ in w. Let us denote by $w[i\,..\,j]$ the factor $w_i \dots w_j$ and by $\mathcal{P}_{i,j}$ the Parikh vector $\mathcal{P}(w[i\,..\,j])$. For two vectors Q_1, Q_2 we write $Q_1 \le Q_2$ if $Q_1[c] \le Q_2[c]$ for each coordinate c.

An integer q is called an *Abelian period* of w if for $k = \left\lfloor \frac{n}{q} \right\rfloor$

$$\mathcal{P}_{1,q} = \mathcal{P}_{q+1,2q} = \dots = \mathcal{P}_{(k-1)q+1,kq} \quad \text{and } \mathcal{P}_{kq+1,n} \le \mathcal{P}_{1,q}.$$

* The author is supported by grant no. N206 566740 of the National Science Centre.

An Abelian period is called *full* if it is a divisor of n. A pair (q, i) is called a *weak Abelian period* of w if q is an Abelian period of $w[i+1\,..\,n]$ and $\mathcal{P}_{1,i} \le \mathcal{P}_{i+1,i+q}$. For example, the word *ababacabaabcbaab* has full Abelian periods 8 and 16, Abelian periods $6, 8, 9, 10, 11, 12, 13, 14, 15, 16$ and its shortest weak period is $(5, 3)$.

Fici et al. [13] gave an $O(n \log\log n)$ time algorithm for full Abelian periods and an $O(n^2)$ time algorithm for Abelian periods. An $O(n^2 m)$ time algorithm for weak Abelian periods was developed in [12] and it was recently improved to $O(n^2)$ time [7].

Our results. We present an $O(n)$ time deterministic algorithm finding all full Abelian periods. We also give an algorithm finding all Abelian periods, which comes in two variants: an $O(n \log\log n + n \log m)$ time deterministic and an $O(n \log\log n)$ time randomized. All algorithms run on $O(n)$ space in the standard word-RAM model with $\Omega(\log n)$ word size. The randomized algorithm is Monte Carlo and returns the correct answer with high probability, i.e. for each $c > 0$ the parameters can be set so that the probability of error is at most $\frac{1}{n^c}$.

As a tool we develop a data structure that after $O(n)$ preprocessing time computes $\gcd(i, j)$ for any $i, j \in \{1, \ldots, n\}$ in $O(1)$ time, which might be of its own interest. We are not aware of any solutions to this problem besides the folklore ones: preprocessing all answers ($O(n^2)$ preprocessing, $O(1)$ queries), using Euclid's algorithm (no preprocessing, $O(\log n)$ queries) or prime factorization ($O(n)$ preprocessing [14], queries in time proportional to the number of distinct prime factors, which is $O(\frac{\log n}{\log\log n})$).

The structure of the paper. Our algorithms use several non-trivial number-theoretic results, which are presented in the next two sections. The data structure for gcd-queries is developed in Section 2 and the tools specific to Abelian periods are described in Section 3. Then in Section 4 we introduce the proportionality relation on Parikh vectors, which provides a convenient characterization of Abelian periods in a string. Further properties of this relation are explored in Section 5. In particular we reduce efficient testing of this relation to a problem of equality of members of certain vector sequences, which potentially being of $\Theta(nm)$ total size, admit an $O(n)$-sized representation. Deterministic and randomized constructions of an efficient data structure for the vector equality problem (based on such representations) are proposed in Section 6. Finally in Section 7 we conclude with our main algorithms for Abelian periods and full Abelian periods.

2 Greatest Common Divisor queries

The key idea behind our data structure is an observation that gcd-queries are easy when one of the arguments is prime or both arguments are small enough for the precomputed answers to be used. We exploit this fact by reducing each query to a constant number of such special-case queries. In order to achieve this we define a *special decomposition* of an integer $k > 0$ as a triple (k_1, k_2, k_3) such that $k = k_1 \cdot k_2 \cdot k_3$ and

$$k_i \le \sqrt{k} \quad \text{or} \quad k_i \in \mathit{Primes} \qquad \text{for } i = 1, 2, 3.$$

▶ **Example 1.** $(2, 64, 64)$ is a special decomposition of 8192. $(1, 18, 479)$, $(2, 9, 479)$ and $(3, 6, 479)$ are up to permutations all special decompositions of 8622.

Let us introduce an operation $\otimes$ such that $(k_1, k_2, k_3) \otimes p$ results by multiplying the smallest of k_i's by p. For example, $(8, 2, 4) \otimes 7 = (8, 14, 4)$. For an integer $\ell > 1$, let $\mathit{MinDiv}[\ell]$ denote the least prime divisor of k.

▶ **Fact 2.** *Let $\ell > 1$ be an integer, $p = \mathit{MinDiv}[\ell]$ and $k = \ell/p$. If (k_1, k_2, k_3) is a special decomposition of k then $(k_1, k_2, k_3) \otimes p$ is a special decomposition of ℓ.*

Proof. Assume that $k_1 \le k_2 \le k_3$. If $k_1 = 1$ then $k_1 \cdot p = p$ is prime. Otherwise, k_1 is a divisor of ℓ and by the definition of p we have $p \le k_1$. Therefore: $(k_1 p)^2 = k_1^2 p^2 \le k_1^3 p \le k_1 k_2 k_3 p = \ell$. Consequently $k_1 p \le \sqrt{\ell}$ and in both cases $(k_1 p, k_2, k_3)$ is a special decomposition of ℓ. ◀

Fact 2 allows computing special decompositions provided that the values *MinDiv*$[k]$ can be computed efficiently. This is, however, a by-product of a linear time prime number sieve of Gries and Misra [14].

▶ **Lemma 3** ([14], Section 5)**.** *The values MinDiv$[k]$ for all $k \in \{2, \dots, n\}$ can be computed in $O(n)$ time.*

▶ **Theorem 4.** *After $O(n)$ time preprocessing, given any $k, \ell \in \{1, \dots, n\}$ the value* $\gcd(k, \ell)$ *can be computed in constant time.*

Proof. In the preprocessing phase we compute in $O(n)$ time two tables:
(a) a *Gcd-small*$[i,j]$ table such that *Gcd-small*$[i,j] = \gcd(i,j)$ for all $i, j \in \{1, \dots, \lfloor\sqrt{n}\rfloor\}$;
(b) a *Decomp*$[k]$ table such that *Decomp*$[k]$ is a special decomposition of k for each $k \le n$.

The *Gcd-small* table is filled using elementary steps in Euclid's subtraction algorithm and the *Decomp* table is computed according to Fact 2.

```
Algorithm Preprocessing(n)
    for $i := 1$ to $\lfloor\sqrt{n}\rfloor$ do
        Gcd-small$[i,i] := i$;
    for $i := 1$ to $\lfloor\sqrt{n}\rfloor$ do
        for $j := 1$ to $i-1$ do
            Gcd-small$[i,j]$ := Gcd-small$[i-j,j]$;
            Gcd-small$[j,i]$ := Gcd-small$[i-j,j]$;
    Decomp$[1] := (1,1,1)$;
    for $i := 2$ to $n$ do
        $p$ := MinDiv$[i]$;
        Decomp$[i]$ := Decomp$[i/p] \otimes p$;
    return (Gcd-small, Decomp);
```

```
Algorithm Query$(k, \ell)$
    $(x_1, x_2, x_3)$ := Decomp$[k]$;
    $(y_1, y_2, y_3)$ := Decomp$[\ell]$;
    $g := 1$;
    foreach $i, j \in \{1,2,3\}$ do
        if $\max(x_i, y_j) \le \sqrt{n}$ then
            $d$ := Gcd-small$[x_i, y_j]$;
        else if $x_i = y_j$ then $d := x_i$;
        else $d := 1$;
        $g := g \cdot d$;
        $x_i := x_i/d$;  $y_j := y_j/d$;
    return $g$;
```

The algorithm Query(k, ℓ) computes $\gcd(k, \ell)$ using special decompositions (x_1, x_2, x_3) and (y_1, y_2, y_3) of k and ℓ respectively. The values x_i and y_j are altered during the execution of the algorithm, but remain prime or bounded by $\sqrt{n}$. In each step we have $d = \gcd(x_i, y_j)$; if $x_i, y_j \le \sqrt{n}$ then *Gcd-small* table is used and otherwise the gcd can be greater than 1 only if $x_i = y_j \in$ *Primes*. We maintain an invariant that $k = x_1 x_2 x_3 \cdot g$ and $\ell = y_1 y_2 y_3 \cdot g$. At the end $\gcd(x_i, y_j) = 1$ holds for all $i, j \in \{1,2,3\}$ and consequently $g = \gcd(k, \ell)$. ◀

3 Number-theoretic tools for Abelian periods

Now we introduce two abstract *filter* operations and show how to perform them efficiently.

For integers $n, k > 0$ let *Mult*(k, n) be the set of multiples of k not exceeding n, i.e.

$$Mult(k,n) = \{m \cdot k : m \in \mathbb{Z}_+, m \cdot k \le n\}.$$

Also denote *Div*$(n) = \{d \in \mathbb{Z}_+ : d \mid n\}$, the set of divisors of n.

▶ **Lemma 5.** *Let n be a positive integer and $A \subseteq \{1, \dots, n\}$. There exists an $O(n)$ time algorithm that computes the set*

$$FILTER1(A, n) = \{d \in Div(n) : Mult(d, n) \subseteq A\}.$$

Proof. Let $A' = \{1, \ldots, n\} \setminus A$. Observe that for $d \in \mathit{Div}(n)$

$$d \notin \mathit{FILTER1}(A, n) \iff \exists_{j \in A'}\ d \mid j.$$

Moreover, for $d \in \mathit{Div}(n)$ and $j \in \{1, \ldots, n\}$ we have

$$d \mid j \iff d \mid d', \quad \text{where} \quad d' = \gcd(j, n).$$

These observations lead to the following algorithm.

```
Algorithm $FILTER1(A, n)$
    $D := Div(n)$; $X := Div(n)$;
    foreach $j \in A'$ do
        $D := D \setminus \{\gcd(j, n)\}$;
    foreach $d, d' \in Div(n)$ do
        if $d \mid d'$ and $d' \notin D$ then
            $X := X \setminus \{d\}$;
    return $X$;
```

We use $O(1)$ time gcd queries from Theorem 4. The number of pairs (d, d') is $o(n)$, since $|\mathit{Div}(n)| = o(n^{\varepsilon})$ for any $\varepsilon > 0$, see [1]. Consequently, the algorithm runs in $O(n)$ time. ◀

▶ **Lemma 6.** *Let $\approx$ be an arbitrary equivalence relation on $\{k_0, k_0 + 1, \ldots, n\}$ which can be tested in constant time. Then, there exists an $O(n \log \log n)$ time algorithm that computes the set:*

$$\mathit{FILTER2}(\approx) = \{k \in \{k_0, \ldots, n\} \,:\, \forall_{i \in \mathit{Mult}(k,n)}\ i \approx k\}.$$

Proof. In the algorithm we use the following observation, which holds for $k \in \{k_0, \ldots, n\}$:

$$k \in \mathit{FILTER2}(\approx) \iff \forall_{p \in \mathit{Primes}\,:\, k \cdot p \le n}\ (k \approx k \cdot p\ \wedge\ k \cdot p \in \mathit{FILTER2}(\approx)). \tag{1}$$

The $(\Rightarrow)$ part of the equivalence is obvious. For the proof of the $(\Leftarrow)$ part consider any k satisfying the right hand side of (1) and any integer $\ell \ge 2$ such that $k \cdot \ell \le n$. We need to show that $k \approx k \cdot \ell$. Let p be a prime divisor of ℓ. By the right hand side, we have $k \approx k \cdot p$, and since $k \cdot p \in \mathit{FILTER2}(\approx)$, we get $k \cdot p \approx k \cdot p \cdot (\ell/p) = k \cdot \ell$.

The following algorithm uses (1) for k decreasing from $n/2$ to k_0 to compute $\mathit{FILTER2}(\approx)$. It uses an invariant $Y = \{k_0, \ldots, k\} \cup (\mathit{FILTER2}(\approx) \cap \{k + 1, \ldots, n\})$ while checking the right hand side of (1) for k.

```
Algorithm $FILTER2(\approx)$
    $Y := \{k_0, \ldots, n\}$;
    for $k := n/2$ downto $k_0$ do
        foreach $p \in Primes$, $p \cdot k \le n$ do
(⋆)         if $k \cdot p \not\approx k$ or $k \cdot p \notin Y$ then
                $Y := Y \setminus \{k\}$;
    return $Y$;
```

In the algorithm we assume to have an ordered list of primes up to n. It can be computed in $O(n)$ time, see [14]. For a fixed $p \in \mathit{Primes}$ the instruction $(\star)$ is called for at most $\frac{n}{p}$ values of k. The total number of operations performed by the algorithm is thus $O(n \log \log n)$ due to the following well-known fact from number theory, see [1]:

$$\sum_{p \in \mathit{Primes},\, p \le n} \tfrac{1}{p} = O(\log \log n).$$ ◀

4 Characterization of Abelian periods by proportionality relation

Let w be a word of length n. Let $\mathcal{P}_i = \mathcal{P}(w[1\,..\,i])$. Two positions $i, j \in \{1, \dots, n\}$ are called *proportional*, which we denote $i \sim j$, if $\mathcal{P}_i[k] = c \cdot \mathcal{P}_j[k]$ for each k, where c is a real number independent of k. Note that $\sim$ is an equivalence relation, see also Figure 1.

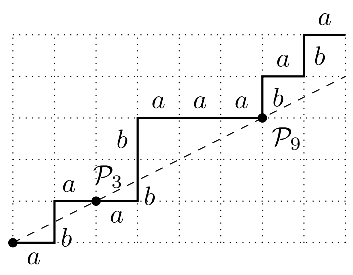


Figure 1 Here $\mathcal{P}_3 = (2,1)$ (the word *aba*) and $\mathcal{P}_9 = (6,3)$ (the word *abaabbaaa*), hence $3 \sim 9$. In other words, the points $\mathcal{P}_3$ and $\mathcal{P}_9$ lie on the same line originating from $(0,0)$.

▶ **Definition 7.** An integer k is called a *candidate* (as a potential Abelian period) if $i \sim k$ for each $i \in \mathit{Mult}(k,n)$, or equivalently $k \sim 2k \sim 3k \sim \dots$

Define the following *tail* table (assume $\min \emptyset = \infty$):

$$\mathit{tail}[i] \;=\; \min\{j : \mathcal{P}_{i,n} \le \mathcal{P}_{i-j,i-1}\}.$$

▶ **Example 8.** For the Fibonacci word $\mathit{Fib} = abaababaabaababaababa$ of length 21, the first ten elements of the table $\mathit{tail}[i]$ are ∞, the remaining eleven elements are:

i	11	12	13	14	15	16	17	18	19	20	21
$\mathit{Fib}[i]$	a	a	b	a	b	a	a	b	a	b	a
$\mathit{tail}[i]$	∞	10	11	8	8	7	5	5	3	2	2

The following lemma is proved (implicitly) in [7].

▶ **Lemma 9.** *Let w be a word of length n. The values $\mathit{tail}[i]$ for $1 \le i \le n$ can be computed in $O(n)$ time.*

The notions of a candidate and the tail table let us formulate a condition for a given integer to be an Abelian period or a full Abelian period of w.

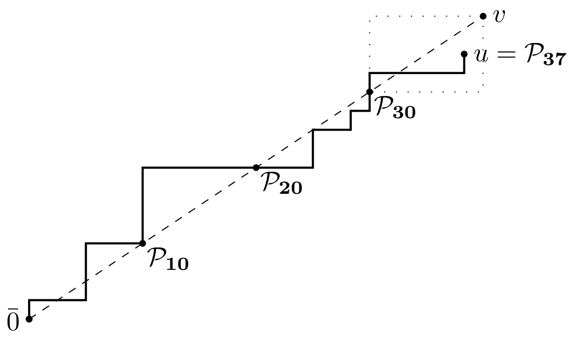


Figure 2 Illustration of Fact 10b. The word $u = baaabbbaaabbbbaaaaaaaaabbaababbaaaaab$ of length 37 has an Abelian period 10. We have $10 \sim 20 \sim 30$ and $\mathcal{P}_{31,37}$ is dominated by the period $(\mathcal{P}_{10})$, i.e. the graph of the word ends within the rectangle marked in the figure, the point v dominates the point u.

▶ **Fact 10.** *Let w be a word of length n. A positive integer $q \le n$ is:*
(a) *a full Abelian period of w if and only if $q \mid n$ and q is a candidate;*
(b) *an Abelian period of w if and only if q is a candidate and $\mathcal{P}_{kq+1,n} \le \mathcal{P}_{(k-1)q+1,kq}$ for $k = \left\lfloor \frac{n}{q} \right\rfloor$, which is equivalent to tail$[kq+1] \le q$ (see Fig. 2).*

5 Efficient implementation of the proportionality relation

Denote by $s = \mathit{LeastFreq}(w)$, a least frequent letter of w. Let q_0 be the position of the first occurrence of s in w. For $i \in \{q_0, \dots, n\}$ let $\gamma_i = \mathcal{P}_i/\mathcal{P}_i[s]$. Vectors γ_i are introduced in order to deal with vector equality instead of vector proportionality.

▶ **Lemma 11.** *If $i, j \in \{q_0, \dots, n\}$ then $i \sim j$ is equivalent to $\gamma_i = \gamma_j$.*

Proof. ($\Rightarrow$) If $i \sim j$ then the vectors $\mathcal{P}_i$ and $\mathcal{P}_j$ are proportional. Multiplying any of them by a constant only changes the proportionality ratio. Hence, $\mathcal{P}_i/\mathcal{P}_i[s]$ and $\mathcal{P}_j/\mathcal{P}_j[s]$ are proportional. The denominators of both fractions are positive, since $i, j \ge q_0$. However, the s-th components of γ_i and γ_j are 1, consequently these vectors are equal.

($\Leftarrow$) $\mathcal{P}_i/\mathcal{P}_i[s] = \mathcal{P}_j/\mathcal{P}_j[s]$ means that $\mathcal{P}_i$ and $\mathcal{P}_j$ are proportional, so that $i \sim j$. ◀

▶ **Example 12.** Consider the word $w = acbaabacaacb$ for which the alphabet is of size 3, $\mathit{LeastFreq}(w) = b$ and $q_0 = 3$. We have:

$$\gamma_3 = (1,1,1), \quad \gamma_4 = (2,1,1), \quad \gamma_5 = (3,1,1), \quad \gamma_6 = (\tfrac{3}{2},1,\tfrac{1}{2}), \quad \gamma_7 = (2,1,\tfrac{1}{2}),$$
$$\gamma_8 = (2,1,1), \quad \gamma_9 = (\tfrac{5}{2},1,1), \quad \gamma_{10} = (3,1,1), \quad \gamma_{11} = (3,1,\tfrac{3}{2}), \quad \gamma_{12} = (2,1,1),$$

We conclude that $\gamma_4 = \gamma_8 = \gamma_{12}$ and $\gamma_5 = \gamma_{10}$ and consequently $4 \sim 8 \sim 12$ and $5 \sim 10$.

Let us formally define a natural way to store a sequence of vectors with a small total Hamming distance between consecutive elements, like $\mathcal{P}_i$ or, as we prove in Lemma 15, γ_i.

▶ **Definition 13.** Given a vector v, consider an *elementary operation* of the form "$v[j] := x$" that changes the j-th component of v to x. Let $\bar{u}_1, \dots, \bar{u}_k$ be a sequence of vectors of the same dimension, and let $\xi = (\sigma_1, \dots, \sigma_r)$ be a sequence of elementary operations. We say that ξ is a *diff-representation* of $\bar{u}_1, \dots, \bar{u}_k$ if $(\bar{u}_i)_{i=1}^{k}$ is a subsequence of the sequence $(\bar{v}_j)_{j=0}^{r}$, where $\bar{v}_j = \sigma_j(\dots(\sigma_2(\sigma_1(\bar{0})))\dots)$.

▶ **Example 14.** Let ξ be the sequence: $v[1] := 1$, $v[2] := 2$, $v[1] := 4$, $v[3] := 1$, $v[4] := 3$, $v[3] := 0$, $v[1] := 1$, $v[2] := 0$, $v[4] := 0$, $v[1] := 3$, $v[2] := 2$, $v[1] := 2$, $v[4] := 1$. This sequence is schematically presented as the top rectangle in Fig. 3. The sequence of vectors produced by the sequence ξ, starting from $\bar{0}$, is:

$$(0,0,0,0),\ (1,0,0,0),\ (1,2,0,0),\ (4,2,0,0),\ (4,2,1,0),\ (4,2,1,3),\ (4,2,0,3),$$
$$(1,2,0,3),\ (1,0,0,3),\ (1,0,0,0),\ (3,0,0,0),\ (3,2,0,0),\ (2,2,0,0),\ (2,2,0,1).$$

Hence ξ is a diff-representation of the above vector sequence as well as all its subsequences.

▶ **Lemma 15.** **(a)** *$\sum \mathit{dist}_H(\gamma_{i+1}, \gamma_i) \le 2n$, where dist_H is the Hamming distance.*
(b) *An $O(n)$-sized diff-representation of $(\gamma_i)_{i=q_0}^{n}$ can be computed in $O(n)$ time.*

Proof. To prove (a) observe that $\mathcal{P}_i$ differs from $\mathcal{P}_{i-1}$ only at the coordinate corresponding to $w[i]$. If $w[i] \ne s$, the same holds for γ_i and γ_{i-1}. If $w[i] = s$, vectors γ_i and γ_{i-1} may differ on all coordinates, so m operations might be necessary, but s occurs at most $\frac{n}{m}$ times.

As a direct consequence of (a), the sequence $(\gamma_i)_{i=q_0}^{n}$ admits a diff-representation with at most $2n + m$ operations in total. It can be computed by an algorithm that apart from γ_i maintains $\mathcal{P}_i$ in order to compute the new values of the changing coordinates of γ_i. ◀

▶ **Lemma 16.** *For a word w of length n, the equivalence class of n under $\sim$ can be computed in $O(n)$ time.*

Proof. Observe that if $k \sim n$ then $k \geq q_0$. Indeed, if $k \sim n$, then $\mathcal{P}_k$ is proportional to $\mathcal{P}_n$, so all letters occurring in w also occur in $w[1 \dots k]$. This lets us use the characterization of Lemma 11 and a diff-representation provided by Lemma 15 to reduce the task to the following problem with $\delta_i = \gamma_i - \gamma_n$.

▶ **Claim 17.** *Given a diff-representation of the vector sequence $\delta_{q_0}, \dots, \delta_n$ we can decide for which i vector δ_i is equal to $\bar{0}$ in $O(m+r)$ time, where m is the size of the vectors and r is the size of the representation.*

The solution simply maintains δ_i and the number of non-zero coordinates of δ_i. ◀

The main tool for proportionality queries is a data structure for the following problem.

▶ **Problem 1 (Integer vector equality).** *Assume we are given a diff-representation ξ of a vector sequence $(\bar{u}_i)_{i=1}^{k}$. Let m be the dimension of the vectors and r be the size of the representation. Assume the vectors have integer components of absolute value $(m+r)^{O(1)}$. Preprocess ξ to answer queries of the form: "Is $\bar{u}_i = \bar{u}_j$?" for $i, j \in \{1, \dots, k\}$.*

In Section 6 we show that after $O(m + r\log m)$ time deterministic or $O(m+r)$ time randomized preprocessing these queries can be answered in constant time. In the latter case, with a small probability we can get false positive answers.

Note that the next lemma can be used for testing proportionality only for $i, j \geq q_0$. In other words, it allows testing $\sim|_{\{q_0,\dots,n\}}$, the restriction of $\sim$ to $\{q_0, \dots, n\}$.

▶ **Lemma 18.** *Let w be a word of length n over an alphabet of size m. There exists a data structure of $O(n)$ size which for given $i, j \in \{q_0, \dots, n\}$ decides whether $i \sim j$ in constant time. It can be constructed by an $O(n \log m)$ time deterministic or an $O(n)$ time randomized algorithm (Monte Carlo, correct with high probability).*

Proof. By Lemma 11, to answer the proportionality-queries it suffices to efficiently compare the vectors γ_i, which, by Lemma 15, admit a diff-representation of size $O(n)$. Problem 1 requires integer values, so we split γ into two sequences α and β, of numerators and denominators respectively. We need to store the fractions in a reduced form so that comparing numerators and denominators can be used to compare fractions. Thus we set

$$\alpha_i[j] = \mathcal{P}_i[j]/d \quad \text{and} \quad \beta_i[j] = \mathcal{P}_i[s]/d,$$

where $d = \gcd(\mathcal{P}_i[j], \mathcal{P}_i[s])$ can be computed in $O(1)$ time using a single gcd-query of Theorem 4, since the values of $\mathcal{P}_i$ are non-negative integers up to n. Consequently the values of α and β are also positive integers not exceeding n. This allows using a solution to Problem 1 given in Theorem 28, so that the whole algorithm runs in the desired $O(n \log m)$ and $O(n)$ time, respectively, using $O(n)$ space. ◀

6 Vector equality in diff-representation

Recall that in the integer vector equality problem we are given a diff-representation of a vector sequence $(\bar{u}_i)_{i=1}^{k}$, i.e. a sequence ξ of elementary operations $\sigma_1, \sigma_2, \dots, \sigma_r$ on a vector of dimension m. Each σ_i is of the form: set the j-th component to some value x. We assume that x is an integer of magnitude $(m+r)^{O(1)}$. Let $\bar{v}_0 = \bar{0}$ and for $1 \leq i \leq r$ let $\bar{v}_i$ be the vector obtained from $\bar{v}_{i-1}$ by performing σ_i. Our task is answering queries of the form

"Is $\bar{u}_i = \bar{u}_j$?" but it reduces to answering equality queries of the form "Is $\bar{v}_i = \bar{v}_j$?", since $(\bar{u}_i)_{i=1}^{k}$ is a subsequence of $(\bar{v}_i)_{i=0}^{r}$ by definition of the diff-representation.

▶ **Definition 19.** A function $H : \{0,\dots,r\} \to \{0,\dots,\ell\}$ is called an *ℓ-naming* for ξ if $H(i) = H(j)$ holds if and only if $\bar{v}_i = \bar{v}_j$.

In order to answer the equality queries we construct an ℓ-naming with $\ell = (m+r)^{O(1)}$. Integers of this magnitude can be stored in $O(1)$ space, so this suffices to answer the equality queries in constant time.

6.1 Deterministic construction of a naming function

Let $\xi = (\sigma_1,\dots,\sigma_r)$ be a sequence of operations on a vector of dimension m. Let $A = \{1,\dots,m\}$ be the set of coordinates. For any $B \subseteq A$, let $select_B[i]$ be the index of the ith operation concerning B in ξ. Moreover, let $rank_B[i]$, where $i \in \{0,\dots,r\}$, be the number of operations concerning coordinates in B among $\sigma_1,\dots,\sigma_i$ and let $r_B = rank_B[r]$.

▶ **Definition 20.** Let ξ be a sequence of operations, A be the set of coordinates and $B \subseteq A$. Let $h : \{0,\dots,r_B\} \to \mathbb{Z}$ be a function. Then define:

$$Squeeze(\xi, B) = \xi_B \quad \text{where } \xi_B[i] = \xi[select_B[i]],$$

$$Expand(\xi, B, h) = \eta_B \quad \text{where } \quad \eta_B[i] = h(rank_B[i]).$$

In other words, the squeeze operation produces a subsequence ξ_B of ξ consisting of operations concerning B. The expand operation is in some sense an inverse of the squeeze operation, it propagates the values of h from the domain B to the full domain A.

▶ **Example 21.** Let ξ be the sequence from Example 14, here $A = \{1,2,3,4\}$. Let $B = \{1,2\}$ and assume $H_B = [0,1,2,6,2,1,4,5,3]$. Then (see also Fig. 3):

$$Expand(\xi, B, H_B) = (0,1,2,6,6,6,6,2,1,1,4,5,3,3).$$

For a pair of sequences η', η'', denote by $Align(\eta', \eta'')$ the sequence of pairs η such that $\eta[i] = (\eta'[i], \eta''[i])$ for each i. Moreover, for a sequence η of $r+1$ pairs of integers, denote by $Renumber(\eta)$ a sequence H of $r+1$ integers in the range $\{0,\dots,r\}$ such that $\eta[i] < \eta[j]$ if and only if $H[i] < H[j]$ for any $i, j \in \{0,\dots,r\}$.

The recursive construction of a naming function for ξ is based on the following fact.

▶ **Fact 22.** *Let ξ be a sequence of elementary operations, $A = B \cup C$ ($B \cap C = \emptyset$) be the set of coordinates, H_B be an r_B-naming function for ξ_B and H_C an r_C-naming function for ξ_C. Additionally, let*

$$\eta_B = Expand(\xi, B, H_B), \quad \eta_C = Expand(\xi, C, H_C), \quad H = Renumber(Align(\eta_B, \eta_C))$$

Then H is an r-naming function for ξ.

The algorithm makes an additional assumption about the sequence ξ.

▶ **Definition 23.** We say that a sequence of operations ξ is *normalized* if for each operation $v[j] := x$ we have $x \in \{0,\dots,r_{\{j\}}\}$, where (as defined above) $r_{\{j\}}$ is the number of operations in ξ concerning the jth coordinate.

If for each operation $v[j] := x$ the value x is of magnitude $(m+r)^{O(1)}$, then normalizing the sequence ξ, i.e., constructing a normalized sequence with the same answers to all equality queries, takes $O(m+r)$ time. This is done using a radix sort of triples (j, x, i) and by mapping the values x corresponding to the same coordinate j to consecutive integers.

▶ **Lemma 24.** *Let ξ be a normalized sequence of r operations on a vector of dimension m. An r-naming for ξ can be deterministically constructed in $O(r \log m)$ time.*

Proof. If the dimension of vectors is 1 (that is, $|A| = 1$), the single components of the vectors $\bar{v}_i$ already constitute an r-naming. This is due to the fact that ξ is normalized.

For larger $|A|$, the algorithm uses Fact 22, see the pseudocode below and Figure 3.

```
Algorithm $ComputeH(\xi)$
    if $\xi$ is empty then return $\bar{0}$;
    if $|A| = 1$ then compute $H$ naively;
    Split $A$ into two halves $B$, $C$;
    $\xi_B := Squeeze(\xi, B)$; $\xi_C := Squeeze(\xi, C)$;
    $H_B := ComputeH(\xi_B)$; $H_C := ComputeH(\xi_C)$;
    $\eta_B := Expand(\xi, B, H_B)$; $\eta_C := Expand(\xi, C, H_C)$;
    return $Renumber(Align(\eta_B, \eta_C))$;
```

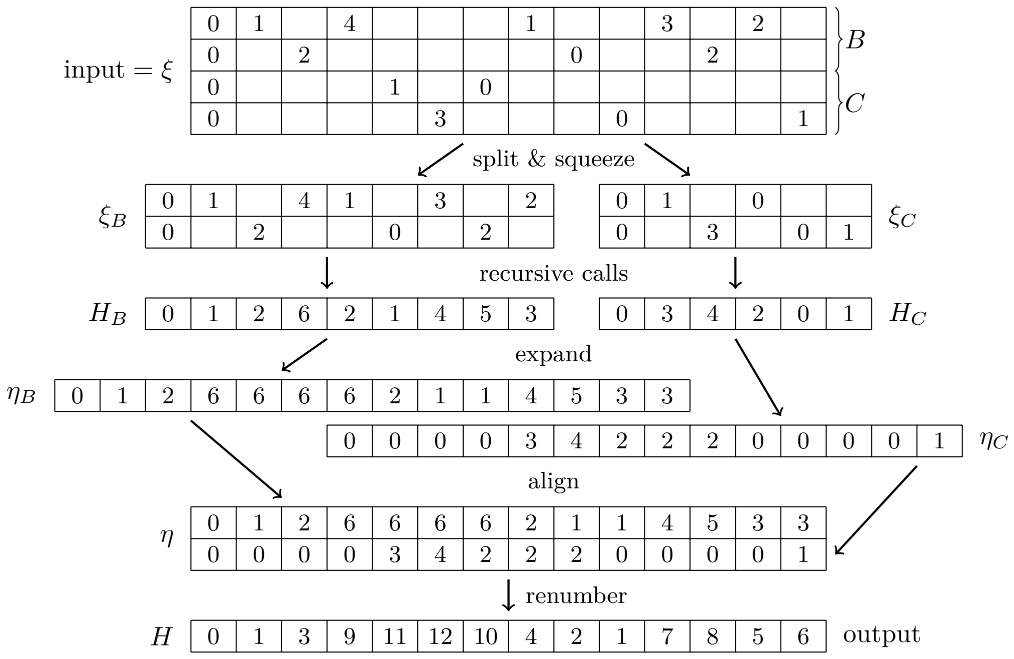


Figure 3 A schematic diagram of performance of algorithm *ComputeH*. The columns correspond to elementary operations and the rows correspond to coordinates of the vectors.

Let us analyze the complexity of a single recursive step of the algorithm. Tables *rank* and *select* are computed in $O(r)$ time, hence both squeezing and expanding are performed in $O(r)$ time. Renumbering, implemented using radix sort and bucket sort, also runs in $O(r)$ time, since the values of H_B and H_C are positive integers bounded by r. Hence, the recursive step takes $O(r)$ time.

We obtain the following recursive formula for $T(r,m)$, an upper bound on the execution time of the algorithm for a sequence of r operations on a vector of length m:

$$T(r,1) = O(r), \quad T(0,m) = O(1)$$
$$T(r,m) = T(r_1, \lfloor m/2 \rfloor) + T(r_2, \lceil m/2 \rceil) + O(r) \quad \text{where } r_1 + r_2 = r.$$

A solution to this recurrence yields $T(r,m) = O(r \log m)$. ◀

6.2 Randomized construction of a naming function

Our randomized construction is based on fingerprints, see [15]. Let us fix a prime number p. For a vector $\bar{v} = (v_1, v_2, \dots, v_m)$ we introduce a polynomial over the field $\mathbb{Z}_p$:

$$Q_{\bar{v}}(x) = v_1 + v_2 x + v_3 x^2 + \ldots + v_m x^{m-1} \in \mathbb{Z}_p[x].$$

Let us choose $x_0 \in \mathbb{Z}_p$ uniformly at random. Clearly, if $\bar{v} = \bar{v}'$ then $Q_{\bar{v}}(x_0) = Q_{\bar{v}'}(x_0)$. The following lemma states that the converse is true with high probability.

▶ **Lemma 25.** *Let $\bar{v} \neq \bar{v}'$ be vectors in $\{0,\dots,n\}^m$. Let $p > n$ be a prime number and let $x_0 \in \mathbb{Z}_p$ be chosen uniformly at random. Then*

$$\mathbb{P}\left(Q_{\bar{v}}(x_0) = Q_{\bar{v}'}(x_0)\right) \leq \tfrac{m}{p}.$$

Proof. Note that, since $p > n$, $R(x) = Q_{\bar{v}}(x) - Q_{\bar{v}'}(x) \in \mathbb{Z}_p[x]$ is a non-zero polynomial of degree $\leq m$, hence it has at most m roots. Consequently, x_0 is a root of R with probability bounded by m/p. ◀

▶ **Lemma 26.** *Let $\bar{v}_1, \dots, \bar{v}_r$ be vectors in $\{0,\dots,n\}^m$. Let $p > \max(n, (m+r)^{c+3})$ be a prime number, where c is a positive constant, and let $x_0 \in \mathbb{Z}_p$ be chosen uniformly at random. Then $H(i) = Q_{\bar{v}_i}(x_0)$ is a naming function with probability at least $1 - \frac{1}{(m+r)^c}$.*

Proof. Assume that H is not a naming function. This means that there exist i, j such that $H(i) = H(j)$ despite $\bar{v}_i \neq \bar{v}_j$. Hence, by the union bound and Lemma 25 we obtain the conclusion of the lemma:

$$\mathbb{P}(H \text{ is not a naming}) \leq \sum_{i,j\,:\,\bar{v}_i \neq \bar{v}_j} \mathbb{P}\left(H(i) = H(j)\right) \leq \sum_{i,j\,:\,\bar{v}_i \neq \bar{v}_j} \tfrac{m}{p} \leq \tfrac{mr^2}{p} \leq \tfrac{1}{(m+r)^c}.$$ ◀

▶ **Lemma 27.** *Let ξ be a sequence of r operations on a vector of dimension m with values of magnitude $n = (m+r)^{O(1)}$. There exists a randomized $O(m+r)$ time algorithm that constructs a function H which is a k-naming for ξ with high probability for $k = (m+r)^{O(1)}$.*

Proof. Assume all values in ξ are bounded by $(m+r)^{c'}$. Let $c \geq c'$. Let us choose a prime p such that $(m+r)^{3+c} < p < 2(m+r)^{3+c}$. Moreover let $x_0 \in \mathbb{Z}_p$ be chosen uniformly at random.

Then we set $H(i) = Q_{\bar{v}_i}(x_0)$. By Lemma 26, this is a naming function with probability at least $1 - \frac{1}{(m+r)^c}$.

If we know all powers $x_0^j \bmod p$ for $j \in \{1,\dots,m\}$, then we can compute $H(i)$ from $H(i-1)$ (a single operation) in constant time. Thus $H(i)$ for all $1 \leq i \leq r$ can be computed in $O(m+r)$ time. ◀

With a naming function stored in an array, answering equality queries is straightforward. In the randomized version, there is a small chance that H is not a naming function, which makes the queries Monte Carlo (with one-sided error). Nevertheless, the answers are correct with high probability. Thus we obtain the following result.

▶ **Theorem 28.** *The integer vector equality problem can be solved in $O(n)$ space and:*
(a) *in $O(m + r \log m)$ time deterministically or*
(b) *in $O(m+r)$ time using a Monte Carlo algorithm (with one-sided error, correct w.h.p.).*

7 Two main algorithms

In this section we combine our tools to develop efficient algorithms computing all Abelian periods of two types.

▶ **Theorem 29.** *Let w be a word of length n over the alphabet $\{1, \dots, m\}$. Full Abelian periods of w can be computed in $O(n)$ time.*

Proof. Full Abelian periods are computed using the characterization given by Fact 10a. Recall that $\mathit{FILTER1}([n]_\sim, n) = \{d \mid n : \mathit{Mult}(d,n) \subseteq [n]_\sim\} = \{d \mid n : \mathit{Mult}(d,n) \subseteq [d]_\sim\}$.

```
Algorithm Full Abelian periods
    Compute the data structure for answering gcd queries;     {Theorem 4}
    $A := \{k : k \sim n\}$;                                   {Lemma 16}
    $\mathcal{K} := \mathit{FILTER1}(A, n)$;                   {Lemma 5}
    return $\mathcal{K}$;
```

The algorithms from Lemmas 5 and 16 take $O(n)$ time. Hence, the whole algorithm works in linear time. ◀

▶ **Theorem 30.** *Let w be a word of length n over the alphabet $\{1, \dots, m\}$. There exist an $O(n \log\log n + n \log m)$ time deterministic and an $O(n \log\log n)$ time randomized algorithm that compute all Abelian periods of w. Both algorithms require $O(n)$ space.*

Proof. Abelian periods are computed using the characterization given by Fact 10b. Recall that Lemma 18 allows testing $\sim|_{\{q_0,\dots,n\}}$, the restriction of $\sim$ to $\{q_0, \dots, n\}$, only. Nevertheless all Abelian periods of w are at least q_0 and thus it suffices to initialize Y to the set of candidates greater than or equal to q_0, that is the set

$$\{k \in \{q_0, \dots, n\} : \mathit{Mult}(k,n) \subseteq [k]_\sim\} = \mathit{FILTER2}(\sim|_{\{q_0,\dots,n\}}).$$

```
Algorithm Abelian periods
    Compute the data structure for answering gcd queries;                     {Theorem 4}
    Prepare data structure to answer in $O(1)$ time proportionality-queries;  {Lemma 18}
    Compute table $\mathit{tail}$;                                            {Lemma 9}
    $Y := \mathit{FILTER2}(\sim|_{\{q_0,\dots,n\}})$;                         {Lemma 6}
    $\mathcal{K} := \emptyset$;
    foreach $q \in Y$ do
        $j := q \cdot \lfloor \frac{n}{q} \rfloor + 1$;
        if $\mathit{tail}[j] < q$ then $\mathcal{K} := \mathcal{K} \cup \{q\}$;
    return $\mathcal{K}$;
```

The deterministic version of the algorithm from Lemma 18 runs in $O(n \log m)$ time and the randomized version runs in $O(n)$ time. The algorithm from Lemma 6 runs in $O(n \log\log n)$ time and all the remaining algorithms (see Theorem 4, Lemma 9) run in linear time. This implies the required complexity of the Abelian periods' computation. ◀

References

1 Tom M. Apostol. *Introduction to Analytic Number Theory.* Undergraduate Texts in Mathematics. Springer, 1976.

2 Sergey V. Avgustinovich, Amy Glen, Bjarni V. Halldórsson, and Sergey Kitaev. On shortest crucial words avoiding Abelian powers. *Discrete Applied Mathematics*, 158(6):605–607, 2010.

3 Francine Blanchet-Sadri, Jane I. Kim, Robert Mercas, William Severa, and Sean Simmons. Abelian square-free partial words. In Adrian Horia Dediu, Henning Fernau, and Carlos Martín-Vide, editors, *LATA*, volume 6031 of *Lecture Notes in Computer Science*, pages 94–105. Springer, 2010.

4 Francine Blanchet-Sadri and Sean Simmons. Avoiding Abelian powers in partial words. In Giancarlo Mauri and Alberto Leporati, editors, *Developments in Language Theory*, volume 6795 of *Lecture Notes in Computer Science*, pages 70–81. Springer, 2011.

5 Peter Burcsi, Ferdinando Cicalese, Gabriele Fici, and Zsuzsanna Lipták. Algorithms for jumbled pattern matching in strings. *Int. J. Found. Comput. Sci.*, 23(2):357–374, 2012.

6 Sorin Constantinescu and Lucian Ilie. Fine and Wilf's theorem for Abelian periods. *Bulletin of the EATCS*, 89:167–170, 2006.

7 Maxime Crochemore, Costas Iliopoulos, Tomasz Kociumaka, Marcin Kubica, Jakub Pachocki, Jakub Radoszewski, Wojciech Rytter, Wojciech Tyczyński, and Tomasz Waleń. A note on efficient computation of all Abelian periods in a string. *Information Processing Letters*, 113(3):74–77, 2013.

8 James D. Currie and Ali Aberkane. A cyclic binary morphism avoiding Abelian fourth powers. *Theor. Comput. Sci.*, 410(1):44–52, 2009.

9 James D. Currie and Terry I. Visentin. Long binary patterns are Abelian 2-avoidable. *Theor. Comput. Sci.*, 409(3):432–437, 2008.

10 Michael Domaratzki and Narad Rampersad. Abelian primitive words. *Int. J. Found. Comput. Sci.*, 23(5):1021–1034, 2012.

11 P. Erdös. Some unsolved problems. *Hungarian Academy of Sciences Mat. Kutató Intézet Közl.*, 6:221–254, 1961.

12 Gabriele Fici, Thierry Lecroq, Arnaud Lefebvre, and Élise Prieur-Gaston. Computing Abelian periods in words. In Jan Holub and Jan Žďárek, editors, *Proceedings of the Prague Stringology Conference 2011*, pages 184–196, Czech Technical University in Prague, Czech Republic, 2011.

13 Gabriele Fici, Thierry Lecroq, Arnaud Lefebvre, Elise Prieur-Gaston, and William Smyth. Quasi-linear time computation of the abelian periods of a word. In Jan Holub and Jan Žďárek, editors, *Proceedings of the Prague Stringology Conference 2012*, pages 103–110, Czech Technical University in Prague, Czech Republic, 2012.

14 David Gries and Jayadev Misra. A linear sieve algorithm for finding prime numbers. *Commun. ACM*, 21(12):999–1003, December 1978.

15 Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.

16 Veikko Keränen. Abelian squares are avoidable on 4 letters. In Werner Kuich, editor, *ICALP*, volume 623 of *Lecture Notes in Computer Science*, pages 41–52. Springer, 1992.

17 Tanaeem M. Moosa and M. Sohel Rahman. Indexing permutations for binary strings. *Inf. Process. Lett.*, 110(18-19):795–798, 2010.

18 P. A. Pleasants. Non-repetitive sequences. *Proc. Cambridge Phil. Soc.*, 68:267–274, 1970.

Finding Pseudo-repetitions

Paweł Gawrychowski[*1], **Florin Manea**[†2], **Robert Mercaş**[‡3], **Dirk Nowotka**[§2], **and Cătălin Tiseanu**[4]

1 **Max-Planck-Institut für Informatik,**
Saarbrücken, Germany, `gawry@cs.uni.wroc.pl`
2 **Christian-Albrechts-Universität zu Kiel, Institut für Informatik,**
D-24098 Kiel, Germany, `{flm,dn}@informatik.uni-kiel.de`
3 **Otto-von-Guericke-Universität Magdeburg, Fakultät für Informatik,**
PSF 4120, D-39016 Magdeburg, Germany, `robertmercas@gmail.com`
4 **University of Maryland at College Park, Computer Science Department,**
A.V. Williams Bldg., College Park, MD 20742, USA, `ctiseanu@umd.edu`

Abstract

Pseudo-repetitions are a natural generalization of the classical notion of repetitions in sequences. We solve fundamental algorithmic questions on pseudo-repetitions by application of insightful combinatorial results on words. More precisely, we efficiently decide whether a word is a pseudo-repetition and find all the pseudo-repetitive factors of a word.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Stringology, Pattern matching, Repetition, Pseudo-repetition

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.257

1 Introduction

The notions of repetition and primitivity are fundamental concepts on sequences used in a number of fields, among them being stringology and algebraic coding theory. A word is a repetition (or power) if it equals a repeated catenation of one of its prefixes. We consider a more general concept here, namely *pseudo-repetitions in words*. A word w is a pseudo-repetition if it equals a repeated catenation of one of its proper prefixes t and its image $f(t)$ under some morphism or antimorphism (for short "anti-/morphism") f, thus $w \in t\{t, f(t)\}^+$.

Pseudo-repetitions, introduced in a restricted form by Czeizler et al. [3], lacked so far a well-developed algorithmic part. Given that the motivation for studying these objects originates from bioinformatics, where efficient algorithms are crucial, producing such tools seems not only natural but even necessary. This work is aimed to fill this gap. We investigate the following two basic algorithmic problems: decide whether a word w is a pseudo-repetition for an anti-/morphism f and find all k-powers of pseudo-repetitions occurring as factors in a word w, for an f as above; in these problems w is given as input, while f, although of unrestricted form, is fixed, thus not a part of the input. We establish algorithms and complexity bounds for these problems for various types of anti-/morphisms thereby improving significantly the results from [2]. Apart from the application of standard stringology tools, like suffix arrays, we extend the toolbox by nontrivial applications of results from combinatorics on words.

[*] Paweł Gawrychowski is supported by the *NCN* grant 2011/01/D/ST6/07164.
[†] Florin Manea is supported by the *DFG* grants 582014 and 596676.
[‡] Robert Mercaş is supported by the *Alexander von Humboldt Foundation*.
[§] Dirk Nowotka is supported by the *DFG Heisenberg* grant 590179.

Background and Motivation. The motivation of introducing pseudo-repetition and pseudo-primitivity in [3] originated from the field of computational biology, namely the facts that the Watson-Crick complement can be formalized as an antimorphic involution and both a single-stranded DNA and its complement (or its image through such an involution) basically encode the same information. Until now, pseudo-repetitions were considered only in the cases of involutions, following the original motivation, and the results obtained were mostly of combinatoric nature (e.g., generalizations of the Fine and Wilf theorem - see, e.g., [3, 8]).

A natural extension of these concepts is to consider anti-/morphisms in general, which is done in this paper. Considering that the notion of repetition is central in combinatorics of words and the plethora of applications that this concept has (see [7]), the study of pseudo-repetitions seems even more attractive, at least from a theoretical point of view. While the biological motivation seems appropriate only for the case of antimorphic involutions, the general problem of identifying pseudo-repetitions can be seen as a formalization of scenarios where we are interested in identifying sequences having a hidden repetitive structure. Indeed, as each pseudo-repetition is an iterated catenation of a factor and its encoding through some simple function, such words have an intrinsic, yet not obvious, repetitive structure.

Some Basic Concepts. For more detailed definitions we refer to [7].

Let V be a finite alphabet; V^* denotes the set of all words over V and V^k the set of all words of length k. The *length* of a word $w \in V^*$ is denoted by $|w|$. The *empty word* is denoted by λ. We denote by $\text{alph}(w)$ the alphabet of all letters that occur in w. A word $u \in V^*$ is a *factor* of $v \in V^*$ if $v = xuy$, for some $x, y \in V^*$; we say that u is a *prefix* of v, if $x = \lambda$, and a *suffix* of v, if $y = \lambda$. We denote by $w[i]$ the symbol at position i in w, and by $w[i..j]$ the factor of w starting at position i and ending at position j, consisting of the catenation of the symbols $w[i], \ldots, w[j]$, where $1 \leq i \leq j \leq n$; we define $w[i..j] = \lambda$ if $i > j$. Also, we write $w = u^{-1}v$ when $v = uw$. The powers of a word w are defined recursively by $w^0 = \lambda$ and $w^n = ww^{n-1}$ for $n \geq 1$. If w cannot be expressed as a nontrivial power of another word, then w is *primitive*. A *period* of a word w over V is a positive integer p such that $w[i] = w[j]$ for all i and j with $i \equiv j \pmod{p}$. By $per(w)$ we denote the smallest period of w.

The following classical result is extensively used in our investigation:

▶ **Theorem 1** (Fine and Wilf [4]). *Let u and v be in V^*. If two words $\alpha \in u\{u, v\}^+$ and $\beta \in v\{u, v\}^+$ have a common prefix of length greater than or equal to $|u| + |v| - \gcd(|u|, |v|)$, then u and v are powers of a common word of length $\gcd(|u|, |v|)$.*

A function $f : V^* \to V^*$ is a morphism if $f(xy) = f(x)f(y)$ for all $x, y \in V^*$; f is an antimorphism if $f(xy) = f(y)f(x)$ for all $x, y \in V^*$. In order to define a morphism or an antimorphism it is enough to give the definitions of $f(a)$ for all $a \in V$. An anti-/morphism $f : V^* \to V^*$ is an involution if $f^2(a) = a$ for all $a \in V$. We say that f is *uniform* if there exists a number k with $f(a) \in V^k$ for all $a \in V$; if $k = 1$ then f is called *literal*. If $f(a) = \lambda$ for some $a \in V$, then f is called *erasing*, otherwise *non-erasing*.

We say that a word w is an *f-repetition*, or, alternatively, an f-power, if w is in $t\{t, f(t)\}^+$, for some prefix t of w. If w is not an f-power, then w is *f-primitive*. As an example, the word $ACGTAC$ is primitive from the classical point of view (i.e., $\mathbf{1}$-primitive, where $\mathbf{1}$ is the identical anti-/morphism) as well as f-primitive for the morphic involution f defined by $f(A) = T$, $f(C) = G$, $f(T) = A$, and $f(G) = C$. However, for the antimorphic involution $f(A) = T$ and $f(C) = G$ (which is, in fact, a formalization of the Watson-Crick complement, from biology), we get that $ACGTAC = AC \cdot f(AC) \cdot AC$, thus, it is an f-repetition.

Finally, the computational model we use to design and analyse our algorithms is the standard unit-cost RAM (Random Access Machine) with logarithmic word size, which is generally used in the analysis of algorithms.

2 Algorithmic problems

In the upcoming algorithmic problems, we assume that the words we process are sequences of integers (called letters, for simplicity). In general, if the input word has length n then we assume its letters are in $\{1, \ldots, n\}$, so each letter fits in a single memory-word. This is a common assumption in algorithmics on words (see, e.g., the discussion in [6]).

In the first problem, which seems to us the most interesting one in the general context of pseudo-repetitions, we approach the fundamental problem of deciding whether a word is an f-repetition, for a fixed anti-/morphism f.

▶ Problem 1. Let $f : V^* \to V^*$ be an anti-/morphism. Given $w \in V^*$, decide whether this word is an f-repetition.

We solve this problem in the general case of erasing anti-/morphisms in $\mathcal{O}(n \lg n)$ time. However, in the particular case of uniform anti-/morphisms we obtain an optimal solution running in linear time. The latter covers the biologically motivated case of involutions from [3]. This optimal result seems interesting to us, as it shows that pseudo-repetitions can be detected as fast as repetitions, if the way we encode the repeated factor (i.e., the function f) is simple enough, yet not the identity. We also extend our results to a more general form of Problem 1, testing whether $w \in \{t, f(t)\}^+$ for a proper factor t of w. Except for the most general case (of erasing anti-/morphisms), where we solve this problem in $\mathcal{O}(n^{1+\frac{1}{\lg \lg n}} \lg n)$ time, we preserve the same time complexity as we obtained for Problem 1.

Two other natural algorithmic problems are related to the fundamental combinatorial property of freeness of words, in the context of pseudo-repetitions. More precisely, we are interested in identifying the factors of a word which are pseudo-repetitions.

▶ Problem 2. Let $f : V^* \to V^*$ be an anti-/morphism and $w \in V^*$ a given word.
(1) Enumerate all (i, j, ℓ), $1 \leq i, j, \ell \leq |w|$, such that there exists t with $w[i..j] \in \{t, f(t)\}^\ell$.
(2) Given k, enumerate all (i, j), $1 \leq i, j \leq |w|$, so there exists t with $w[i..j] \in \{t, f(t)\}^k$.

Question (2) was originally considered in [2], while the first one is its natural generalisation. Our approach to question (1) is based on constructing data structures which enable us to retrieve in constant time the answer to queries $rep(i, j, \ell)$: "Is there $t \in V^*$ such that $w[i..j] \in \{t, f(t)\}^\ell$?", for $1 \leq i \leq j \leq n$ and $1 \leq \ell \leq n$, where $n = |w|$. For unrestricted f, one can produce such data structures in $\mathcal{O}(n^{3.5})$ time. When f is non-erasing, the time taken to construct them is $\mathcal{O}(n^3)$, while when f is a literal anti-/morphism we can do it in time $\mathcal{O}(n^2)$. Once we have these structures, we can identify in $\Theta(n^3)$ time, in the general case, all the triples (i, j, ℓ) such that $w[i..j] \in \{t, f(t)\}^\ell$, answering (1) in $\mathcal{O}(n^{3.5})$ time. Similarly, for f non-erasing (respectively, literal) we answer question (1) in $\Theta(n^3)$ (respectively, $\Theta(n^2 \lg n)$) time and show that there are input words on which every algorithm solving this question has a running time asymptotically equal to ours (including the preprocessing time). Unfortunately, the time bound obtained for most general case is not tight.

Exactly the same data structures are used in the simplest case of literal anti-/morphisms to answer the more particular question (2). We obtain an algorithm that outputs in $\mathcal{O}(n^2)$ time, for given w and k, all pairs (i, j) such that $w[i..j] \in \{t, f(t)\}^k$; this time bound is shown to be tight. Taking advantage of the fact that k is given as input (so fixed throughout the algorithm) we can refine our solution for question (1) in order to get a $\Theta(n^2)$-time solution of question (2) for f non-erasing, again a tight bound, and a $\mathcal{O}(n^2 k)$-time solution for the general case. Our results improve significantly the algorithmic results reported in [2].

2.1 Prerequisites

We begin this section by presenting several number theoretic properties. Lemma 2 is used in the time complexity analysis of our algorithms, while Lemma 3 and its corollary are utilised in the solutions of Problem 2. Given two natural numbers k and n, we write $k \mid n$ if k divides n. We denote by $d(n)$ the number of divisors of n and by $\sigma(n)$ their sum.

▶ **Lemma 2.** *Let n be a natural number. The following statements hold:
(1) $\sum_{1\leq\ell\leq n} d(\ell) \in \Theta(n \lg n)$, $\sum_{1\leq\ell\leq n} d(\ell) \geq n \lg n$, $d(n) \in o(n^\epsilon)$ for all $\epsilon > 0$ (see [1]); (2) $\sigma(n) \in \mathcal{O}(n \lg \lg n)$ (see [1]); (3) $\sum_{1\leq\ell\leq n}(n-\ell+1)d(\ell) \in \Theta(n^2 \lg n)$.* ◀

▶ **Lemma 3.** *Let n be a natural number. We can compute in $\mathcal{O}(n^3)$ time a three dimensional array $T[k][m][\ell]$, with $1 \leq k, m, \ell \leq n$, where $T[k][m][\ell] = 1$ if and only if there exists a divisor s of ℓ and the numbers k_1 and k_2 such that $k_1 + k_2 = k$ and $k_1 s + k_2 sm = \ell$.* ◀

▶ **Corollary 4.** *Let R be a fixed natural constant, and n and k be given natural numbers. We can compute in $\mathcal{O}(n \lg n)$ time a matrix $T_k[m][\ell]$ with $1 \leq m \leq R$ and $1 \leq \ell \leq n$, where $T_k[m][\ell] = 1$ if and only if there exists a divisor s of ℓ and the numbers k_1 and k_2 such that $k_1 + k_2 = k$ and $k_1 s + k_2 sm = \ell$. The constant hidden by the $\mathcal{O}$-notation depends on R.* ◀

We briefly present the data structures we use. For a word u with $|u| = n$ over $V \subseteq \{1, \ldots, n\}$ we can build in linear time a suffix array structure as well as data structures allowing us to return in constant time the answer to queries "How long is the longest common prefix of $u[i..n]$ and $u[j..n]$?", denoted $LCPref(u[i..n], u[j..n])$. For more details, see [5, 6], and the references therein. Also, for u and an anti-/morphism f, we compute an array len with n elements defined as $len[i] = |f(u[1..i])|$, for $1 \leq i \leq n$. For f non-erasing we also compute an array inv, having $|f(u)|$ elements, such that $inv[i] = j$ if $len[j] = i$ and $inv[i] = -1$ otherwise. These computations are done in $\mathcal{O}(n)$ time. Note the following result:

▶ **Lemma 5.** *Let $w \in V^*$ be a word of length n. We compute the values $per[i]$, the period of $w[1..i]$, for all $i \in \{1, \ldots, n\}$ in linear time $\mathcal{O}(n)$. Also, we compute the values $per[i][j]$, the period of $w[i..j]$, for all $i, j \in \{1, \ldots, n\}$ in quadratic time $\mathcal{O}(n^2)$.* ◀

Next we show an important property of pseudo-repetitions, for non-erasing morphisms.

▶ **Lemma 6.** *Let f be a non-erasing anti-/morphism, and x, y, z be words over V such that $f(x) = f(z) = y$. If $\{x, y\}^* x \{x, y\}^* \cap \{z, y\}^* z \{z, y\}^* \neq \emptyset$ then $x = z$.*

Proof. We sketch the proof only for the case when f is a morphism; a similar argument works for antimorphisms. If $\{x, y\}^* x \{x, y\}^* \cap \{z, y\}^* z \{z, y\}^* \neq \emptyset$ then we may assume without losing generality there exists w such that $w = xw'$, $w' \in \{x, y\}^*$, and $w \in \{z, y\}^* z \{z, y\}^*$.

If z is a prefix of w, as $f(x) = f(z)$ and f is non-erasing, we get easily that $x = z$.

Assume now that $w = yzw''$ with $w'' \in \{z, y\}^*$. It is not hard to see that from $|x| \leq |y|$ and $w = xw'$ we obtain that $|x|$ is a period of y, and, thus, $y = x^\ell u$ where $\ell > 0$ and u is a prefix of x. If y and x are powers of the same word v, then $x = v^{k_1}$, $y = v^{k_2}$ and $u = v^{k_3}$, so z is also a power of v. Since $f(x) = f(z)$ we conclude again that $x = z$. Further, assume that x and y are not powers of the same word. Hence, u is a proper prefix of x, i.e., $x = uv$ for $u \neq \lambda \neq v$. Consequently, w' has a prefix of the form $x^p y$, with $p \geq 0$, and it follows that after the first $|y|$ symbols of w both the factor vu and the factor z occur (as vu occurs after the first $|y| - |x|$ symbols of w'). Since $|vu| = |x|$ we get easily that $z = vu$. So, $|z| = |x|$, $y = f(z) = f(vu) = f(v)f(u)$ and $y = f(x) = f(u)f(v)$. It follows that y is a power of a primitive word t. By an involved case analysis, it follows that x is a power of the same primitive word as y, a contradiction.

In the case when $w = yyzw''$ for some $w'' \in \{z, y\}^*$, we can apply Theorem 1 to the prefix of length $2|y|$ of w (which is a prefix of a word from $x\{x, y\}^*$, as well) and obtain that x and y are powers of the same word. Once again, we obtain that $z = x$. ◀

The next lemmas provide insights to the combinatorial properties of f-repetitions, for f a general morphism, and are utilised in showing the soundness and efficiency of our algorithms. When using them, we take x to be the shorter and y the longer of the words t and $f(t)$.

▶ **Lemma 7.** *Let x and y be words over V such that x and y are not powers of the same word. If $w \in \{x, y\}^*$ then there exists a unique decomposition of w in factors from $\{x, y\}$.* ◀

▶ **Lemma 8.** *Let $x, y \in V^+$ and $w \in \{x, y\}^* \setminus \{x\}^*$ be words such that $|x| \leq |y|$ and x and y are not powers of the same word. Let $M = \max\{p \mid x^p$ is a prefix of $w\}$ and $N = \max\{p \mid x^p$ is a prefix of $y\}$. Then $M \geq N$. Moreover, if $M = N$ then $w \in y\{x, y\}^*$ holds, while if $M > N$ then either it is the case that $w \in x^{M-N}y\{x, y\}^* \setminus x^{M-N-1}yxV^*$, or we have $w \in x^{M-N-1}y\{x, y\}^+ \setminus x^{M-N}yV^*$ and $N > 0$.* ◀

2.2 Solution of Problem 1

§A general solution. We first assume that f is a morphism and let $n = |w|$. We construct in linear time the word $x = wf(w)$ of length $m = n + |f(w)|$ (which is in $\mathcal{O}(n)$); note that the length of x (hence, the constant hidden by the $\mathcal{O}$-notation) depends on the fixed morphism f. Moreover, we build in $\mathcal{O}(n)$ time data structures enabling us to answer *LCPref* queries for x.

Using these data structures, Algorithm 1 tests whether w is an f-repetition or not.

Algorithm 1 *Test*(w, f): decides whether w is an f-repetition

```
1:  Test whether there exists a word $x$ such that $w = x^k$, with $k \geq 2$. If yes, then we halt
    and decide that $w$ is an $f$-repetition. Otherwise, go to step 2.
    {If the result of the test is positive we decide that $w$ is an $f$-repetition, as repetitions can
    be seen as trivial $f$-repetitions. The algorithm continues for $w$ primitive.}
2:  for $t = w[1..i]$, such that $i < n$, $len[i] \geq 1$, $t$ and $f(t)$ are not powers of some $x \in V^*$ do
3:      Set $x = t$ and $y = f(t)$ if $i \leq len[i]$ or $x = f(t)$ and $y = t$, otherwise;
4:      Set $s = i + 1$, $\ell'_i = |y|$, $\ell''_i = |x|$; {We have $\ell'_i = \max\{len[i], i\}$ and $\ell''_i = \min\{i, len[i]\}$}
5:      If $s = n + 1$ halt and decide that $w$ is an $f$-repetition;
6:      Compute $M = \max\{p \mid x^p$ is a prefix of $w[s..n]\}$, $N = \max\{p \mid x^p$ is a prefix of $y\}$;
7:      If $w[s..n] = x^M$, set $s = n + 1$, go to step 5;{If $w[s..n] \in \{x\}^+$ then $w \in t\{t, f(t)\}^*$}
8:      If $x^{M-N}y$ occurs at position $s$, set $s = (M - N)\ell''_i + \ell'_i$, go to step 5;
9:      If $M > N$ and $x^{M-N-1}yx$ occurs at position $s$, set $s = (M - N - 1)\ell''_i + \ell'_i$, go to step 5;
        {By Lemma 8, $w[s..n]$ should have either $x^{M-N}y$ or $x^{M-N-1}yx$ as prefix. By Lemma 8,
        if $x^{M-N-1}yx$ occurs at position $s$, we shall check whether $w[s..n] \in x^{M-N-1}y\{x, y\}^+$.}
        {If none of the above holds, we get that $w[s..n] \notin \{t, f(t)\}^+$, so $w \notin t\{t, f(t)\}^+$.}
10: end for
11: Halt and decide that $w$ is not an $f$-repetition.
```

Following the comments inserted in its description, it is not hard to see that Algorithm 1 is sound. In the following, we compute its complexity. The step where we test whether w is a repetition takes $\mathcal{O}(n)$ time, as it can be done by locating the occurrences of w in ww. Further, note that the computation in each of the steps $6 - 9$ of the algorithm can be executed in constant time using the data structures we already constructed. Indeed, for some $s \leq n$, we can compute the largest ℓ such that $w[s..\ell]$ is a power of x in constant time as follows. In the worst case, $\ell = s-1$, or, in other words, $w[s..\ell] = \lambda$, when x does not occur at position s. Otherwise, ℓ is the largest number less than or equal to $\mathit{LCPref}(w[s..n], w[s+|x|..n])$ such that $\ell - s + 1$ is divisible by $|x|$. This strategy is used in step 6 to compute M and N. The verification from step 7 takes clearly constant time: we just check whether $n - s + 1 = M|x|$. Moreover, step 8 and 9 can also be implemented in constant time using *LCPref* queries; indeed, we know that x^{M-N} occurs at position s, and then we just have to check whether y occurs at position $s + (M-N)|x|$ by a *LCPref* query, for step 8, or, respectively, whether yx occurs at position $s+(M-N-1)|x|$ by two *LCPref* queries, for step 9. Further, the iterative process in steps $3-9$ is executed for each prefix $w[1..i]$ of w, and during each iteration the algorithm makes at most $\mathcal{O}(\lfloor \frac{n}{\ell'_i} \rfloor)$ steps, as s can take at most $\lfloor \frac{n}{\ell'_i} \rfloor$ different values (in the cycle defined by the "go to" instruction from step 8). Since $\ell'_i \geq i$, the overall time complexity of the algorithm is upper bounded by $\mathcal{O}(\sum_{1\leq i\leq n} \lfloor \frac{n}{i} \rfloor)$. Thus, the time complexity of Algorithm 1 is $\mathcal{O}(n \lg n)$. As a side note, in the case when f is erasing, $w \in t\{t, f(t)\}^+$ for some t with $f(t) = \lambda$ if and only if $w \in \{t\}^+$, that is, w is a repetition. Hence, we run the iterative process starting in step 2 only for prefixes $w[1..i]$ with $\mathit{len}[i] \geq 1$.

The case when f is an antimorphism is similar. We take $x = wf(w)$, build the same data structures, and proceed just as in the former case. As the single difference, now we have $w[s+1..s+\mathit{len}[i]] = f(w[1..i])$ iff $\mathit{LCPref}(s+1, m - \mathit{len}[i] + 1) = \mathit{len}[i]$, where $m = |x|$.

When f is uniform we can easily obtain a more efficient algorithm. In this case, $|t|$ divides n, so we only need to run the iterative instruction for the prefixes $w[1..i]$ of w with $i \mid n$. Hence, the total running time of the algorithm is, in this case, upper bounded by $\mathcal{O}(\sum_{i|n} \frac{n}{i}) \in \mathcal{O}(n \lg\lg n)$, by Lemma 2.

§A linear time solution for the case when f is uniform. We can obtain an even faster solution for Problem 1 for the case when f is uniform by using some more intricate precomputed data structures in order to speed-up Algorithm 1. To this end, we analyse again the computation performed by Algorithm 1 on an input word w.

The main phase of the algorithm is the following. For a prefix $t = w[1..i]$ of w with $i \mid n$ we run a cycle (steps $5-9$) that extends iteratively a prefix $w[1..s-1]$, where $s \geq i+1$, of the word w such that the newly obtained prefix is in $t\{t, f(t)\}^*$. However, at each iteration the prefix is extended with a word of the form $t^k f(t)$, with $k \geq 0$. As k can be actually equal to 0, we can only say that the number of iterations of the cycle is upper bounded by $\frac{n}{|f(t)|} \leq \frac{n}{|t|}$. Here we plug in our speed-up strategy: we try to extend the prefix in each of the iterations of the cycle from steps $5-9$ with a word that belongs to $\{t, f(t)\}^\alpha$ for some fixed number α that depends on n, but not on t. In this way, we upper bound the number of iterations of the cycle by $\frac{n}{\alpha|t|}$, and the overall complexity of the algorithm by $\mathcal{O}(\frac{n \lg\lg n}{\alpha})$. Finally, in order to obtain an algorithm solving Problem 1 in linear time, we choose $\alpha = \lceil \lg\lg n \rceil$.

Let $R = |f(a)|$, for $a \in \mathrm{alph}(w)$; as f is uniform, the definition of R does not depend on the choice of a from V, and we also have $R = \frac{|f(u)|}{|u|}$, $\forall u \in V^*$. Let $r_t = \max\{\ell \mid t^\ell \text{ prefix of } f(t)\}$. Clearly, $r_t \leq R$ and we can assume without losing generality that $\alpha > R$. Indeed, this holds for $n > 2^{2^R}$, which is the case when we want to optimise Algorithm 1; for smaller n the algorithm runs in constant time $\mathcal{O}(1)$, as $n \lg\lg n \leq R2^{2^R}$ and R is constant (f being fixed).

It only remains to show how we can implement efficiently the above mentioned extension of the prefix. First, note that there exists a constant C such that $\frac{(\lg n)^4 (\lg\lg n)^2}{n} \leq C$ for all n. Therefore, running the original form of Algorithm 1 for the prefixes t of w with $|t| > \frac{n}{(\lg n)^2 \lg\lg n}$ and $|t| \mid n$ (that is, at most $(\lg n)^2 \lg\lg n$ prefixes) takes $\mathcal{O}(n)$ time. Therefore, from now on, we only consider prefixes t such that $|t| \mid n$, $|t| < \frac{n}{(\lg n)^2 \lg\lg n}$, and, assuming that the input word is not a repetition, t and $f(t)$ are not powers of the same word.

Now consider a prefix t, as above. There are $2^\alpha \in \mathcal{O}(\lg n)$ words in $\{t, f(t)\}^\alpha$. Every such word can be encoded by a bit-string of length α: each occurrence of t is encoded by 0 and an occurrence of $f(t)$ by 1. Denote these bit-strings $v_1, \ldots, v_{2^\alpha}$, and let $\overline{v}_i$ be the word encoded by v_i, for all $1 \leq i \leq 2^\alpha$. Further, for a bit-string v_ℓ we can determine by binary search two values b_ℓ and e_ℓ such that all the suffixes contained in the suffix array of w between the positions b_ℓ and e_ℓ have the word $\overline{v}_\ell t^{r_t}$ as a prefix. From Theorem 1, applied for two strings $\overline{v}_i t^{r_t}$ and $\overline{v}_j t^{r_t}$ with $i \neq j$, and the facts that t and $f(t)$ are not powers of the same word and r_t is the maximal power of t occurring as a prefix of $f(t)$, we get that the intervals $[b_i, e_i]$ and $[b_j, e_j]$ are disjoint. The time needed to compute these values for each ℓ is $\mathcal{O}(\lg n \lg \lg n)$, as a comparison between the word $\overline{v}_\ell t^{r_t}$ and a suffix of w can be done in $\mathcal{O}(\lg \lg n)$ by looking at the encoding v_ℓ and the string t^{r_t} (a prefix of $f(t)$) and, consequently, comparing only the factors of length $|t|$ and $|f(t)|$ of $\overline{v}_\ell t^{r_t}$ with those of the words from the suffix array. Thus, the time needed to compute b_ℓ and e_ℓ for all ℓ is $\mathcal{O}((\lg n)^2 \lg \lg n)$. Next, we construct a set E_t containing the values e_ℓ ordered increasingly, while keeping track for each e_ℓ of the corresponding values of ℓ and b_ℓ. Note that E_t contains $\mathcal{O}(\lg n)$ integers from $\{1, \ldots, n\}$.

We need one more result before concluding this preprocessing phase. We want to store a static set $S \subseteq \{1, \ldots, n\}$ so that finding the successor in S of a given $x \in \{1, \ldots, n\}$ takes constant time. Thus, we use a static d-ary tree of depth 2, where $d = \lceil n^{0.5} \rceil$, so that the whole tree has n leaves corresponding to different values of x. We mark all leaves corresponding to the elements of S, and remove all nodes with no marked leaf in the corresponding subtree. At each remaining inner node v we store a table of length d where for each child of v (both remaining and already removed) we store the successor of the rightmost leaf in its corresponding subtree. The total size of the structure is $\mathcal{O}(|S| n^{0.5})$ and we can construct it in the same time if we start with an empty S and add its elements one-by-one, creating new inner nodes when necessary. Furthermore, using the tables we can find the successor of any x in $\mathcal{O}(1)$ time by traversing the path from the root of the tree towards x as long as the nodes exist and taking the minimum of the successors stored for these nodes. If we store each E_t in this way the query time is constant and the total construction time and space is in $\mathcal{O}(d(n) n^{0.5} \lg n) \subseteq \mathcal{O}(n)$, where the final upper bound follows from Lemma 2.

By the previously given explanations, this entire preprocessing takes linear time. We now use it to solve in linear time Problem 1.

Assume now that we run step 5 of the algorithm for some prefix t of w as above and the word $w[s..n]$ with $s \leq n - (\alpha + r_t)|t| + 1$. There is at most one ℓ such that the index i_s of $w[s..n]$ in the suffix array of w is between b_ℓ and e_ℓ (that is, $\overline{v}_\ell t^{r_t}$ is a prefix of $w[s..n]$). This ℓ can be found, if it exists, in $\mathcal{O}(1)$ using the precomputed data structures (i.e., the sets E_t, organised as described above): return the value ℓ such that e_ℓ is the minimal element of E_t greater than or equal to i_s and $b_\ell \leq i_s$. Then, we repeat the procedure for the word $w[s'..n]$ where $w[s..s'-1] = \overline{v}_\ell$, but only if $n - s' + 1 \geq (\alpha + r_t)|t|$ or $s' = n+1$. If $n - s' + 1 \leq (\alpha + r_t)|t|$ we run the processing of the original algorithm. Clearly, this process takes $\mathcal{O}(\frac{n}{\alpha t} + 2\alpha)$ steps for each t, so the complete algorithm runs in $\mathcal{O}(n)$ time. We only have to show that it works correctly, i.e., it decides whether $w \in t\{t, f(t)\}^+$. The soundness is proven by the following remark. If $w[s..n]$ starts with $\overline{v}_j t^{r_t}$ for some $j \leq 2^\alpha$, then it is enough to consider in the next iteration only the word $w[s + |\overline{v}_j|..n]$, and no other word $w[s + |\overline{v}_k|..n]$ where $k \leq 2^\alpha$ such that $\overline{v}_k$ is also a prefix of $w[s..n]$. Indeed, if there exists v_k leading to a solution, we get a contradiction with either the fact that r_t is the maximal power of t occurring as a prefix of $f(t)$, or with the fact that t and $f(t)$ are not powers of the same word.

To conclude, this implementation of Algorithm 1 runs in optimal linear time for f uniform.

§Summary. We were able to show the following theorem.

▶ **Theorem 9.** *Let $f : V^* \to V^*$ be an anti-/morphism. Given $w \in V^*$, one can decide whether $w \in t\{t, f(t)\}^+$ in $\mathcal{O}(n \lg n)$ time. If f is uniform we only need $\mathcal{O}(n)$ time.* ◀

The more general problem of testing whether there exists t with $w \in \{t, f(t)\}\{t, f(t)\}^+$ for f a fixed anti-/morphism is also worth considering. Solving this problem seems to require a different strategy than the one in Algorithm 1. There we take prefixes t of w, which determine uniquely $f(t)$, and check whether $w \in t\{t, f(t)\}^*$. Here, a prefix y does not determine uniquely, in general, a factor x such that $f(x) = y$, so more possibilities have to be considered when checking whether there exists t such that $w \in f(t)\{t, f(t)\}^*$. However, the cases of f non-erasing and uniform anti-/morphisms have solutions based on results in the line of Lemmas 6 and 7, leading to similar complexities as for Problem 1. The case of erasing anti-/morphisms is solved by a more involved algorithm, based on both combinatorics on words and number theoretic insights.

▶ **Theorem 10.** *Let $f : V^* \to V^*$ be an anti-/morphism. Given $w \in V^*$, we decide whether $w \in \{t, f(t)\}\{t, f(t)\}^+$ in $\mathcal{O}(n^{1+\frac{1}{\lg \lg n}} \lg n)$ time. If f is non-erasing we solve the problem in $\mathcal{O}(n \lg n)$ time, while when f is uniform we only need $\mathcal{O}(n)$ time.* ◀

2.3 Solution of Problem 2

Recall that our approach to solve the first question of Problem 2 is based on constructing, for the input word w, data structures that enable us to obtain in constant time the answer to queries $rep(i, j, \ell)$: "Is there $t \in V^*$ such that $w[i..j] \in \{t, f(t)\}^\ell$?", for all $1 \leq i \leq j \leq |w|$ and $1 \leq \ell \leq |w|$. Moreover, a solution for the second question is derived directly from this strategy: we only need to construct similar data structures, that allow us to answer, this time, queries $rep(i, j, \ell)$ for a single ℓ, given as input of the problem together with w.

§The case of erasing morphisms. We start by presenting the solution of the first question of the problem. Given an arbitrary anti-/morphism f and a word w of length n, we can construct the aforementioned data structures in $\mathcal{O}(n^{3.5})$ time. More precisely, we construct an oracle-structure that already contains the answers to every possible query.

We only give an informal description of our construction. Assume that $|w| = n$. The idea is to compute the $n \times n \times n$ three dimensional array M such that $M[i][j][k] = 1$ if there exists a word t with $w[i..j] \in \{t, f(t)\}^k$, and $M[i][j][k] = 0$, otherwise. We proceed as follows.

Let i be a position in w. We first consider the prefixes t of $w[i..n]$ such that t and $f(t)$ are not powers of the same word. Note that, for such a prefix t of $w[i..n]$, with $t \neq \lambda \neq f(t)$, and $j > i$ there is at most one number k such that $w[i..j] \in t\{t, f(t)\}^{k-1}$. The set of these prefixes is partitioned in $n^{0.5} + 1$ sets $S_{i,\delta} = \{t \mid |f(t)| = \delta\}$, for $1 \leq \delta \leq n^{0.5}$, and $S_i = \{t \mid |f(t)| > n^{0.5}\}$; note that some of these sets may actually be empty. Further, for each δ we compute $f_{i,\delta} = \max\{k \mid x^k \text{ is a suffix of } w[1..i], |x| = \delta\}$.

We first deal with the case when $t \in S_i$, for $1 \leq i \leq n$. We compute for each j the number k such that $w[i..j] \in t\{t, f(t)\}^{k-1}$; this can be done in constant time (for each j) using *LCPref*-queries, as in the previous algorithms. More precisely, for some j we only need to look at the corresponding value for $j - |t|$ and $j - |f(t)|$, increase them with 1 (if they are defined) and store as the value corresponding to j the one obtained from $j - |t|$ if t occurs as a suffix of $w[i..j]$ or the one corresponding to $j - |f(t)|$ if $f(t)$ occurs as a suffix of $w[i..j]$ (due to Lemma 7, at most one case holds); if none of these values was defined, or neither t nor $f(t)$ occurs as a suffix of $w[i..j]$, the value corresponding to j remains undefined.

This entire process takes linear time. Then, for j such that $w[i..j] \in t\{t, f(t)\}^{k-1}$ and all $k' \in \{0, 1, \ldots, f_{i,\delta}\}$, where $\delta = |f(t)|$, we set $M[i - k'\delta][j][k + k'] = 1$. It is not hard to see that for $\delta > n^{0.5}$ we have $f_{i,\delta} < n^{0.5}$, so the process described above takes $\mathcal{O}(n^{0.5})$ time for each j. Now, we repeat the process for all $i \in \{1, \ldots, n\}$ and all prefixes t from S_i and discover all the factors $w[i'..j']$ and numbers k such that $\{f(t), t\}^k$, with $|f(t)| > n^{0.5}$. The time needed to do the computations described above is $\mathcal{O}(n^{3.5})$.

Further, we consider the case of the words of the sets $S_{i,\delta}$, for some fixed $\delta < n^{0.5}$ and all $1 \leq i \leq n$. For each i, for each t in $S_{i,\delta}$, and for each j we compute and put the pairs (i, k) such that $w[i..j] \in t\{t, f(t)\}^{k-1}$ in a list R^δ_j. This takes roughly $\mathcal{O}(n^3)$ time. Note that the number of elements of the list R^δ_j is also bounded by n^2, as for each i we have a unique decomposition of $w[i..j]$ in k parts, starting with a prefix t.

Now, for each j (and, recall, that δ is fixed), we build an $n \times n$ matrix T^δ_j, initially with all the entries set to 0. Now we partition this matrix in *diagonal arrays* obtained as follows: for ℓ from 1 to n and for p from 1 to n, if the element $T^\delta_j[\ell][p]$ is not stored already in a diagonal array, we construct a new diagonal array that stores the elements $T^\delta_j[\ell][p], T^\delta_j[\ell - \delta][p+1], \ldots$ $T^\delta_j[\ell - d\delta][p + d]$, for $0 \leq d < \frac{\ell}{\delta}$. While constructing this matrix we can keep track for each element of the array it belongs to. This procedure takes, clearly, $\mathcal{O}(n^2)$ time. These arrays partition the elements of the matrix T^δ_j so the total number of their elements is n^2.

To continue, for each element (i, k) of the list R^δ_j, we check in which diagonal array (i, k) is and memorise that we should mark (i.e., set to 1) in this array the consecutive elements $T^\delta_j[i][k], T^\delta_j[i - \delta][k + 1], \ldots, T^\delta_j[i - f_{i,\delta}\delta][k + f_{i,\delta}]$. This, again, can be done in $\mathcal{O}(n^2)$ time, as we only need to memorise the first and the last of these elements (called, in the following, margins). When we are done we have to mark r_d groups of consecutive elements in each diagonal array d, where $\sum_d r_d \in \mathcal{O}(n^2)$. To do the marking we sort the margins of the groups associated with each diagonal array, with the counting sort algorithm, and then traverse each array, keeping track of how many groups contain each of its elements, and mark the elements appearing in at least one group. Sorting the lists of intervals takes $\mathcal{O}(\sum_d r_d) = \mathcal{O}(n^2)$ time, and, thus, the marking takes $\mathcal{O}(n^2)$ time in total. Once the elements of all groups are marked, for all i and k we set $M[i][j][k] = 1$ if and only if $T^\delta_j[i][k] = 1$.

The overall complexity of the computation described above for a fixed δ is $\mathcal{O}(n^3)$. As we iterate through all $\delta \leq n^{0.5}$, we get that this case requires $\mathcal{O}(n^{3.5})$, as well. Now, we know all triples (i, j, k) such that $w[i..j] \in \{t, f(t)\}^k$ and t and $f(t)$ are not powers of the same word.

Further, we consider the case of triples (i, j, k) such that $w[i..j] \in \{t, f(t)\}^k$, where t and $f(t)$ are powers of the same word. By Lemma 5 we compute in $\mathcal{O}(n^2)$ time the periods of all the factors $w[i..j]$ of w and of the factors $f(w[i..j])$ of $f(w)$. We also compute in cubic time the array T from Lemma 3. Now we can check in constant time using *LCPref* queries whether $per(t) = p$, $p \mid |t|$, and $f(w[i..i+p-1])$ is a power of $w[i..i+p-1]$ (i.e., t and $f(t)$ are powers of the same word). If this is the case, we compute $m = \frac{|f(w[i..i+p-1])|}{p}$ and set $M[i][j][k] = 1$ if and only if $T[k][m][j - i + 1] = 1$. Indeed, $M[i][j][k] = 1$ if and only if there exists s, k_1, k_2 such that $s \mid j - i + 1$, $k_1 + k_2 = k$, and $w[i..j] = ((w[i..i + p - 1])^s)^{k_1}(f((w[i..i + p - 1])^s))^{k_2}$, that is, $sk_1 + smk_2 = j - i + 1$, which is equivalent to $T[k][m][j - i + 1] = 1$

There is a simple case that remained to be discussed. If $f(w[i..j]) = \epsilon$, then $M[i][j][k] = 1$, for all $k \geq 1$. Identifying and memorising all such factors takes $\mathcal{O}(n^3)$ time.

By the above case analysis, we conclude that we can compute all the non-zero entries of the matrix M in $\mathcal{O}(n^{3.5})$ time. The answer to $rep(i, j, k)$ is given by the entry $M[i][j][k]$.

Finally, we consider the case when we search f-repetitions with k factors, for a fixed k. This time, we compute a two dimensional matrix M_k such that $M_k[i][j] = M[i][j][k]$, defined previously. Fortunately, M_k can be computed much quicker than the whole matrix M.

According to Corollary 4 the case of t and $f(t)$ being factors of the same word can be implemented in quadratic time (the constant R from the statement of the corollary can be taken as the maximum length of $f(a)$, for all letters $a \in \text{alph}(w)$). Further, when t and $f(t)$ are not periods of the same word we just need to compute, for each i, t and j the number k' such that $w[i..j] \in t\{t, f(t)\}^{k'-1}$ and check (in constant time) whether $f(t)^{k-k'}$ is a suffix of $w[1..i]$; if all these hold, we get that $M_k[i][j] = 1$. However, note that we do not need to go through all the possible values of j. Indeed, we first generate all the prefixes of $w[i..n]$ that have the form t^ℓ with $\ell \leq k$ and see if one of them is longer than $|t| + |f(t)|$. If yes, we try to extend the longest such prefix with t or $f(t)$ iteratively until we use k factors t or $f(t)$ in the constructed word. By Lemma 7 we obtain in this process only $\mathcal{O}(k)$ such words, and these are exactly the prefixes of $w[i..n]$ that can be expressed as the catenation of at most k factors t and $f(t)$; in other words, this process provides a set that contains all the values j for which $M_k[i][j] = 1$. According to these, the whole process of computing the non-zero entries of the matrix M' takes $\mathcal{O}(n^2 \cdot k)$ time. Note that the answer to a query $rep(i, j, k)$ is given by $M_k[i][j]$; as we already mentioned, we only ask queries for the value k given as input.

§The case of non-erasing morphisms. For f non-erasing, the oracle matrix M described previously can be computed in $\mathcal{O}(n^3)$ time, where $|w| = n$. Initially, we set $M[i][j][k] = 0$, for $i, j, k \in \{1, \dots, n\}$.

As in the case of erasing morphisms, by Lemma 5 we compute (and store) in quadratic time the periods of all the factors $w[i..j]$ of w and of the factors $f(w[i..j])$ of $f(w)$. We also compute in cubic time the array T from Lemma 3.

First we analyse the simplest case. We can check in constant time using *LCPref* queries whether $per(w[i..j]) = p$, $p \mid (j-i+1)$, and $f(w[i..i+p-1])$ is a power of $w[i..i+p-1]$. If so, we compute $m = \frac{|f(w[i..i+p-1])|}{p}$ and set $M[i][j][k] = 1$ if and only of $T[k][m][j-i+1] = 1$.

Further we present the more complicated cases.

First, let i be a number from $\{1, \dots, n\}$. We want to detect the factors $w[i..j]$ that belong to $t\{t, f(t)\}^{k-1}$ for some prefix t of $w[i..n]$ such that t and $f(t)$ are not powers of the same word (this case was already covered) and $k \geq 2$. To do this we try all the possible prefixes t of $w[i..n]$. Once we choose such a $t = w[i..\ell]$ we set $M[i][\ell][1] = 1$. Further, starting from the pair $(\ell, 1)$, we compute, by backtracking, all the pairs (m, e) such that $w[i..m] \in t\{t, f(t)\}^{e-1}$; basically, from the pair (m, e) we obtain the pairs $(m + |t|, e + 1)$ if $w[m + 1..m + |t|] = t$ and the pair $(m + |f(t)|, e + 1)$ if $w[m + 1..m + |f(t)|] = f(t)$. By Lemma 7 we obtain exactly one pair of the form $(m, \cdot)$ (as there is an unique decomposition of $w[i..m]$ into factors t and $f(t)$ as long as t and $f(t)$ are not powers of the same word). Therefore, computing all these pairs takes linear time. Further, if we obtained the pair (m, k) we set $M[i][m][k] = 1$.

The whole process just described can be clearly implemented in $\mathcal{O}(n^3)$ time. At this point we know all the possible triples (i, j, k) such that $w[i..j] \in t\{t, f(t)\}^{k-1}$ for some t. It remains to find also the triples (i, j, k) such that $w[i..j] \in f(t)\{t, f(t)\}^{k-1}$ for some t.

In this case, for each $i \in \{1, \dots, n\}$ we go through all the prefixes $y = w[i..\ell]$ of $w[i..n]$ and assume that $y = f(t)$. Further, we compute a set of pairs (m, e) such that $w[i..m] = y^e$; this can be done easily in linear time, using *LCPref*-queries. Now, for each of these pairs, say (m, e), we try to find a factor $t = w[m + 1..m']$ such that $f(t) = y$ and t and y are not powers of the same word. Once we found such a factor t (which can be done in constant time using *LCPref* queries and the array *inv*) we store the pair $(m + |t|, e + 1)$ and starting from this pair we compute, as in the previous case, all the pairs (m'', e') such that $w[m+1..m''] \in t\{t, y\}^{e'-e-1}$. The key remark regarding this process is that, by Lemma 6, no two pairs having the first component equal to m'' are obtained for a fixed i. As the number of values that m'' may take is upper bounded by n, the entire computation of these pairs takes $\mathcal{O}(n)$ time. Once this is completed, we set $M[i][m][k] = 1$ for each (m, k) obtained.

In this way we identified all the triples (i, j, k) such that $w[i..j] \in \{t, f(t)\}^k$, for some t, in cubic time and stored in the array M the answers to all the possible *rep*-queries.

Now, consider the case when we search f-repetitions with k factors, for a given k and f non-erasing. The computation goes on exactly as in the case of general morphisms with the only difference that when we consider the prefix t of a word $w[i..n]$ we can restrict our search to the prefixes t shorter than $\frac{n}{k}$. Thus, the overall complexity of computing the entries of the matrix M_k decreases to $\mathcal{O}(n \cdot \frac{n}{k} \cdot k) = \mathcal{O}(n^2)$ time. Again, the answers to a query $rep(i, j, k)$ for the given value k is given by the entry $M_k[i][j]$ of the matrix M_k.

§The case of literal morphisms. In the case when f is literal, we are able to construct faster some data structures enabling us to answer *rep* queries. More precisely, we do not need to construct the entire oracle structure, but only some less complex matrix allowing us to retrieve in constant time the answers to our queries. To this end, we first create for the word $wf(w)$ the same data structures as in the initial solution of Problem 1. Further, we define an $n \times n$ matrix M such that for $1 \leq i, d \leq n$ the element $M[i][d] = (j, i_1, i_2)$ stores the beginning index of the longest word $w[j..i]$ contained in $\{t, f(t)\}^+$ for some word t of length d, as well as the last occurrences $w[i_1..i_1 + |t| - 1]$ of t and $w[i_2..i_2 + |f(t)| - 1]$ of $f(t)$ in $w[j..i]$, such that d divides both $i - i_1 + 1$ and $i - i_2 + 1$. If there exist t and t' with $t \neq t'$ and $w[j..i] \in \{t, f(t)\}^k \cap \{t', f(t')\}^k$, we have $t = f(t')$ and $f(t) = t'$; in this case, $M[i][d]$ equals (j, i_1, i_2) if $i_1 > i_2$ or (j, i_2, i_1), otherwise. The array M can be computed in $\mathcal{O}(n^2)$ time by dynamic programming. Intuitively, $M[i][d]$ is obtained in constant time from $M[i - d][d]$ using *LCPref* queries on $wf(w)$.

The matrix M helps us answer *rep*-queries in constant time. Indeed, the answer to a query $rep(i, j, k)$ is yes if and only if $k \mid j - i + 1$ and the first component of the triple $M[j][\frac{j-i+1}{k}]$ is lower than or equal to i, and no, otherwise.

§Solving Problem 2. We now give the final solutions for the two questions of Problem 2.

Let us begin with the first question. It is straightforward how one can use the computed data structures to identify, given a word w of length n, the triples (i, j, k) such that the factor $w[i..j]$ is in $\{t, f(t)\}^k$ for some t. Indeed, we return the solution-set comprising all the triples (i, j, k) for which the answer to $rep(i, j, k)$ is yes. The time needed to do so is $\Theta(n^3)$ (without the preprocessing), as we go through all possible triples (i, j, k) and check whether $rep(i, j, k)$ returns yes or no. Furthermore, any algorithm solving this problem needs $\Omega(n^3)$ operations in the worst case. Take, for instance, the non-erasing uniform morphism f defined by $f(a) = aa$ and $w = a^n$. It follows that $w[i..j]$ is in $\{a, f(a)\}^k$, for all i and j with $\lfloor (j - i + 1)/2 \rfloor \leq k \leq j - i + 1$; hence, for these w and f we have $\Theta(n^3)$ triples (i, j, k) in the solution set of our problem.

For f a literal anti-/morphism, we propose a $\Theta(n^2 \lg n)$ algorithm solving the discussed problem. Using the Sieve of Eratosthenes, we compute in $\mathcal{O}(n \lg n)$ time the lists of divisors for all numbers ℓ with $1 \leq \ell \leq n$. Further, for each pair $(i, i + \ell - 1)$ with $\ell \geq 1$ and all $d \mid \ell$ we check whether $rep(i, i + \ell - 1, d)$ returns yes. If so, the triple $(i, i + \ell - 1, d)$ is one of those we were looking for. Clearly, the algorithm is correct. Its complexity is $\mathcal{O}(n \lg n) + \Theta(\sum_{1 \leq \ell \leq n} (n - \ell + 1) d(\ell))$. Following Lemma 2, the overall complexity of this algorithm is $\Theta(n^2 \lg n)$. Moreover, any algorithm solving this problem does $\Omega(n^2 \lg n)$ operations in the worst case: for $w = a^n$ and the anti-/morphism $f(a) = a$, a correct algorithm returns exactly $\sum_{1 \leq \ell \leq n} (n - \ell + 1) d(\ell) \in \Theta(n^2 \lg n)$ triples. This proves our claim.

In the case of the second question of our problem, we proceed as follows. Recall that, in this case, we are given both a word w and a number k. To identify the pairs (i, j) such that the factor $w[i..j]$ is in $\{t, f(t)\}^k$ for some t we just have to go through all the possible values for i and j and check the answer of the query $rep(i, j, k)$. Clearly, this takes $\Theta(n^2)$ time. The preprocessing, in which the data structures needed to answer rep queries are built, takes in the more efficient case of non-erasing morphisms $\mathcal{O}(n^2)$ time, as well; in the general case, the preprocessing takes $\mathcal{O}(n^2k)$ time, and this is more than the time needed to actually answer all the queries. This improves in a more general framework the results reported in [2], where the same problem was solved in time $\mathcal{O}(n^2 \lg n)$. Finally, note that the bounds obtained for non-erasing morphisms are tight, since all the factors of length $k\ell$ of $w = a^n$ are equal to $(a^\ell)^k$, thus being solutions to our problem, no matter what anti-/morphism f is used. Hence, the number of elements in the solution-set of question (2) of Problem 2 for w is in $\Theta(n^2)$.

§Summary. Before concluding this section, recall that the key idea in our approach is to solve both parts of Problem 2 using rep queries. In order to assert the efficiency of this method note that, once data structures allowing us to answer such queries are constructed, our algorithms solve the two parts of Problem 2 efficiently. In particular, no other algorithm solving any of the two questions of Problem 2 can run faster than ours (excluding the preprocessing part), in the worst case. Hence, in general, a faster preprocessing part yields a faster complete solution for the problem. However, in the case of non-erasing and, respectively, literal anti-/morphisms (which includes the biologically motivated case of involutions) the preprocessing is as time-consuming as the part where we use the data structures we previously constructed to actually solve the questions of the problem. Thus, the time bounds obtained in these cases are tight.

▶ **Theorem 11.** *Let $f : V^* \to V^*$ be an anti-/morphism and $w \in V^*$ a given word, $|w| = n$.*
(1) One can identify in time $\mathcal{O}(n^{3.5})$ the triples (i, j, k) with $w[i..j] \in \{t, f(t)\}^k$, for a proper factor t of $w[i..j]$.
(2) One can identify in time $\mathcal{O}(n^2k)$ the pairs (i, j) such that $w[i..j] \in \{t, f(t)\}^k$ for a proper factor t of $w[i..j]$, when k is also given as input.
For a non-erasing f we solve (1) in $\Theta(n^3)$ time and (2) in $\Theta(n^2)$ time. For a literal f we solve (1) in $\Theta(n^2 \lg n)$ time and (2) in $\Theta(n^2)$ time.

References

1 T. M. Apostol. *Introduction to analytic number theory.* Springer, 1976.

2 E. Chiniforooshan, L. Kari, and Z. Xu. Pseudopower avoidance. *Fundam. Informat.*, 114(1):55–72, 2012.

3 E. Czeizler, L. Kari, and S. Seki. On a special class of primitive words. *Theoret. Comput. Sci.*, 411:617–630, 2010.

4 N. J. Fine and H. S. Wilf. Uniqueness theorem for periodic functions. *Proc. of the American Mathemat. Soc.*, 16:109–114, 1965.

5 D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology.* Cambridge University Press, New York, NY, USA, 1997.

6 J. Kärkkäinen, P. Sanders, and S. Burkhardt. Linear work suffix array construction. *J. ACM*, 53:918–936, 2006.

7 M. Lothaire. *Combinatorics on Words.* Cambridge University Press, 1997.

8 F. Manea, R. Mercas, and D. Nowotka. Fine and Wilf's theorem and pseudo-repetitions. In *MFCS*, volume 7464 of *LNCS*, pages 668–680. Springer, 2012.

Algorithms for Designing Pop-Up Cards

Zachary Abel*[1], Erik D. Demaine†‡[2], Martin L. Demaine[2], Sarah Eisenstat‡[2], Anna Lubiw§[3], André Schulz¶[4], Diane L. Souvaine‖[5], Giovanni Viglietta[6], and Andrew Winslow‖[5]

1 **MIT Department of Mathematics, 77 Massachusetts Ave., Cambridge, MA 02139, USA,** `zabel@math.mit.edu`
2 **MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA,** `{edemaine,mdemaine,seisenst}@mit.edu`
3 **David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada,** `alubiw@uwaterloo.ca`
4 **Institut für Mathematsche Logik und Grundlagenforschung, Universität Münster,** `andre.schulz@uni-muenster.de`
5 **Department of Computer Science, Tufts University, Medford, MA 02155, USA,** `{dls,awinslow}@cs.tufts.edu`
6 **School of Computer Science, Carleton University, Ottawa ON, Canada** `viglietta@gmail.com`

Abstract

We prove that every simple polygon can be made as a (2D) pop-up card/book that opens to any desired angle between 0 and 360°. More precisely, given a simple polygon attached to the two walls of the open pop-up, our polynomial-time algorithm subdivides the polygon into a single-degree-of-freedom linkage structure, such that closing the pop-up flattens the linkage without collision. This result solves an open problem of Hara and Sugihara from 2009. We also show how to obtain a more efficient construction for the special case of orthogonal polygons, and how to make 3D orthogonal polyhedra, from pop-ups that open to 90°, 180°, 270°, or 360°.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases geometric folding, linkages, universality

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.269

1 Introduction

Pop-up books have been entertaining children with their playful mechanics since their mass production in the 1970s. But the history of pop-ups is much older [27], and they were originally used for scientific and historical illustrations. The earliest known example of a "movable book" is Matthew Paris's *Chronica Majora* (c. 1250), which uses turnable disks (volvelle) to represent a calendar and uses flaps to illustrate maps. A more recent scientific example is George Spratt's *Obstetric Tables* (1850), which uses flaps to illustrate procedures for delivering babies. Dean & Sons' *Little Red Riding Hood* (1850) is the first known movable book where a flat page rises into a 3D scene, though here it was actuated by pulling a string.

* Supported in part by an NSF Graduate Research Fellowship.
† Supported in part by NSF ODISSEI grant EFRI-1240383 and Expedition grant CCF-1138967.
‡ Supported in part by NSF grant CCF-1161626 and DARPA/AFOSR grant FA9550-12-1-0423.
§ Supported by the Natural Sciences and Engineering Research Council of Canada.
¶ Supported in part by the German Research Foundation (DFG) under grant SCHU 2458/1-1.
‖ Supported in part by National Science Foundation grants CCF-0830734 and CBET-0941538.

The first known examples of *self-erecting* pop-ups, where the rise into 3D is actuated by opening the page, are a card promoting the Trinity Buildings in New York City (c. 1908), and S. Louis Girand's *Bookano Book* (c. 1930s). Modern pop-ups have taken these principles to new heights, often employing linkage-like mechanisms to form elaborate 3D shapes and motions; some good guides for designing pop-ups are [1, 3, 5, 20]. In recent years, pop-up books have risen to an art form with such art books as Bataille's *ABC3D* [2], Carter's series of dot/spot books [4], and Pelhem's poetic pop-up book [26]. One striking form of pop-ups is *origamic architecture*, which form buildings and other geometric structures, and are usually made by cutting a single sheet of card stock. A few examples of origamic architecture books are [7, 8, 32]; see [11] for a thorough bibliography.

Our results. This paper investigates the computational geometry of pop-ups, in particular, algorithmic design of pop-ups. We achieve three main results:

1. Any 2D n-gon (extruded orthogonally into 3D) can be popped up by opening a book to a specified angle θ with $0 < \theta \leq 360°$, using a construction of complexity $O(n^2)$.
2. Any orthogonal n-gon (extruded orthogonally into 3D) can be popped up by opening a book to a specified angle $\theta \in \{90°, 180°, 270°, 360°\}$, using a construction of complexity $\Theta(n)$.
3. Any orthogonal polyhedron can be popped up by opening a book to a specified orthogonal angle $\theta \in \{90°, 180°, 270°, 360°\}$, using a construction of complexity $O(n^3)$.

All of our constructions use rigid flat polygonal pieces to form single-degree-of-freedom linkage structures, which uniquely and deterministically unfold from the flat state to the open state, while avoiding collision.

Related work. Our results solve an open problem of Hara and Sugihara [14], who gave an algorithmic construction for arbitrary polygons, but with no guarantees of collision avoidance (and indeed the construction sometimes requires collisions). In another result in computational geometry, Uehara and Teramoto [31] proved that pop-ups with creases that can fold both mountain and valley are NP-hard to open or close.

In computer graphics, Mitani et al. [23, 24] showed how to automatically design pop-ups within a common class of 90° origamic architecture, in which the surface is monotone (hit only once) in the view direction. This work led to Tama Software's Pop-Up Card Designer [30]. Li et al. [22] developed a software system for converting a given 3D model into one that fits within this class. Several other systems enable designing and simulating pop-ups by composing standard pop-up gadgets, including Glassner's [12, 13], Popup Workshop [16, 15], Okamura and Igarashi's [25], and Iizuka et al.'s [19].

Geometric pop-ups have also been studied for specific examples of polyhedra. The first such example is a rhombic dodecahedron of the second type [10]. Other examples include the dodecahedron [29] and other Platonic solids [17, 6, 21]. These types of pop-ups are typically not attached to pages of a book, however.

Applications. Pop-ups have potential practical applications as well. Nano and micro fabrication technology are well-established for patterning 2D sheets, but remain in their infancy for 3D surfaces. Pop-ups offer a way to transform patterned 2D sheets into 3D surfaces. This idea was recently explored in the context of MEMS [18], where Hui et al. manufactured a 1.8mm-tall 3D model of the UC Berkeley Campanile clock tower using pop-ups.

2 Models of Pop-Ups

Our basic model is of a book with planar front and back covers which, when opened to a desired angle θ, pops up a 3D paper construction made from pieces of stiff paper that are folded and glued to each other and to the covers. (We will not deal with the more restrictive model of origamic architecture where one piece of paper is cut and folded but not glued.)

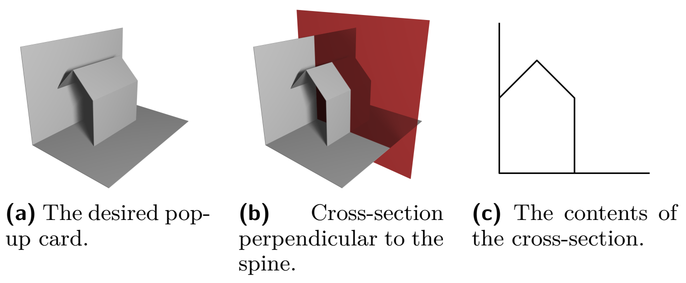

(a) The desired pop-up card. **(b)** Cross-section perpendicular to the spine. **(c)** The contents of the cross-section.

Figure 1 Three views of a desired 3D structure, before the creases and extra paper have been added to make it pop up.

Given a desired 3D structure, we aim to design a book that pops up the structure by adding creases and extra pieces of paper. Adding creases may be necessary to let the structure fold up when the book is closed. Adding extra paper may be necessary to make the structure pop up into the correct shape when the book is opened.

Until Section 5, we consider a restricted version of the problem that arises when all fold lines and all gluing lines are parallel to the spine, as in Figure 1. In this case, a cross-section of the 3D structure in a plane perpendicular to the spine yields a 2D pop-up: the pop-up structure forms a planar linkage composed of rigid *bars* (line segments) connected at joints. A *joint* is a point where bars intersect, usually at an endpoint of at least one of the bars. We distinguish three kinds of joints:

Common joints: Two or more bars are linked at one of their endpoints.

Flaps: A bar contains a joint in its interior, where an endpoint of another bar is linked. The location of the joint at the interior of the first bar is fixed.

Sliceforms: A joint (called a sliceform) can be formed by the intersection X of two bars. The intersection point X cannot shift along the bars, but the two bars can change their angle at X (scissors-like). Notice that we do not consider the two edges crossing if they are linked by a sliceform.

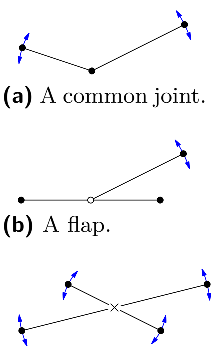

(a) A common joint. **(b)** A flap. **(c)** A sliceform.

Figure 2 The three types of joints used in this paper.

To distinguish the different joints in figures, we use a dot (●) for common joints and endpoints of edges, an empty circle (○) for flaps, and a cross (×) for sliceforms.

The common joint is sufficient to simulate the other joint types. A flap can be simulated by forming a zero-area triangle among the three collinear points. A sliceform can be simulated by common joints and flaps as illustrated in Figure 3.

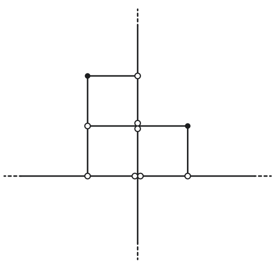


Figure 3 Simulating sliceforms.

In the 2D case, we want to construct a linkage L with one degree of freedom that unfolds to the desired polygon P. During the folding motion we require that no bars cross, and that the order of the bars emanating from a joint is preserved. Let the vertices of P be $v_1, v_2, \ldots, v_n$ labelled in counter-clockwise order. The edge incident to v_i and v_{i+1} is named e_i, and the edge between v_n and v_1 is named e_n. We assume that P is contained in one of the two wedges bounded by the rays $\overrightarrow{v_1v_2}$ and $\overrightarrow{v_1v_n}$. The angle of the wedge containing P is called the *opening angle*, and the union of the rays $\overrightarrow{v_1v_2}$ and $\overrightarrow{v_1v_n}$ is called the *cover*. We require L to have the following properties:

1. In one configuration of L, the boundary of L coincides with P. We call this the *open configuration*. The linkage L contains the edges e_1 and e_n of P as bars. If a joint of L coincides with a vertex v_i in the open configuration, we name it p_i.
2. In one configuration of L that can be reached from the open configuration, all edges are collinear and p_1 is an endpoint of the union of the edges of L. This configuration is called the *closed configuration*.
3. There is a unique motion that transforms the open configuration into the closed configuration. During this motion, L is contained inside the wedge defined by the cover and the opening angle decreases continuously. We refer to this motion as the *closing motion*. Every configuration of L obtained during the closing motion is called an *intermediate configuration*. The open configuration might have several joints that are opened 180°. In order to specify the folding uniquely, we prescribe for every such ambiguity the way the vertex moves during the folding motion. Collinear points in the open configuration appear naturally in pop-up structures. In the real world the folding motion at these points is prescribed by the creasing of the paper.

The *combinatorial complexity* of a 2D pop-up is equal to the number of joints in the pop-up.

3 Orthogonal Polygon Pop-Ups

In this section, we assume the polygon P is orthogonal, i.e., every edge of P is either horizontal or vertical. Under this assumption, we show how to construct a pop-up linkage L for the polygon P with combinatorial complexity linear in n. The techniques we use in this section are based on a particular type of motion:

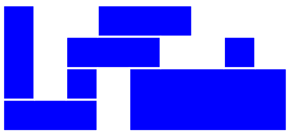
▶ Definition 1. A *shear* is a motion of a linkage that leaves parallel edges parallel.

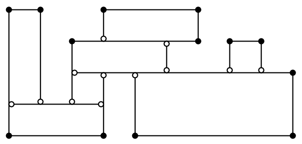
In Section 3.1, we explain how to construct pop-ups for polygons with opening angle 90°, also known as 90° *pop-ups*. In Section 3.2, we extend this result to larger opening angles.

3.1 90° Pop-Ups

To construct 90° pop-ups, we use a process called *h-superimposing*. As a first step we split P into *stripes* such that (i) each stripe is an axis-aligned rectangle, (ii) the left and right boundary edges of a stripe are a part of the boundary of P, and (iii) the union of any two stripes is not a rectangle. We obtain such a decomposition by extending all horizontal edges of P horizontally as long as they lie in P. See Figure 4a for an illustration. Two stripes are *adjacent* if they (partially) share an edge.

Let L_1 be the linkage obtained by extending all horizontal edges as long as they lie within P. The newly introduced degree-3 vertices become flaps. An example of this is depicted in Figure 4b. This intermediate linkage may have more than one degree of freedom: any pair of adjacent stripes that do not share a vertical bar can shear independently. To handle this, note that for any pair of adjacent stripes, there must be at least one vertical line passing through the (strict) interior of both stripes. We call this a *bracing line* for the stripe pair. The subset of the line contained in the stripe pair is called a *bracing segment*. For each pair of adjacent stripes that do not share a vertical bar, we add a bracing segment to the linkage, creating a sliceform joint where the segment intersects with the boundary between


(a) The stripes induced by extending all edges horizontally.


(b) The intermediate linkage, with too many degrees of freedom.

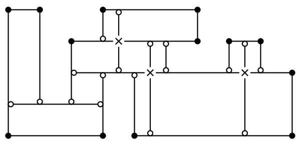

(c) The final linkage, with bracing segments to enforce a single shear motion.

Figure 4 The result of h-superimposing an orthogonal polygon.

the stripes. See Figure 4c for an example. Let L_2 be the linkage resulting from the addition of the bracing segments.

▶ Theorem 1. The linkage L_2 obtained by h-superimposing defines a pop-up fold for the orthogonal polygon P with 90° opening angle. The motion of L_2 is a shear. The combinatorial complexity of L_2 is $O(n)$.

All omitted proofs may be found in the full version of this paper.

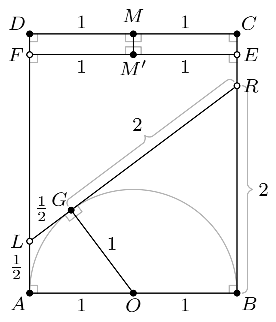
3.2 180°, 270°, and 360° Pop-Ups

This section is devoted to constructing pop-up folds with larger opening angles. We reduce this problem to the 90° pop-up scenario by introducing a linkage (called a *reflector gadget*) that allows us to reflect a shear. The open configuration of the gadget is constructed as shown in Figure 5a. Figure 5b depicts an intermediate configuration.

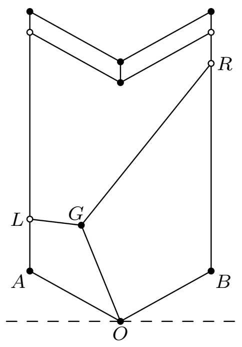
▶ Lemma 1. The reflector gadget has one degree of freedom. Its closing motion has the following properties:

1. the vertical line segments in the open configuration remain vertical during the induced motion,
2. the boundary of the gadget is symmetric with respect to a line of reflection running through $\overline{OM}$, and
3. the linkage folds to a line without introducing any crossings in an intermediate configuration.

We use the properties of the reflector to combine two 90° pop-ups to create a pop-up with larger opening angle. We discuss 180° pop-ups first. In this case both cover edges lie on a line through p_1. To guide our construction we add a bisector s of the cover edges that runs through p_1. Furthermore, we add two lines parallel to s such that the induced stripe contains s and does not contain any point of P except those lying on s. This stripe is called $\mathcal{S}$. The edges that "appear" when intersecting P with the boundary of $\mathcal{S}$ are added to the linkage L. We "fill" each rectangle obtained by intersecting P with $\mathcal{S}$ with a reflector gadget. The components of $P \setminus \mathcal{S}$ are turned into a linkage by h-superimposing as discussed in Section 3.1, so that every component of $P \setminus \mathcal{S}$ supports a shearing motion. The shearing motions are linked by the reflector gadgets, so the combined linkage L has one degree of freedom. By the properties of the reflector, the left and right side of



(a) The open configuration.



(b) An intermediate configuration.

Figure 5 The reflector gadget that helps to "reflect" two shearing motions.

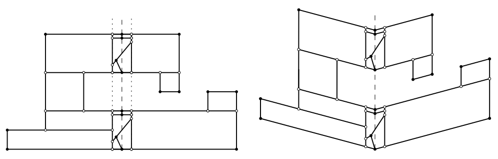

(a) (b)

Figure 6 A 180° pop-up fold constructed with the help of reflector gadgets. (a) The open configuration. (b) An intermediate configuration.

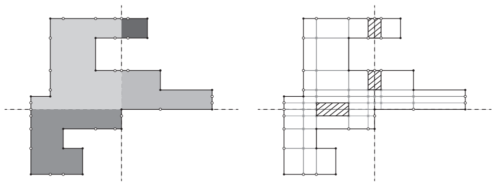

(a) (b)

Figure 7 (a) A polygon with opening angle 270°. The induced connected components are drawn with different shades of grey. (b) The pop-up linkage. The reflector gadgets have to be inserted at the crossed regions.

s perform a shear and both parts of P stay on their own side, relative to s. Hence it is impossible for L to self-intersect. Notice that we can always make the stripe $\mathcal{S}$ thin enough that the rectangles of $P \cap \mathcal{S}$ are not "too wide" for the reflector gadgets. See Figure 6 for an example. We conclude with:

▶ Theorem 2. The method described above constructs a pop-up fold for the polygon P with opening angle 180°. The combinatorial complexity of the linkage is $O(n)$.

In order to realize 270° and 360° folds we extend the 180° construction as follows. We split P into pieces by cutting it along the horizontal and vertical lines through p_1. We then turn each connected component of the split polygon into a 90° linkage, by adding bars and joints as discussed in Theorem 1. Then each piece of the polygon will be constrained to move in a shear motion, but different pieces will not necessarily move together. To synchronize the pieces, we use reflector gadgets to connect them. To generate the space for the gadgets, we add bars that sandwich the horizontal and vertical lines through p_1, thereby creating vertical and horizontal strips in which the reflector gadgets can be placed. Because no gadgets lie inside the intersection of the vertical and horizontal strip, no two reflector gadgets interfere. Figure 7 shows an example of an 270° fold. We conclude with the following theorem:

▶ Theorem 3. The method described above constructs a pop-up fold for the polygon P with opening angle 270° or 360°. The combinatorial complexity of the linkage is $O(n)$.

4 General Polygon Pop-Ups

In this section we provide a different method for constructing pop-ups of polygons. This method works for all simple P (not necessarily orthogonal), but has a higher asymptotic complexity of $O(n^2)$. Before giving the construction, we provide a key geometric lemma about the non-crossing of nested "V-fold" linkages.

4.1 Nested V-folds

We define an *outward V-fold* as the single-degree-of-freedom linkage formed by a (weakly) convex quadrilateral $ABCD$ with $AB + BC = AD + DC$. (This was called a V-fold in [14].) Such a linkage folds flat as the opening angle $\angle BAD$ decreases to zero. If, in the open configuration, the angle at C is 180° and the angle at A is less than 180° (i.e. the quadrilateral is a nontrivial triangle with C on side BD), we call this linkage a *flat outward V-fold*. Similarly, the linkage formed by a (weakly) non-convex quadrilateral $ABCD$ with

$AB - BC = AD - DC$ has one degree of freedom and folds flat without overlap, and is called an *inward V-fold.* If the angle at C is 180° and the angle at A is less than 180° it is a *flat inward V-fold.*

▶ Theorem 4. (a) Let $ABCD$ and $AB'C'D'$ be flat outward V-folds on the same rays with $\triangle BAD \subset \triangle B'AD'$, where we may have $B = B'$ or $D = D'$. Then these linkages do not cross during the closing motions. In fact, they do not touch at all, except at the closing configuration and possibly at the endpoints $B = B'$ or $D = D'$ if either equality holds.

(b) The same statement holds with "outward" replaced by "inward".

4.2 The General Pop-Up Construction: The Method

We may now describe the construction for pop-ups of general polygons. As in Section 2, we wish to construct a one-degree-of-freedom linkage L contained in simple polygon $P = v_1v_2 \cdots v_n$, where P is contained in the wedge formed by rays v_1v_2 and v_1v_n. We sometimes refer to the crease point v_1 as O. The opening angle θ of the original configuration, namely the angle of polygon P at vertex O, may take any value $0 < \theta \leq 360°$.

First we discuss the general strategy and provide a linkage L_1 that has a pop-up motion for polygon P but has more than one degree of freedom. Later we brace the linkage to remove the excess flexibility.

We first subdivide the wedge around O containing P by rays starting at O, where there is one such ray through each vertex of P and additional rays are inserted so that consecutive rays form acute angles. Suppose $r_1, \ldots, r_t$ are these rays in order around $O = v_1$, starting at $r_1 = \overrightarrow{Ov_2}$ and ending at $r_t = \overrightarrow{Ov_n}$. The region of the plane between rays r_i and r_{i+1} is the *ith wedge*, W_i. We subdivide polygon P by these rays: any positive length segment of a ray r_i contained in P or its boundary is inserted as a single bar in linkage L_1 and is called a *wall segment.* Notice that edges of P may be wall segments. Also, by slight abuse of terminology, a positive length subsegment of a wall segment is also called a wall segment. Any isolated points on $r_i \cap P$ are necessarily vertices of P and are called *wall points.*

The rays r_i subdivide P into a number of triangles and quadrilaterals, called *cells.* Each cell has two wall portions on consecutive rays: at least one of these is a wall segment, and the other may be a wall segment or point. A cell that has two wall segments is called an *internal cell*, and those with a wall point are *ear cells.* Two cells are *adjacent* if they share a wall segment. By adding at most one new ray for each ear cell, (and renumbering the rays as necessary), we may assume that each ear cell is adjacent to a unique interior cell.

The rays r_i also subdivide the boundary of P into segments. On each such segment AB that is not a wall segment (which implies A and B are on consecutive rays), insert a joint C at the point that would make $OACB$ an outward V-fold at O, i.e., C is the unique point on AB with $OA + AC = OB + BC$. This linkage L_1 serves our first stated purpose:

▶ Lemma 2. The linkage L_1, constructed from P by adding wall segments and extra boundary vertices as described here, can be continuously folded flat without overlap.

Proof. Let ϕ_i be the angle of wedge W_i, i.e., the angle between rays r_i and r_{i+1} at O. Consider any continuous rotation of rays $r_1, \ldots, r_t$ around O such that all angles $\phi_1, \ldots, \phi_{t-1}$ decrease monotonically to 0. Let each wall portion on ray r_i rotate around O to stay on ray r_i, and for each boundary portion ACB of P within wedge W_i, let ACB fold outward as would the outward V-fold $OACB$. Then path ACB remains inside wedge W_i throughout the motion, and therefore does not interact with portions of P in different wedges. Furthermore, by Theorem 4, the various boundary portions in wedge W_i do not touch each other throughout the motion. It follows that this is indeed a continuous planar motion of L_1. ◀

The rest of the construction shows how to add additional support to L_1 to turn it into a one-degree-of-freedom linkage whose motion has the form described in the proof of Lemma 2. We cut down the freedoms of L_1 in several steps, given in the next three subsections.

4.3 Constraining Wall Segments to Rotations

For two segments PQ and RS whose lines intersect at a point O, consider the *rotation gadget* as illustrated in Figure 8. (When we apply this below, PQ and RS will be wall segments, and O will be the crease point.) This linkage is specified as follows: $AB \parallel DE$ are any two segments not sharing an endpoint with PQ or RS with AB closer to O than DE; C is chosen on AB so that $OA + AC = OB + BC$, and the 180° angle at C is declared to fold outward, with F on DE chosen similarly; G and H are chosen so that both $DACG$ and $CBEH$ are parallelograms.

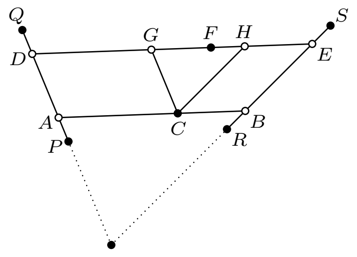


Figure 8 Rotation gadget.

▶ Lemma 3. The linkage illustrated in Figure 8 has one degree of freedom. If PQ and point O are held fixed in the plane, then in the unique motion, segment RS rotates rigidly around point O from its starting position to a closed configuration where PQ and RS are collinear.

▶ Lemma 4. Let L_2 be the linkage derived from L_1 as follows: for every internal cell, attach a rotator gadget inside the cell connecting (internal subintervals of) the wall segments. Then the motions of L_2 correspond exactly to those motions of L_1 where wall segments only rotate around O, and planar motions of L_1 extend (uniquely) to planar motions of L_2.

4.4 Synchronizing Wall Segments

We next show how to synchronize the wall segments to ensure that all wall segments originally on ray r_i remain on a single ray through O throughout any continuous motion. Let $\phi_1, \ldots, \phi_{t-1}$ be the initial angles of the wedges $W_1, \ldots, W_{t-1}$. For an internal cell $ABCD$ with $AB \subset r_i$ and $CD \subset r_{i+1}$, we know that any motion of L_2 rotates AB and CD around O, and we define the *angle* of the cell at any time as the angle between rays OAB and OCD.

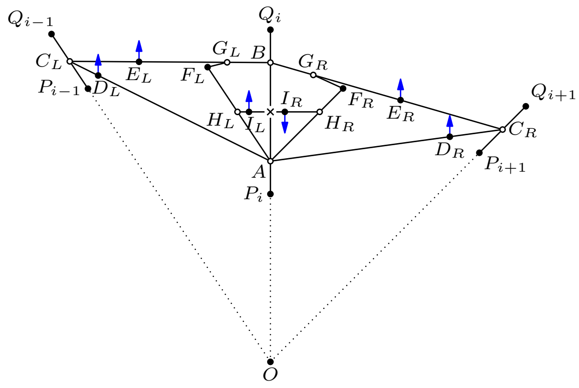


Figure 9 Synchronizing gadget.

▶ Definition 2. For each $1 \leq i \leq t-2$, construct a linkage M_i with two adjacent flat V-folds as follows. Points A, D, B, E, C are collinear, and connected in order (with B a flap on bar DE), and point O connects to A, B, and C. Angle OBA is 90°, $\angle AOB = \phi_i$, and $\angle BOC = \phi_{i+1}$. Finally, if i is even then $OADB$ is an *outward* flat V-fold and $OBEC$ is an *inward* flat V-fold, and if i is odd then $OADB$ is chosen outward and $OBEC$ is inward.

▶ Lemma 5. The linkage M_i defined as above has a single degree of freedom and folds from the initial configuration to a flat one without overlap. Furthermore, there is a continuous, strictly increasing, and invertible function $m_i : [0, \phi_i] \to [0, \phi_{i+1}]$ such that $m_i(\angle AOB) = \angle BOC$ during this motion.

Inductively define $\Phi_1(s) = s$ and $\Phi_i(s) = m_{i-1}(\Phi_{i-1}(s))$; these will control the rates at which internal cells' angles change. Specifically, fix an internal cell $X_1Y_1Y_2X_2$ with two wall segments X_1Y_1 and X_2Y_2 such that $X_1Y_1 \subset r_1$ and $X_2Y_2 \subset r_2$ initially. (We may have $X_1 = X_2 = O$.) Let s be a variable representing the angle of cell $X_1Y_1Y_2X_2$ during any motion. We will brace L_2 to a new linkage so that every internal cell initially in W_i will have angle $\Phi_i(s)$ during the motion.

To do this, we make the following additions to L_2 to form a new linkage L_3: For every pair of adjacent internal cells with wall segments $P_{i-1}Q_{i-1} \subset r_{i-1}$, $P_iQ_i \subset r_i$, and $P_{i+1}Q_{i+1} \subset r_{i+1}$ (note that P_iQ_i need not be a maximal wall segment for either cell), attach a *synchronizing gadget* as shown in Figure 9. The full version of this paper provides a more detailed description of this process.

▶ Lemma 6. Define L_3 as the linkage constructed from L_2 by inserting a synchronizing gadget between every pair of adjacent internal cells as described above. Then the continuous motions of L_3 correspond to those motions of L_2 such that the angle of any internal cell originally in wedge W_i is now $\Phi_i(s)$, where s represents the (changing) angle of cell $X_1Y_1Y_2X_2$. Furthermore, planar motions of L_2 induce planar motions of L_3.

4.5 Constraining Ear Cells

The configurations of all internal cells in L_3 are determined by $s = \angle Y_1OY_2$. The only unwanted degrees of freedom of L_3 must therefore come from the ear cells, which have not yet been modified. In this section we constrain these to produce the final linkage L.

Consider an ear cell with wall segment $P_iQ_i \in r_i$ and wall point $V_{i+1} \in r_{i+1}$, say. This is adjacent to a unique interior cell, with wall segment $P_{i-1}Q_{i-1}$ along r_{i-1}. To constrain ear cell $P_iQ_iP_{i+1}$, we simply add *two* synchronization gadgets centered on P_iQ_i that both connect to $V_{i+1} \in r_{i+1}$ and some point $V_{i-1} \in P_{i-1}Q_{i-1}$. Adding these synchronization gadgets for each ear cell produces the final linkage L:

▶ Theorem 5. The linkage L obtained from L_3 by adding two synchronization gadgets to each ear cell is a pop-up for the polygon P. Its boundary is connected and forms the polygon P in its opened configuration, and there are $O(n^2)$ total bars in the linkage.

5 Orthogonal Polyhedra Pop-Ups

In this section, we apply some of the techniques of 2D pop-up folds to the design of 3D pop-up structures that take the shape of orthogonal polyhedra. We first show how to construct pop-ups with an opening angle of 90°, then extend the construction to larger opening angles.

5.1 3D Pop-Up Model

In the 3D case, we model a pop-up using a model similar to *rigid origami*. A structure in rigid origami is composed of infinitely thin rigid sheets of paper, each in the shape of a simple polygon, connected using hinged joints. If two or more sheets are joined at a hinge and one is held fixed, then the only possible motion for the other sheet(s) is rotation around the hinge. A fold or a crease in a pop-up is equivalent to a hinge connecting two sheets. A flap in a pop-up corresponds to attaching the edge of one sheet to the center of another.

Let P be a simple polyhedron with n vertices $v_1, \ldots, v_n$. We select one edge e in P to be the *spine* of the pop-up. Let f_1 and f_2 be the faces adjacent to e. The *opening angle* of the pop-up is the measure of the dihedral angle between f_1 and f_2. The *cover* of the pop-up consists of the union of two halfplanes. The first halfplane in the cover is the half of the supporting plane of f_1 that contains f_1 and has the extension of e as its boundary. The half of the cover containing f_2 is defined similarly.

A rigid-origami structure L is a *3D pop-up* for P if it has an open configuration, a closed configuration, and a unique folding motion from open to closed, all defined analogously to the configurations of a 2D pop-up. The *combinatorial complexity* of the 3D pop-up L is equal to the number of hinges.

Note that unlike in the 2D case, it is not sufficient to add more paper and more creases. By the Bellows Theorem [28, 9], if we treat a polyhedron as a linkage where each face is rigid and faces must rotate around edges, then all motions preserve the volume of the polyhedron. Hence, we cannot fold the polyhedron flat unless we cut the boundary of the polyhedron.

5.2 Scaffold Pop-Ups

Suppose we have a simple orthogonal polyhedron P with an opening angle of 90°. Without loss of generality, we may assume that e lies along the z-axis, and that f_1 lies in the positive x section of the xz plane. Suppose further that f_2 lies in the positive y section of the yz plane. Let $x_1, \ldots, x_n$ be the sorted x-coordinates of all vertices in P. Similarly, let $y_1, \ldots, y_n$ be the sorted y-coordinates and let $z_1, \ldots, z_n$ be the sorted z-coordinates. Then *grid cell* (i, j, k) is the rectangular box $[x_i, x_{i+1}] \times [y_j, y_{j+1}] \times [z_k, z_{k+1}]$. By construction, the polyhedron P is the union of a face-connected subset R of grid cells. The *scaffold* of P is the union of all faces f of cells in R such that f is parallel to the spine.

The *grid slice* G_k consists of the union of all grid cells (i, j, k), not necessarily contained in P. Let the *slice scaffold* S_k be the intersection of the scaffold with G_k. The slice scaffold contains no faces perpendicular to the z-axis, and every cross section perpendicular to the z-axis is the same. Hence, the problem of constructing a pop-up for S_k is purely 2D.

To construct a pop-up for S_k with the correct shear motion, we must somehow combine faces of S_k into larger rigid sheets. If an edge borders exactly three faces, then the two coplanar faces can be fused into a rigid sheet, with the third face added as a flap. Suppose instead that we have an edge with x and y coordinates (x_i, y_j) bordering exactly four faces. If $(i+j)$ is even, then we rigidify the pair of faces perpendicular to the x-axis; otherwise, we rigidify the pair of faces perpendicular to the y-axis. This construction means that the four sheets adjacent to a given grid cell are arrayed in a pinwheel pattern. This ensures that the shear motion of one cell must be the same as the shear of all adjacent cells.

Suppose that we use this construction to make a pop-up-like structure for each slice, which we will call a *pinwheel slice.* Place all pinwheel slices side-by-side so that the initial position takes the shape of the scaffold. Call the result of this process the *sliced pinwheel scaffold.* Unfortunately, the sliced pinwheel scaffold has too many degrees of freedom: each slice scaffold is disconnected from its neighbors, and even within a single slice the scaffold may be disconnected.

Given any pair $r_1, r_2 \in R$ of adjacent cells in adjacent slices, we wish to cause any motions of the sheets around r_1 to affect the sheets around r_2. For each such pair r_1, r_2, we fuse each of the four sheets that surround r_1 in the initial configuration with the corresponding coplanar sheet around r_2, to create four larger rigid sheets in the initial opening configuration. Call the result of this fusing the *pinwheel scaffold* of P.

▶ Lemma 7. The pinwheel scaffold of a polyhedron P is a pop-up for the scaffold of P. The pinwheel scaffold has complexity $O(n^3)$.

The pinwheel scaffold has a number of faces parallel to the spine. All such faces are contained within P when the scaffolding is open, and all faces on the boundary of P that are parallel to the spine also exist in the scaffolding (although they may be subdivided). The only missing pieces are the faces of P that are perpendicular to the spine.

5.3 Additional Faces

To add those pieces to the pinwheel scaffold, we first subdivide the faces using our rectilinear grid so that the sheets we wish to add to the pinwheel scaffold are faces of the grid cells. We must attach each such sheet to the sheets in the scaffold that form the adjacent grid cell.

There are four potential hinges that we could use to attach the new face to the scaffold. The hinges we choose to use are the hinge parallel to the x-axis with the smallest y-coordinate, and the hinge parallel to the y-axis with the smallest x-coordinate. By construction, the angle between these two hinges will grow smaller as the pinwheel scaffold folds. Therefore, if we attach the new face using these hinges, it is sufficient to add a crease to the new sheet emanating from the intersection of the two hinges at a 45° angle. For consistency, we make each crease constructed in this fashion fold in the positive z-direction. We call the resulting rigid origami structure the *draped scaffold*.

▶ Theorem 6. The draped scaffold of P is a pop-up for P with complexity $O(n^3)$.

The draped scaffold may be used to construct 90° pop-ups in 3D. By combining this structure with a reflector gadget as in Section 3.2, we can extend our construction to handle larger multiples of 90°. See the full version for details.

6 Conclusion and Open Problems

In this paper, we demonstrate techniques for designing 2D pop-ups for general polygons, and 3D pop-ups for orthogonal polyhedra. The most obvious open question is whether there is a way to construct 3D pop-ups for general polyhedra. Another question to consider is which 2D or 3D shapes are constructible using a single sheet of material with no gluing, as in most origamic architecture.

References

1 Carol Barton. *The Pocket Paper Engineer: How to Make Pop-Ups Step-by-Step.* Popular Kinetics Press, Glen Echo, Maryland, 2005–2008. Two volumes.
2 Marion Bataille. *ABC3D.* Roaring Brook Press, 2008.
3 Duncan Birmingham. *Pop-Up Design and Paper Mechanics: How to Make Folding Paper Sculpture.* Guild of Master Craftsman Publications, 2010.
4 David A. Carter. *One Red Dot: A Pop-up Book for Children of All Ages.* Little Simon, 2005. Other books include *Blue 2* (2006) and *600 Black Spots* (2007).
5 David A. Carter and James Diaz. *The Elements of Pop-Up.* Little Simon, New York, 1999.
6 David Cassell. Pop-up polyhedra. *Mathematics in School*, 17(1):24–27, January 1988.
7 Masahiro Chatani. *Key to Origamic Architecture.* Shokokusha, 1985.
8 Masahiro Chatani. *Pattern Sheets of Origamic Architecture.* Books Nippan, 1986. Two volumes.
9 R. Connelly, I. Sabitov, and A. Walz. The bellows conjecture. *Beiträge Algebra Geom*, 38(1):1–10, 1997.
10 John Lodge Cowley. *Solid Geometry.* London, 1752.
11 Evermore Origamic Architecture. Pop-up card books. `http://www.evermore.com/oa/books.php3`, 2011.

12 Andrew Glassner. Interactive pop-up card design, part 1. *IEEE Computer Graphics and Applications*, 22(1):79–86, 2002.

13 Andrew Glassner. Interactive pop-up card design, part 2. *IEEE Computer Graphics and Applications*, 22(2):74–85, 2002.

14 Takuya Hara and Kokichi Sugihara. Computer-aided design of pop-up books with two-dimensional v-fold structures. In *Abstracts from the 7th Japan Conference on Computational Geometry and Graphs*, Kanazawa, Japan, November 2009.

15 Susan Hendrix. Popup workshop. `http://l3d.cs.colorado.edu/~ctg/projects/popups/`, 2007.

16 Susan L. Hendrix and Michael A. Eisenberg. Computer-assisted pop-up design for children: computationally enriched paper engineering. *Advanced Technology for Learning*, 3(2), April 2006.

17 Peter Hilton and Jean Pedersen. Constructing pop-up polyhedra. In *Build Your Own Polyhedra*, chapter 7, pages 101–105. Addison-Wesley, 1994. Based on an article "Pop-up Polyhedra" by Jean Pedersen, *California Mathematics*, April 1983, pages 37–41.

18 Elliot E. Hui, Roger T. Howe, and M. Steven Rodgers. Single-step assembly of complex 3-D microstructures. In *Proceedings of the 13th Annual International Conference on Micro Electro Mechanical Systems*, pages 602–607, January 2000.

19 Satoshi Iizuka, Yuki Endo, Jun Mitani, Yoshihiro Kanamori, and Yukio Fukui. An interactive design system for pop-up cards with a physical simulation. *The Visual Computer*, 27(6):605–612, 2011. Proceedings of Computer Graphics International 2011.

20 Paul Jackson. *The Pop-Up Book: Step-by-Step Instructions for Creating Over 100 Original Paper Projects.* Holt Paperbacks, 1993.

21 Scott Johnson and Hans Walser. Pop-up polyhedra. *The Mathematical Gazette*, 81(492):364–380, 1997.

22 Xian-Ying Li, Chao-Hui Shen, Shi-Sheng Huang, Tao Ju, and Shi-Min Hu. Popup: Automatic paper architectures from 3D models. *ACM Transactions on Graphics*, 29(4):Article 111, 2010. Proceedings of SIGGRAPH 2010.

23 Jun Mitani and Hiromasa Suzuki. Computer aided design for origamic architecture models with polygonal representation. In *Proceedings of Computer Graphics International*, pages 93–99, 2004.

24 Jun Mitani, Hiromasa Suzuki, and Hiroshi Uno. Computer aided design for origamic architecture models with voxel data structure. *Transactions of Information Processing Society of Japan*, 44(5):1372–1379, 2003.

25 Sosuke Okamura and Takeo Igarashi. An assistant interface to design and produce a pop-up card. *International Journal of Creative Interfaces and Computer Graphics*, 1(2):40–50, 2010.

26 David Pelham. *Trail: Paper Poetry Pop-Up.* Little Simon, 2007.

27 Ellen G. K. Rubin. A history of pop-up and movable books: 700 years of paper engineering. Public lecture, November 10 2010. `http://www.youtube.com/watch?v=iDJJOaZ1myM`.

28 I. Kh. Sabitov. On the problem of the invariance of the volume of a deformable polyhedron. *Uspekhi Mat. Nauk*, 50(2(302)):223–224, 1995.

29 Hugo Steinhaus. *Mathematical Snapshots*, pages 196–198. Oxford University Press, 1950. Republished by Dover Publications, 1999.

30 Tama Software. Pop-up card designer. `http://www.tamasoft.co.jp/craft/popupcard_en/`, 2007. Pro version, `http://www.tamasoft.co.jp/craft/popupcard-pro_en/`, 2008.

31 Ryuhei Uehara and Sachio Teramoto. The complexity of a pop-up book. In *Proceedings of the 18th Annual Canadian Conference on Computational Geometry*, Ontario, Canada, August 2006.

32 Diego Uribe. *Fractal Cuts.* Tarquin, 1994.

Space-Time Trade-offs for Stack-Based Algorithms

Luis Barba[1], Matias Korman*[2], Stefan Langerman[1], Rodrigo I. Silveira†[2], and Kunihiko Sadakane[3]

1 **Université Libre de Bruxelles (ULB), Brussels, Belgium.**
`{lbarbafl,stefan.langerman}@ulb.ac.be`

2 **Universitat Politècnica de Catalunya (UPC), Barcelona, Spain.**
`{matias.korman,rodrigo.silveira}@upc.edu`

3 **National Institute of Informatics (NII), Tokyo, Japan.**
`sada@nii.ac.jp`

Abstract

In memory-constrained algorithms we have read-only access to the input, and the number of additional variables is limited. In this paper we introduce the compressed stack technique, a method that allows to transform algorithms whose space bottleneck is a stack into memory-constrained algorithms. Given an algorithm $\mathcal{A}$ that runs in $O(n)$ time using a stack of length $\Theta(n)$, we can modify it so that it runs in $O(n^2/2^s)$ time using a workspace of $O(s)$ variables (for any $s \in o(\log n)$) or $O(n \log n/ \log p)$ time using $O(p \log n/ \log p)$ variables (for any $2 \leq p \leq n$). We also show how the technique can be applied to solve various geometric problems, namely computing the convex hull of a simple polygon, a triangulation of a monotone polygon, the shortest path between two points inside a monotone polygon, 1-dimensional pyramid approximation of a 1-dimensional vector, and the visibility profile of a point inside a simple polygon. Our approach exceeds or matches the best-known results for these problems in constant-workspace models (when they exist), and gives a trade-off between the size of the workspace and running time. To the best of our knowledge, this is the first general framework for obtaining memory-constrained algorithms.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases space-time trade-off, constant workspace, stack algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.281

1 Introduction

The amount of resources available to computers is continuing to grow exponentially year after year. Many algorithms are nowadays developed with little or no regard to the amount of memory used. However, with the appearance of specialized devices, there has been a renewed interest in algorithms that use as little memory as possible.

Moreover, even if we can afford large amounts of memory, it might be preferable to limit the number of writing operations. For instance, writing into flash memory is a relatively

* M.K. was supported by the Secretary for Universities and Research of the Ministry of Economy and Knowledge of the Government of Catalonia and the European Union and was partially supported by the ESF EUROCORES programme EuroGIGA - ComPoSe IP04 - MICINN Project EUI-EURC-2011-4306.

† R.I.S. was supported by the FP7 Marie Curie Actions Individual Fellowship PIEF-GA-2009-251235 and was partially supported by the ESF EUROCORES programme EuroGIGA - ComPoSe IP04 - MICINN Project EUI-EURC-2011-4306.

slow and costly operation, which also reduces the lifetime of the memory. Write-access to removable memory devices might also be limited for technical or security reasons. Whenever several concurrent algorithms are working on the same data, write operations also become problematic due to concurrency problems. A possible way to deal with these situations is considering algorithms that do not modify the input, and use as few variables as possible.

Several different memory-constrained models exist in the literature. In most of them the input is considered to be in some kind of read-only data structure. In addition to the input, the algorithm is allowed to use a small amount of variables to solve the problem. In this paper, we look for space-time trade-off algorithms; that is, we devise algorithms that are allowed to use up to s additional variables (for any parameter $s \leq n$). Naturally, our aim is that the running time of the algorithm decreases as s grows.

Many problems have been considered under this framework. In virtually all of the results, either an unconstrained algorithm is transformed to memory-constrained environments, or a new algorithm is created. Regardless of the type, the algorithm is usually an ad-hoc method tailored for the particular problem. In this paper, we take a different approach: we present a simple yet general approach to construct memory-constrained algorithms. Specifically, we present a method that transforms a class of algorithms whose space bottleneck is a stack into memory-constrained algorithms. In addition to being simple, our approach has the advantage of being able to work in a *black-box* fashion: provided that some simple requirements are met, our technique can be applied to any stack-based algorithm without knowing specifics details of their inner workings.

Stack Algorithms. One of the main algorithmic techniques in computational geometry is the incremental approach. At each step, a new element of the input is considered and some internal structure is updated in order to maintain a partial solution to the problem, which in the end will result in the final output. We here focus on *stack algorithms*, that is, incremental algorithms where the internal structure is a stack (and possibly $O(1)$ extra variables). A more precise definition is given in Section 2.

We show how to transform any such algorithm into an algorithm that works in memory-constrained environments. The main idea behind our approach is to avoid storing the stack explicitly, reconstructing it whenever needed. The running time of our approach depends on the size of the workspace. Specifically, it runs in $O(n \log n / \log p)$ time and uses $O(p \log n / \log p)$ variables (for any $2 \leq p \leq n$). In particular, when $p = n^{\varepsilon}$ the technique gives a linear-time algorithm that uses only $O(n^{\varepsilon}/\varepsilon)$ variables (for any $\varepsilon > 0$).

If only $o(\log n)$ space is available, we must restrict the class of algorithms considered slightly. We say that a stack algorithm is *green*[1] if, without using the stack, it is possible to reconstruct its top stack element in linear time (this will be formalized in Section 4.2). We show how to transform any green stack algorithm into one that runs in $O(n^2/2^s)$ time using $O(s)$ variables for any $s \in o(\log n)$.

Our techniques are conceptually very simple, and can be used with any (green) stack algorithm in an essentially black-box fashion. We only need to replace the stack data structure with the compressed data structure explained in Section 4.1, and create one or two additional operations for reconstructing elements in the stack. To the best of our knowledge, this is the first general framework for obtaining memory-constrained algorithms.

Applications. The technique is applicable, among others, to the following well-known and fundamental geometric problems (illustrated in Fig. 1). More details about these problems are presented in Section 5.

[1] or environmentally friendly.

Convex hull of a simple polygon The convex hull problem has already been studied in memory-constrained environments. Brönnimann and Chan [8] modified the method of Lee [17] so as to obtain several linear-time algorithms using memory-reduced workspaces. However, their model of computation allows in-place rearranging (and sometimes modifying) the vertices of the input and therefore does not fit in the memory constrained model considered here. In our model, the well-known *gift wrapping* or *Jarvis march* algorithm [15], reports the convex hull of a set of points (or a simple polygon) in $O(n\bar{h})$ time using $O(1)$ variables, where $\bar{h}$ is the number of vertices on the convex hull. In the same model, Chan and Chen [9] showed how to compute the upper hull of a sorted set of n points in linear time using $O(n^{\varepsilon})$ extra variables for any fixed $\varepsilon > 0$.

Triangulation of a monotone polygon The memory-constrained version of this problem was studied by Asano *et al.* [3]. In that paper, the authors give an algorithm that triangulates *mountains* (a subclass of monotone polygons in which one of the chains is a segment). Combining this result with a trapezoidal decomposition, they give a method to triangulate a planar straight-line graph. Both operations run in quadratic-time in an $O(1)$-workspace.

Shortest path computation Without memory restrictions, the shortest path between two points in a simple polygon can be computed in $O(n)$ time [14]. Asano *et al.* [4] gave an $O(n^2)$ algorithm for solving this problem with $O(1)$-workspace, which later was extended to $O(s)$-workspaces [3]. Their algorithm starts with a (possibly quadratic) preprocessing phase that consists in repeatedly triangulating $\mathcal{P}$, and storing $O(s)$ edges that partition $\mathcal{P}$ into $O(s)$ subpieces of size $O(n/s)$ each. Once the polygon is triangulated, they compute the geodesic between the two points in $O(n^2/s)$ time by navigating through the sub-polygons. Our triangulation algorithm removes the preprocessing overhead of Asano *et al.* when restricted to monotone polygons.

Optimal 1-dimensional pyramid approximation Given an n-dimensional vector $f = (x_1, \ldots, x_n)$, find a unimodal vector $\phi = (y_1, \ldots, y_n)$ that minimizes the squared L_2-distance $||f - \phi||^2 = \sum_{i=1}^{n}(x_i - y_i)^2$. Linear-time algorithms for the problem exist [10], but up to now it had not been studied for memory-constrained settings.

Visibility polygon (or profile) of a point in a simple polygon Asano *et al.* [4] asked for a sub-quadratic algorithm for this problem in $O(1)$-workspaces. Barba *et al.* [6] provided a space-time trade-off algorithm that runs in $O(\frac{nr}{2^s} + n\log^2 r)$ time (or $O(\frac{nr}{2^s} + n\log r)$ randomized expected time) using $O(s)$ variables (where $s \in O(\log r)$, and r is the number of reflex vertices of $\mathcal{P}$). Parallel to this research, De *et al.* [11] proposed an $O(n)$ time algorithm using $O(\sqrt{n})$ variables.

We show in Section 5 that there exist green algorithms for all of the above applications (except for the shortest path computation), hence our technique results in new algorithms that run in $O(n^2/s)$ time for an $O(s)$-workspace (for $s \in o(\log n)$) or $O(n \log n/\log p)$ time using $O(p \log n/\log p)$ variables (for any $2 \leq p \leq n$). In particular, when $p = n^{1/\varepsilon}$, they run in linear-time using $O(n^{\varepsilon})$ variables (for any constant $\varepsilon > 0$). The running time of the trade-off matches or exceeds the best known algorithms throughout its space range. Due to lack of space, many proofs have been deferred to the full version

2 Preliminaries

Given its importance, a significant amount of research has focused on memory-constrained algorithms, some of them dating back to the 1980s [20]. In this paper, we use a generalization of the constant-workspace model, introduced by Asano *et al.* [4, 5]. In this model, the input of the problem is in a read-only data structure. In addition to the input, an algorithm

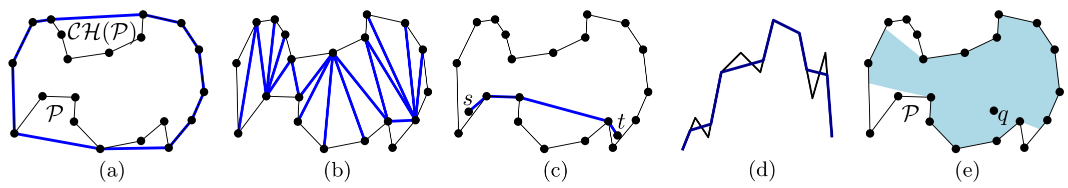


Figure 1 Applications of the compressed stack, from left to right: convex hull of a simple polygon, triangulation of a monotone polygon, shortest path computation between two points inside a monotone polygon, optimal 1-d pyramid approximation, and visibility polygon of a point $q \in \mathcal{P}$.

can only use a constant number of additional variables to compute the solution. Implicit storage consumption required by recursive calls is also considered part of the workspace. In complexity theory, the constant-work space model has been studied under the name of *log space* algorithms [2]. In this paper, we are interested in allowing more than a constant number of workspace variables. Therefore, we say that an algorithm is an s-workspace algorithm if it uses a total of $O(s)$ variables during its execution. We aim for an algorithm whose running time decreases as s grows, effectively obtaining a space-time trade-off. Since the size of the output can be larger than our allowed space s, the solution is not stored but reported in a write-only memory.

In the usual constant-workspace model, one is allowed to perform random access to any of the values of the input in constant time. The technique presented in this paper does not make use of such random access. Thus, unless the algorithm being adapted specifically needs it, our technique works in a more constrained model in which, given a pointer to a specific input value, we can either access it, or move the pointer to the previous or next input value. This is the case in which, for example, the input values are given in a doubly-linked list. We follow this model of computation and allow scanning the input as many times as necessary. Our model is particularly interesting when the input data cannot be modified, write operations are much more expensive than read operation, or whenever several programs need to access the same data concurrently.

Stack Algorithms

Let $\mathcal{A}$ be a deterministic algorithm that uses a stack, and possibly other data structures $\mathcal{DS}$ of total size $O(1)$. We assume that $\mathcal{A}$ uses a generalized stack structure that can access the last k elements that have been pushed into the stack (for some constant k). That is, in addition to the standard PUSH and POP operations, we can execute TOP(i) to obtain the i-th topmost element (for well-definedness purposes, this operation will return $\emptyset$ if either the stack does not have i elements or $i > k$).

We say that an algorithm is a *stack* algorithm if it follows the scheme in Algorithm 1. Notice that the scheme focuses on how the stack is handled, thus other operations could be present in $\mathcal{A}$, provided that the treatment of the stack is unaltered. For simplicity of exposition, we assume that only values of the input are pushed (see line 7 of Algorithm 1). In the general case one could push a tuple whose identifier is a. We allow this fact provided that the tuple has size $O(1)$. Essentially, our technique consists in replacing the $O(n)$-space stack of $\mathcal{A}$ by a *compressed stack* which uses less space. As we will see in Section 4.1, most of the time of our compressed stack structure is spent on computing the top element of the stack after a pop has been executed. For the case in which only $o(\log n)$ space is available, we must add one requirement to the algorithm. We require the existence of a GetTop operation that, given an input value $a \in \mathcal{I}$ and a consecutive interval $\mathcal{I}' \subseteq \mathcal{I}$ of the input,

Algorithm 1 Basic scheme of a stack algorithm

```
1: Initialize stack and auxiliary data structure $\mathcal{DS}$ with $O(1)$ elements from $\mathcal{I}$
2: for all subsequent input $a \in \mathcal{I}$ do
3:    while some-condition($a$,$\mathcal{DS}$,STACK.TOP(1),..., STACK.TOP(k)) do
4:       STACK.POP()
5:    end while
6:    if another-condition($a$,$\mathcal{DS}$,STACK.TOP(1),..., STACK.TOP(k)) then
7:       STACK.PUSH($a$)
8:    end if
9: end for
10: Report(STACK)
```

computes the $(k+1)$-th topmost element of the stack, provided that it belongs to $\mathcal{I}'$. This operation should run in $O(|\mathcal{I}'|)$ time using $O(s)$ variables. Whenever such operation exists, we say that $\mathcal{A}$ is *green*. Notice that this procedure need not be used by $\mathcal{A}$. In fact, in Section 5 we give a list of green algorithms, but none of them uses their corresponding GetTop operations. Full details on this operation are given in Section 4.2.

3 Compressed stack technique for $\Theta(\sqrt{n})$-workspaces

As a warm-up, we first show how to reduce the working space to $O(\sqrt{n})$ variables without increasing the asymptotic running time. Let $a_1, \ldots, a_n \in \mathcal{I}$ be the values of the input, in the order in which they are treated by $\mathcal{A}$. In order to avoid explicitly storing the stack, we virtually subdivide the values of $\mathcal{I}$ into blocks $B_1, \ldots, B_p$, such that each block B_i contains n/p consecutive values.[2] In this section we take $p = \sqrt{n}$. Then the size of each block will be $n/p = p = \sqrt{n}$. Note that, since we scan the values of $\mathcal{I}$ in order, we always know to which block the current value belongs to. Naturally, the stack can contain elements of different blocks. However, by the scheme of the algorithm, we know that all elements of one block will be consecutively pushed into the stack. We virtually group the elements in the stack according to the block that they belong to. At any point during the execution, we explicitly store the elements of the top two blocks in the stack. For the remaining blocks (if any), we store the first and last elements that were pushed into the stack. We say that these blocks are stored in *compressed* format.

For any input value a, we define the *context* of a as the content of the auxiliary data structure $\mathcal{DS}$ right after a has been treated. Note that the context occupies $O(1)$ space in total. For each block, regardless if we store it explicitly or in compressed format, we also store the context of the first element that was pushed into the stack.

It follows that for most blocks we only have the topmost and bottommost elements that we pushed into the stack (denoted a_t and a_b, respectively), but there could possibly be many more elements that we have not stored. For this reason, at some point during the execution of the algorithm we will need to reconstruct the missing elements in the compressed blocks of the stack. In order to do so we introduce a Reconstruct operation. Given a_t, a_b and the context of a_b, Reconstruct explicitly recreates all elements between a_b and a_t that existed in the stack right after a_t was processed.

[2] For simplicity of exposition, we assume that n is a power of p.

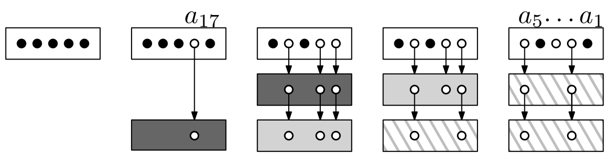


Figure 2 Push operation: the top row has the 25 input values partitioned into blocks of size 5 (white points indicate values that will be pushed during the execution of the algorithm; black points are those that will be discarded). The middle and bottom rows show the situation of the compressed stack before and after a_{17} has been pushed into the stack. Block $\mathcal{F}$ is depicted in dark gray, $\mathcal{S}$ in light gray, and the remaining compressed blocks with a diagonal stripe pattern.

▶ **Lemma 1.** Reconstruct *runs in $O(m)$ time and uses $O(m)$ variables, where m is the number of elements in the input between a_b and a_t.*

Each time we invoke procedure Reconstruct, we do so with the first and last elements that were pushed into the stack of the corresponding block. In particular, we have the context of a_b stored, hence we can correctly invoke the procedure. Also note that we have $m \leq n/p = p = \sqrt{n}$, hence this operation does not use any additional space. In order to obtain the desired running time, we must make sure that not too many unnecessary reconstructions are done. At any point of the execution, let $\mathcal{F}$ and $\mathcal{S}$ be the first and second topmost blocks in the stack, respectively. Recall that these are the only two blocks that are stored explicitly. Moreover, they are the latest blocks that we have visited and contained input values in the stack. There are two cases to consider whenever a value a is pushed: if a belongs to $\mathcal{F}$, it is added normally to the stack in constant time. Otherwise, we must create a new block containing only a. As a result, block $\mathcal{F}$ will become a new block only containing a, $\mathcal{S}$ will be the previous $\mathcal{F}$, and we must compress the former $\mathcal{S}$ (see Fig. 2). All these operations can be done in constant space by smartly reusing pointers. The pop operation is similar: as long as the current block $\mathcal{F}$ contains at least one element, the pop is executed as usual. If $\mathcal{F}$ is empty, we pop values from $\mathcal{S}$ instead. If after a pop operation the block $\mathcal{S}$ becomes empty, we pick the first compressed block from the stack (if any) and reconstruct it in full. Recall that we can do so in $O(\sqrt{n})$ time using $O(\sqrt{n})$ variables using Lemma 1. The reconstructed block becomes the new $\mathcal{S}$ (and $\mathcal{F}$ remains empty).

▶ **Theorem 2.** *The compressed stack technique can be used to transform $\mathcal{A}$ into an algorithm that runs in $O(n)$ time and uses $O(\sqrt{n})$ variables.*

Proof. The general workings of $\mathcal{A}$ remain unchanged, hence the difference in the running time (if any) will be due to push and pop operations. In most cases these operations only need a constant number of operations. The only situation in which an operation takes more than constant time is when pop is performed and block $\mathcal{S}$ becomes empty. In this situation, we must pay $O(\sqrt{n})$ time to reconstruct another block from the stack.

Recall that $\mathcal{A}$ scans the values of $\mathcal{I}$ in order: if at some point in the execution we push an element a belonging to a block B_i, we know that no element of block B_j (for some $j < i$) will afterwards be pushed into the stack. In particular, whenever the pop operation takes $O(\sqrt{n})$ time, we know that no element of the block associated to $\mathcal{S}$ is pushed (nor will be again be) into the stack. Thus, the cost of the reconstruction operation can be charged to that block. No block can be charged twice, hence at most $O(n/p)$ reconstructions are done. Since each reconstruction needs $O(p)$ time, the total time spent reconstructing blocks is bounded by $O(n)$. Regarding space use, at any point of the execution we keep at most two blocks in explicit form and the others compressed. The two top blocks need $O(p)$ space

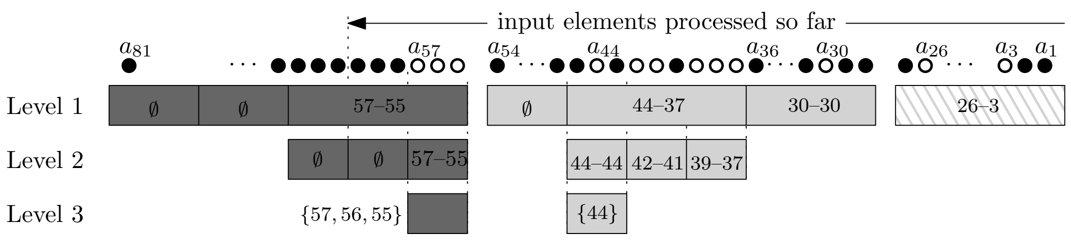


Figure 3 A compressed stack for $n = 81$, $p = 3$ (thus $h = 3$). The compression levels are depicted from top to bottom (for clarity, the blocks of the same level are not equally-sized). Color notation for points and blocks is as in Fig. 2. Compressed blocks contain the indices corresponding to the first and last element inside the block (or three pairs if the block is partially compressed); explicitly stored blocks contain a list of the pushed elements.

whereas the remaining at most $p - 2$ blocks need $O(1)$ space each. Hence the space needed is $2(n/p) + p \times O(1)$, which equals $O(\sqrt{n})$ if $p = \sqrt{n}$. ◀

4 Compressed stack technique

In this section we present our compressed stack technique in full generality. For simplicity of exposition we describe it for the case in which $\mathcal{A}$ only accesses the topmost element of the stack (that is, $k = 1$). In the full version we explain how to extend the algorithm for larger values of k. We first present the technique for $\Omega(\log n)$-workspaces. Afterwards, we discuss the modifications needed for it to work on $o(\log n)$-workspaces.

4.1 For $\Omega(\log n)$-workspaces

In this section we generalize the above approach to the general case in which we partition the input into p blocks for any parameter $2 \leq p \leq n$ (the exact value of p will be determined by the user). Similarly to the previous case, we virtually decompose the input into p blocks of size n/p each. Instead of explicitly storing the top two non-empty blocks, we further subdivide them into into p sub-blocks. This process is then repeated for $h := \log_p n - 1$ levels until the last level, where the blocks are explicitly stored.[3]

We consider three different levels of compression: a block is either stored (i) *explicitly* if it is stored in full, (ii) *compressed* if only the first and last elements of the block are stored, or (iii) *partially-compressed*, if the block is subdivided into p smaller sub-blocks, and only the first and last element of each sub-block are stored. Analogously to the previous section, for each block in either compressed or semi-compressed format we store the context right after the first value of that block was pushed into the stack.

During the execution of the algorithm, the first level of compression will contain p blocks, with the top two partially compressed and the rest compressed. The i-th level of compression (for $1 < i < h$) will consist of two blocks of size n/p^{i-1} that are partially compressed. Thus each block is further subdivided into p sub-blocks of size n/p^i each. The first two non-empty sub-blocks are given to a lower level. In the lower level, the process continues recursively by further dividing the given sub-blocks. This process repeats until the h-th level, in which

[3] For simplicity in the explanation, we assume that our workspace is large enough to store the whole recursion. We note that this approach can be adapted for the case in which we have a workspace of $c \log n$ variables (for any $c > 0$). Details will be given in the full version of the paper.

the block size is $n/p^h = n/p^{\log_p n-1} = p$, and is explicitly stored. Thus, in all but the lowest level, the top two blocks, denoted $\mathcal{F}_i$ and $\mathcal{S}_i$ for level i, are partially-compressed (whereas in the last level they are stored explicitly). See Fig. 3 for an illustration. Note that we allow blocks $\mathcal{F}_i$ to be empty, but blocks $\mathcal{S}_i$ can only be empty when the stack is empty.

▶ **Lemma 3.** *The compressed stack structure uses $O(p \log n/ \log p)$ space.*

Proof. At the first level of the stack we have p blocks. The first two are partially-compressed and need $O(p)$ space each, whereas the remaining blocks are compressed, and need $O(1)$ space each. Since the topmost level can have at most p blocks, the total amount of space needed at the first level is bounded by $O(p)$.

At other levels of compression, we only keep two partially-compressed blocks (or two explicitly stored blocks for the lowest level). Regardless of the level in which it belongs to, each block needs $O(p)$ space. Since the total number of levels is h, the algorithm will never use more than $O(ph)$ space to store the compressed stack. ◀

Push operation. A push can be treated in each level $i \leq h$ independently. First notice that by the way in which values of $\mathcal{I}$ are pushed, the new value a either belongs to $\mathcal{F}_i$ or it is the first pushed element of a new block. In the first case, we register the change to $\mathcal{F}_i$ directly by updating the top value of the appropriate sub-block of $\mathcal{F}_i$ (or adding it to the list of elements if $i = h$). If the value to push does not belong to $\mathcal{F}_i$ we must create a new block, which will contain only a (and, if $i = 1$, we must compress the old $\mathcal{S}_i$). Since we are creating a block, we also store the context of a in the block. As in Section 3, these operations can be done in constant time for a single level.

Pop operation. This operation starts at the bottommost level h, and it is then transmitted to levels above. Naturally, we must first remove the top element of $\mathcal{F}_h$ (unless $\mathcal{F}_h = \emptyset$ in which case we must pop from $\mathcal{S}_h$ instead). In the simplest case, the block from which we popped has at least one more element. If this holds, no block is destroyed and the structure of the stack will not be affected. Note that we know which element of the stack will become the new top (since we have it explicitly stored). Thus, we need only transmit the new top of the stack to levels above. In those levels, we need only update the top element of the corresponding sub-block. The more complex situation happens when the pop operation emptied either $\mathcal{F}_h$ or $\mathcal{S}_h$. In the former case, we transmit the information to a level above. In this level we mark the sub-block as empty and, if this results in an empty block, we again transmit the information to a level above, and so on. During this procedure several blocks $\mathcal{F}_i$ may become empty, but no block of type $\mathcal{S}_j$ will do so (since $\mathcal{F}_i$ is always included in $\mathcal{F}_{i-1}$). Note that this is no problem, since in general we allow blocks $\mathcal{F}_i$ to be empty.

Finally, it remains to consider the case in which the pop operation results in block $\mathcal{S}_h$ becoming empty. First, we transmit this information to level above, which might also result in an empty block, and so on. We stop at a level i in which block $\mathcal{S}_i$ is empty (in which either $i = 1$ or $\mathcal{S}_{i-1}$ is not empty). We now must invoke the reconstruction procedure to obtain blocks $\mathcal{S}_j$ for all $j \geq i$. To reconstruct $\mathcal{S}_i$ we obtain from one level higher $(i-1)$ the first and last elements that correspond to the next non-empty sub-block after $\mathcal{S}_h$. Recall that at level $i-1$ we keep two blocks (of size n/p^{i-1}) in partially-compressed format. Hence, the values we need will be explicitly stored (since, by definition of i, block $\mathcal{S}_{i-1}$ will not empty after the pop is executed). If $i = 1$ and we reached the highest level, we pick the first compressed block and reconstruct that one instead. In either case, the first and last elements of the block to reconstruct are always known. Once $\mathcal{S}_i$ is reconstructed, we can proceed to reconstruct $\mathcal{S}_{i+1}$, and so on until we reconstruct $\mathcal{S}_h$.

Block Reconstruction. This operation is invoked when a block in the i-th level of compression needs to be reconstructed. We are given the first and last elements of that block that were pushed into the stack, denoted a_b and a_t, respectively, as well as the context right after a_b was inserted. Our aim is to obtain all stack elements between a_b and a_t right after a_t was pushed into the stack. This information should be in either explicit format (if $i = h$) or in partially-compressed format (if $i < h$). To reconstruct the block we use our algorithm recursively. The base case (i.e., if $i = h$) is handled with Lemma 1. For larger blocks, we execute $\mathcal{A}$ with the compressed data structure for a smaller size input (from a_b to a_t).

▶ **Lemma 4.** Reconstruct *runs in* $O(m)$ *time and uses* $O(p \log m / \log p)$ *space, where* m *is the number of elements between* a_b *and* a_t*.*

▶ **Theorem 5.** *Any stack algorithm can be adapted so that, for any parameter* $2 \leq p \leq n$*, it solves the same problem in* $O(n \log n / \log p)$ *time using* $O(p \log n / \log p)$ *variables.*

4.2 For $o(\log n)$-workspaces

The previous technique can be used provided that the workspace is of size at least $\Omega(\log n)$. In the following we adapt it for smaller workspaces. From now on, we assume that $\mathcal{A}$ is green. The condition for an algorithm $\mathcal{A}$ to be green is to have a GetTop operation. The general idea of this operation is the following: imagine that $\mathcal{A}$ is treating value a and at some point in time it pops the top element of the stack (denoted by t). Instead of reconstructing, we will invoke procedure GetTop to find the new top element of the stack (denoted by ℓ). This operation must scan $\mathcal{I}$ until ℓ is found. Although we do not know exactly where ℓ lies, we will use the information in our compressed stack to guide this operation. Hence, for efficiency reasons, we restrict the procedure to look within a given interval. That is, procedure GetTop receives three parameters: (i) the input value a that is generating the pop, (ii) two input values $t, b \in \mathcal{I}$, such that t is the current element at the top of the stack (i.e., the one that needs to be popped), and $b \neq t$ is another element that is in the stack, and (iii) the context information right before b was pushed into the stack. GetTop must report the pair $(\ell, \text{Context}(\ell))$ in $O(m)$ time and $O(s)$ variables, where m is the number of input values between b and t in $\mathcal{I}$. Since b is in the stack and $b \neq t$, the value ℓ must always exist.

We apply the block partition strategy of the previous section, with $p = 2$ for s levels (recall that s is our allowed workspace). The only difference in the data structure occurs at the lowermost level, where each block has size $n/2^s$. Although we would like to store the blocks of the lowest level explicitly, the size of a single block is too large to fit into memory (if $s \in o(\log n)$, we have $n/2^s \in \omega(s)$). Instead, the blocks of the bottommost level are stored in compressed format. Recall that we store the context of the first element that is pushed into any block. Additionally, we store the context of stack.top(1) (i.e. the top of the stack).

Push operations are handled exactly as in Section 4.1 (taking into account that the last level is now in compressed format): at each level it suffices to update the topmost element of $\mathcal{F}_i$, or create a new block containing the new input value (if it belongs to a new block).

Pop operations are also handled in a similar fashion as in Section 4.1. In most cases, we must remove one element from $\mathcal{F}_s$, unless the block is empty. In that case, we pop from $\mathcal{S}_s$. If both are empty, we pop from $\mathcal{F}_{s-1}$, and so on. This process ends when either $\mathcal{F}_1 \cup \mathcal{S}_1 = \emptyset$ (so the stack becomes empty after the pop) or we reach a block B_i of level i that does not become empty after the pop. As in Section 4.1 all blocks $\mathcal{S}_i$ of level $i < s$ that become empty must be reconstructed. This is done recursively using Reconstruct. The only difference is at the bottommost level, where instead of reconstructing we use GetTop with the top and bottom element of the block to obtain the new top element of the stack.

▶ **Lemma 6.** *The space used by a pop operation in $o(\log n)$-workspaces is $O(s)$. Moreover, the total time spent in all pop operations is $O(n^2/2^s)$.*

As in $\Omega(\log n)$-workspaces, the time bottleneck of the algorithm is given by the Pop operation, hence we obtain the following bounds.

▶ **Theorem 7.** *Any green stack algorithm can be adapted so that it solves the same problem in $O(n^2/2^s)$ time in an $O(s)$-workspace (for any $s \in o(\log n)$).*

5 Applications

In this section we show how our technique can be applied to several well-known geometric problems. For each problem we present an existing algorithm that is a (green) stack algorithm, where our technique can be applied to produce a space-time trade-off.

5.1 Convex hull of a simple polygon

Computing convex hulls is a fundamental problem in computational geometry, used as an intermediate step to solve many other geometric problems. For the particular case of a simple polygon $\mathcal{P}$ (Fig. 1(a)), there exist several algorithms in the literature that compute the convex hull of $\mathcal{P}$ in linear time (see the survey by Aloupis [1]). Among these, we highlight the one of Lee [17] that is green.

▶ **Lemma 8.** *Lee's algorithm for computing the convex hull of a simple polygon [17] is green.*

▶ **Theorem 9.** *The convex hull of a simple polygon can be reported in $O(n \log n/\log p)$ time using $O(p \log n/\log p)$ additional variables (for any parameter $2 \le p \le n$), or $O(n^2/2^s)$ time using $O(s)$ additional variables (for any parameter $s \in o(\log n)$).*

5.2 Triangulation of a monotone polygon

A simple polygon is called *monotone* with respect to a line ℓ if for any line ℓ' perpendicular to ℓ, the intersection of ℓ' and the polygon is connected. In our context, the goal is to report the diagonal edges of a triangulation of the given monotone polygon (see Fig. 1 (b)). Monotone polygons are a well-studied class of polygons because they are easier to handle than general polygons, can be used to model (polygonal) function graphs, and often can be used as stepping stones to solve problems on simple polygons (after subdividing them into monotone pieces). It is well-known that a monotone polygon can be triangulated in linear time using linear space [13].

▶ **Lemma 10.** *Garey* et al.*'s algorithm for triangulating a monotone polygon [13] is green.*

▶ **Theorem 11.** *A triangulation of a monotone polygon of n vertices can be reported in $O(n \log n/\log p)$ time using $O(p \log n/\log p)$ additional variables (for any parameter $2 \le p \le n$), or $O(n^2/2^s)$ time using $O(s)$ additional variables (for any parameter $s \in o(\log n)$).*

The only previous work on polygon triangulation in memory-constrained environment is due to Asano *et al.* [3]. In that paper, the authors give an algorithm that triangulates *mountains* (a subclass of monotone polygons in which one of the chains is a segment). Combining that result with a trapezoidal decomposition, they give a method to triangulate a planar straight-line graph. Both operations run in quadratic time in an $O(1)$-workspace. Our method speeds up the first half of their algorithm, hence if one were to obtain a time-space trade-off for computing the trapezoidal decomposition, we would instantly obtain a similar result for triangulating any polygon.

5.3 The shortest path between two points in a monotone polygon

Shortest path computation is another fundamental problem in computational geometry with many variations, especially queries restricted within a bounded region (see [18] for a survey). Given a polygon $\mathcal{P}$, and two points $p, q \in \mathcal{P}$, their *geodesic* is defined as the shortest path that connects p and q among all the paths that stay within $\mathcal{P}$ (Fig. 1(c)). It is easy to verify that, whenever $\mathcal{P}$ is a simple polygon, the geodesic always exists and is unique. The length of that path is called the *geodesic distance*.

Asano *et al.* [3,4] gave an $O(n^2/s)$ algorithm for solving this problem in $O(s)$-workspaces, provided that we allow an $O(n^2)$-time preprocessing. This preprocessing phase essentially consists in repeatedly triangulating $\mathcal{P}$, and storing $O(s)$ edges that partition $\mathcal{P}$ into $O(s)$ subpieces of size $O(n/s)$ each. Theorem 11 allows us to remove the preprocessing overhead of Asano *et al.* when $\mathcal{P}$ is a monotone polygon.

▶ **Theorem 12.** *Given a monotone polygon $\mathcal{P}$ of size n and points $p, q \in \mathcal{P}$, we can compute the geodesic that connects them in $O(n^2/s)$-time in an $O(s)$-workspace (for any $s \leq n$).*

5.4 Optimal 1-dimensional pyramid

A vector $\phi = (y_1, \dots, y_n)$ is called *unimodal* if $y_1 \leq y_2 \leq \cdots y_k$ and $y_k \geq y_{k+1} \geq \cdots y_n$ for some $1 \leq k \leq n$. The 1-D optimal pyramid problem [10] is defined as follows. Given an n-dimensional vector $f = (x_1, \dots, x_n)$, find a unimodal vector $\phi = (y_1, \dots, y_n)$ that minimizes the squared L_2-distance $||f - \phi||^2 = \sum_{i=1}^{n}(x_i - y_i)^2$ (Fig. 1(d)). This problem has several applications in the fields of computer vision [7] and data mining [12, 19]. Although the linear-time algorithm of Chun *et al.* [10] does not exactly fit into our scheme, it can be modified so that our approach can be used as well.

▶ **Theorem 13.** *The 1-D optimal pyramid for an n-dimensional vector can be computed in $O(n \log n / \log p)$ time using $O(p \log n / \log p)$ additional variables (for any parameter $2 \leq p \leq n$), or $O(n^2/2^s)$ time using $O(s)$ additional variables (for any $s \in o(\log n)$).*

5.5 Visibility profile in a simple polygon

In the visibility profile (or polygon) problem we are given a simple polygon $\mathcal{P}$, and a point $q \in \mathcal{P}$ from where the visibility profile needs to be computed. A point $p \in \mathcal{P}$ is visible (with respect to q) if and only if $pq \subset \mathcal{P}$, where pq denotes the segment connecting points p and q. The set of points visible from q is denoted by $\mathrm{Vis}_{\mathcal{P}}(q)$ and is called the *visibility profile* (or polygon) of q (see Fig. 1(e)). Visibility computations arise naturally in many areas, such as computer graphics and geographic information systems, and have been widely studied in computational geometry. Among several linear-time algorithms, we are interested in the method of Joe and Simpson [16] since it can be easily shown that it is green.

▶ **Lemma 14.** *Joe and Simpson's algorithm for computing the visibility profile [16] is green.*

▶ **Theorem 15.** *The visibility profile of a point q with respect to $\mathcal{P}$ can be reported in $O(n \log n / \log p)$ time using $O(p \log n / \log p)$ additional variables (for any $2 \leq p \leq n$), or $O(n^2/2^s)$ time using $O(s)$ additional variables (for any $s \in o(\log n)$).*

6 Conclusions

In this paper we have shown how to transform any stack algorithm so as to work in memory-constrained models, and presented several concrete applications where it can be applied.

Moreover, for many applications the technique can be applied in a black box fashion without altering the specifics of the algorithm. In addition, since the technique is rather simple to implement, we believe it can be useful in practice. A natural open problem is extending this approach to other data structures (e.g. trees), which would allow many other useful algorithms to work in memory-constrained workspaces.

References

1 G. Aloupis. A history of linear-time convex hull algorithms for simple polygons. http://cgm.cs.mcgill.ca/~athens/cs601/.

2 S. Arora and B. Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 2009.

3 T. Asano, K. Buchin, M. Buchin, M. Korman, W. Mulzer, G. Rote, and A. Schulz. Memory-constrained algorithms for simple polygons. *CoRR*, abs/1112.5904, 2011.

4 T. Asano, W. Mulzer, G. Rote, and Y. Wang. Constant-work-space algorithms for geometric problems. *Journal of Computational Geometry*, 2(1):46–68, 2011.

5 T. Asano, W. Mulzer, and Y. Wang. Constant-work-space algorithms for shortest paths in trees and simple polygons. *J. Graph Algorithms Appl.*, 15(5):569–586, 2011.

6 L. Barba, M. Korman, S. Langerman, and R. I. Silveira. Computing the visibility polygon using few variables. In *ISAAC*, pages 70–79, 2011.

7 I. Bloch. Unifying quantitative, semi-quantitative and qualitative spatial relation knowledge representations using mathematical morphology. In *TFCV*, pages 153–164, 2003.

8 H. Brönnimann and T. M. Chan. Space-efficient algorithms for computing the convex hull of a simple polygonal line in linear time. *Computational Geometry: Theory and Applications*, 34(2):75–82, 2006.

9 T. M. Chan and E. Y. Chen. Multi-pass geometric algorithms. *Discrete & Computational Geometry*, 37(1):79–102, 2007.

10 J. Chun, K. Sadakane, and T. Tokuyama. Linear time algorithm for approximating a curve by a single-peaked curve. *Algorithmica*, 44(2):103–115, 2006.

11 M. De, A. Maheshwari, and S. C. Nandy. Space-efficient algorithms for visibility problems in simple polygon. *CoRR*, abs/1204.2634, 2012.

12 T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Data mining with optimized two-dimensional association rules. *ACM Transactions on Database Systems*, 26(2):179–213, June 2001.

13 M. R. Garey, David S. Johnson, Franco P. Preparata, and Robert Endre Tarjan. Triangulating a simple polygon. *Information Processing Letters*, 7(4):175–179, 1978.

14 L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences*, 39(2):126–152, October 1989.

15 R.A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2(1):18 – 21, 1973.

16 B. Joe and R. B. Simpson. Corrections to Lee's visibility polygon algorithm. *BIT Numerical Mathematics*, 27:458–473, 1987.

17 D. T. Lee. On finding the convex hull of a simple polygon. *International Journal of Parallel Programming*, 12(2):87–98, 1983.

18 J. S. B. Mitchell. Shortest paths and networks. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 607–642. 2nd edition, 2004.

19 Y. Morimoto, T. Fukuda, S. Morishita, and T. Tokuyama. Implementation and evaluation of decision trees with range and region splitting. *Constraints*, 2:401–427, 1997.

20 J. I. Munro and M. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12:315–323, 1980.

L_1 Shortest Path Queries among Polygonal Obstacles in the Plane

Danny Z. Chen*[1] and Haitao Wang†[2]

1 Department of Computer Science and Engineering
University of Notre Dame, Notre Dame, IN 46556, USA
dchen@cse.nd.edu

2 Department of Computer Science, Utah State University
Logan, UT 84322, USA
haitao.wang@usu.edu

Abstract

Given a point s and a set of h pairwise disjoint polygonal obstacles with a total of n vertices in the plane, after the free space is triangulated, we present an $O(n + h \log h)$ time and $O(n)$ space algorithm for building a data structure (called *shortest path map*) of size $O(n)$ such that for any query point t, the length of the L_1 shortest obstacle-avoiding path from s to t can be reported in $O(\log n)$ time and the actual path can be found in additional time proportional to the number of edges of the path. Previously, the best algorithm computes such a shortest path map in $O(n \log n)$ time and $O(n)$ space. In addition, our techniques also yield an improved algorithm for computing the L_1 geodesic Voronoi diagram of m point sites among the obstacles.

1998 ACM Subject Classification F.2 Analysis of algorithms and problem complexity

Keywords and phrases computational geometry, shortest path queries, shortest paths among obstacles, L_1/L_∞/rectilinear metric, shortest path maps, geodesic Voronoi diagrams

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.293

1 Introduction

Given a point s and a set of h pairwise disjoint polygonal obstacles, $\mathcal{P} = \{P_1, P_2, \ldots, P_h\}$, with a total of n vertices in the plane, where s is considered as a special point obstacle, the plane minus the interior of the obstacles is called the *free space* of $\mathcal{P}$. Two obstacles are pairwise *disjoint* if they do not intersect in their interior. The L_1 *shortest path query problem*, denoted by L_1-SPQ, is to compute a data structure (called *shortest path map* or SPM for short) with s as the *source point* such that for any query point t, an L_1 shortest obstacle-avoiding path from s to t can be obtained efficiently. Note that such a path can have any polygonal segments but the length of each segment of the path is measured by the L_1 metric. Unless otherwise stated, all SPMs mentioned in this paper have the following performances: for any query point t, the length of the L_1 shortest path from s to t can be reported in $O(\log n)$ time and the actual path can be found in additional time proportional to the number of edges of the path.

We also study the L_1 *geodesic Voronoi diagram problem*, denoted by L_1-GVD: Given an obstacle set $\mathcal{P}$ and a set of m point sites in the free space, compute the geodesic Voronoi diagram for the m point sites under the L_1 distance metric among the obstacles in $\mathcal{P}$.

* The research of Chen was supported in part by NSF under Grants CCF-0916606 and CCF-1217906.
† Corresponding author.

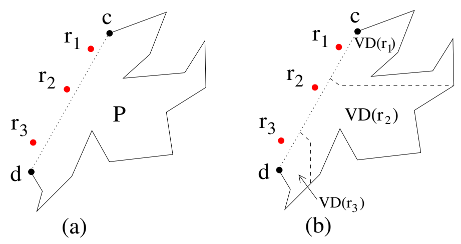


Figure 1 (a) Three weighted point sites (r_1, r_2, r_3) and a simple polygon P with an open edge $\overline{cd}$. The goal is to compute the L_1 geodesic Voronoi diagram in P for the three sites which influence P only through the edge $\overline{cd}$. (b) Illustrating a possible solution: P is partitioned into three Voronoi regions $VD(r_i)$ for each r_i, $1 \le i \le 3$.

Computing L_1 shortest paths has been studied extensively (e.g., see [5, 8, 15, 18, 19, 21]). Mitchell [18, 19] builds an SPM in $O(n \log n)$ time and $O(n)$ space, which is optimal when $h = \Theta(n)$ because finding an L_1 shortest path has a lower bound of $\Omega(n + h \log h)$ on the running time [9]. Throughout this paper, let T refer to the time for triangulating the free space of $\mathcal{P}$ and let $\epsilon > 0$ be any arbitrarily small constant. It is known that $T = O(n \log n)$ and $T = O(n + h \log^{1+\epsilon} h)$ [1]. Recently, Chen and Wang gave an algorithm that can find a single shortest path in $O(T + n + h \log h)$ time and $O(n)$ space [5]. However, the algorithm in [5] cannot construct an SPM, which is left as an open problem in [5]. For L_1-GVD, Mitchell's algorithm [18, 19] can be extended to solve it in $O((n + m) \log(n + m))$ time.

1.1 Our Results

In this paper, we answer the open problem in [5] by presenting an algorithm that builds an SPM of size $O(n)$ in $O(n + h \log h)$ time and $O(n)$ space after the free space is triangulated in $O(T)$ time. Hence, the running time of the overall algorithm is $O(T + n + h \log h)$. If the triangulation can be done optimally (i.e., $T = O(n + h \log h)$), then our algorithm matches the $\Omega(n + h \log h)$ time lower bound [9]. In addition, it is easy to see that given an SPM, we can add $h - 1$ segments in the free space to connect the obstacles in $\mathcal{P}$ together to obtain a single simple polygon and then triangulate the free space, in totally $O(n)$ time [1, 2]. This shows that the problem L_1-SPQ is solvable in $\Theta(T)$ time, i.e., building an SPM is equivalent to triangulating the free space of $\mathcal{P}$ in terms of the running time.

As Mitchell's algorithm [18, 19], our techniques can also be extended to solve the L_1-GVD problem in $O(T' + (m + h) \log(m + h)))$ time, where T' is the time for triangulating the free space of $\mathcal{P}$ with the m point sites and $T' = \min\{O((n + m) \log(n + m)), O(n + (m + h) \log^{1+\epsilon}(m + h))\}$ [1]. Our new algorithm is faster than Mitchell's $O((n + m) \log(n + m))$ time solution for sufficiently small m and h (e.g., when $m + h = O(n^{1-\epsilon})$).

It is well known that with solutions in L_1 version, the same problems in the rectilinear version and L_∞ version can be solved immediately [18, 19]. Hence, our results also hold for the rectilinear version and the L_∞ version of the problems.

A *challenging subproblem* we need to solve is a special case of the (additively) weighted L_1 geodesic Voronoi diagram problem on a simple polygon P: The weighted point sites all lie outside P and influence P through an (open) edge (e.g., see Fig. 1). Our main effort of this paper is to solve this problem in $O(n' + m')$ time, where n' is the number of vertices of P and m' is the number of sites. This problem is interesting in its own right.

This subproblem may not look "challenging" at all as it can be solved by many existing techniques, e.g., the continuous Dijkstra paradigm [18, 19], the sweeping algorithm [11], and divide-and-conquer [20]. However, all these methods would lead to an $O((n' + m') \log(n' + m'))$

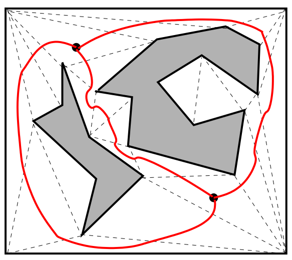


Figure 2 Illustrating a triangulation of the free space among two obstacles and the corridors (with red solid curves). There are two junction triangles indicated by the large dots inside them, connected by three solid (red) curves. Removing the two junction triangles results in three corridors.

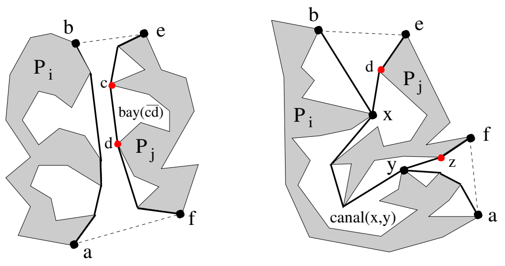


Figure 3 Illustrating an open hourglass (left) and a closed hourglass (right) with a corridor path connecting the apices x and y of the two funnels. The dashed segments are diagonals. The paths $\pi(a,b)$ and $\pi(e,f)$ are shown with thick solid curves. A bay $bay(\overline{cd})$ with gate $\overline{cd}$ (left) and a canal $canal(x,y)$ with gates $\overline{xd}$ and $\overline{yz}$ (right) are also indicated.

time solution, and consequently, the overall time for building an SPM would be $O(n \log n)$. Our linear time algorithm can be viewed as an incremental approach. Incremental approaches have been widely used in geometric algorithms, and normally they can result in good randomized algorithms. Incremental approaches have also been used for constructing Voronoi diagrams, which usually take quadratic time. Our result demonstrates that incremental approaches are able to yield optimal deterministic solutions for building Voronoi diagrams. The new techniques we provide here should be generalized to solving other related problems.

For simplicity of discussion, as in [18, 19], we assume no two obstacle vertices lie on the same horizontal or vertical line. Henceforth, unless otherwise stated, a shortest path refers to an L_1 shortest path and a length is in the L_1 metric.

In the following, in Section 2, we review some geometric structures. In Section 3, we outline our algorithm for computing an SPM. Particularly, our algorithm for solving the challenging subproblem is in Section 4. Due to the space limit, many details (including the algorithm for L_1-GVD) are omited and can be found in the full version of this paper [6].

2 Preliminaries

In this section, we review some geometric structures of $\mathcal{P}$, i.e., the corridors, ocean, bays, and canals, which have been used previously, e.g., [5, 7, 16].

For simplicity, we assume all obstacles are contained in a rectangle $\mathcal{R}$ (see Fig. 2). We also view $\mathcal{R}$ as an obstacle in $\mathcal{P}$. Let $\mathcal{F}$ be the free space inside $\mathcal{R}$. We compute an arbitrary triangulation of $\mathcal{F}$, denoted by $Tri(\mathcal{F})$, in $O(T)$ time.

Let $G(\mathcal{F})$ be the (planar) dual graph of $Tri(\mathcal{F})$. As shown in [16], based on $G(\mathcal{F})$, we compute a 3-regular graph, denoted by G^3 (the degree of every node in G^3 is three), possibly with loops and multi-edges, as follows. First, remove every degree-one node from $G(\mathcal{F})$ along with its incident edge; repeat this process until no degree-one node remains. Second, remove every degree-two node from $G(\mathcal{F})$ and replace its two incident edges by a single edge; repeat this process until no degree-two node remains. The resulting graph is G^3 (see Fig. 2), which has $O(h)$ faces, $O(h)$ nodes, and $O(h)$ edges [16]. Each node of G^3 corresponds to a triangle in $Tri(\mathcal{F})$, which is called a *junction triangle* (see Fig. 2). The removal of all junction triangles from $Tri(\mathcal{F})$ results in $O(h)$ *corridors*, each of which corresponds to one edge of G^3.

The boundary of a corridor C consists of four parts (see Fig. 3): (1) A boundary portion of an obstacle $P_i \in \mathcal{P}$, from a point a to a point b; (2) a diagonal of a junction triangle from

b to a boundary point e on an obstacle $P_j \in \mathcal{P}$ ($P_i = P_j$ is possible); (3) a boundary portion of the obstacle P_j from e to a point f; (4) a diagonal of a junction triangle from f to a. Let $\pi(a,b)$ (resp., $\pi(e,f)$) be the shortest path from a to b (resp., e to f) inside C. The region H_C bounded by $\pi(a,b), \pi(e,f)$, and the two diagonals $\overline{be}$ and $\overline{fa}$ is called an *hourglass*, which is *open* if $\pi(a,b) \cap \pi(e,f) = \emptyset$ and *closed* otherwise (see Fig. 3). If H_C is open, then $\pi(a,b)$ and $\pi(e,f)$ are called *sides* of H_C; otherwise H_C consists of two "funnels" and a path $\pi_C = \pi(a,b) \cap \pi(e,f)$ joining the two apices of the two funnels, called the *corridor path* of C. The two funnel apices connected by the corridor path are called the *corridor path terminals*.

Let $\mathcal{M}$ be the union of all $O(h)$ junction triangles, open hourglasses, and funnels. We call $\mathcal{M}$ the *ocean*. Denote by $SPM(\mathcal{F})$ the SPM we want to compute on the free space $\mathcal{F}$ (with respect to the source point s), and denote by $SPM(\mathcal{M})$ the portion of $SPM(\mathcal{F})$ in $\mathcal{M}$. After the free space is triangulated in $O(T)$ time, the algorithm in [5] computes $SPM(\mathcal{M})$ of size $O(n)$ in $O(n + h \log h)$ time and $O(n)$ space, based on the following observation: For any point $t \in \mathcal{M}$, there exists a shortest s-t path $\pi(s,t)$ in $\mathcal{F}$ such that $\pi(s,t)$ is in $\mathcal{M}$ but possibly contains some corridor paths. Our task in this paper is to compute the portion of $SPM(\mathcal{F})$ in the space $\mathcal{F} \setminus \mathcal{M}$, in additional $O(n)$ time. Below, we partition the space $\mathcal{F} \setminus \mathcal{M}$ into two types of regions: *bays* and *canals*. Consider the hourglass H_C of a corridor C. Depending on whether H_C is open or closed, there are two cases.

If H_C is open (see Fig. 3), then H_C has two sides. Let $S_1(H_C)$ be an arbitrary side of H_C. The obstacle vertices on $S_1(H_C)$ all lie on the same obstacle, say $P \in \mathcal{P}$. Let c and d be any two adjacent vertices on $S_1(H_C)$ such that the line segment $\overline{cd}$ is not an edge of P (see the left figure in Fig. 3, with $P = P_j$). The region enclosed by $\overline{cd}$ and a boundary portion of P between c and d is called the *bay* of $\overline{cd}$ and P, denoted by $bay(\overline{cd})$, which is a simple polygon. We call $\overline{cd}$ the *bay gate* of $bay(\overline{cd})$, which is a common edge of $bay(\overline{cd})$ and $\mathcal{M}$.

If the hourglass H_C is closed, then let x and y be the two apices of its two funnels. Consider two adjacent vertices c and d on a side of a funnel such that the line segment $\overline{cd}$ is not an obstacle edge. If neither c nor d is a funnel apex, then c and d must both lie on the same obstacle and the segment $\overline{cd}$ also defines a bay with that obstacle. However, if either c or d is a funnel apex, say, $x = c$, then x and d may lie on different obstacles. If they lie on the same obstacle, then they also define a bay; otherwise, we call $\overline{xd}$ the *canal gate* at x (see Fig. 3). Similarly, there is also a canal gate at the other funnel apex y, say $\overline{yz}$. Let P_i and P_j be the two obstacles bounding the hourglass H_C. The obstacle-free region enclosed by P_i, P_j, and the two canal gates $\overline{xd}$ and $\overline{yz}$ that contain the corridor path of H_C is called the *canal* of H_C, denoted by $canal(x,y)$, which is a simple polygon. Similarly, the two canal gates are common edges of the canal and $\mathcal{M}$.

Clearly, all the bays and canals together constitute the space $\mathcal{F} \setminus \mathcal{M}$. Note that bays and canals are connected with $\mathcal{M}$ only through their gates.

3 The Algorithm Outline

Our task is to compute $SPM(\mathcal{F})$. To this end, again, $SPM(\mathcal{M})$ has already been computed in [5], our task in this paper is to compute the portion of $SPM(\mathcal{F})$ in $\mathcal{F} \setminus \mathcal{M}$, i.e., all bays and canals, or in other words, compute an SPM for each bay/canal. More intuitively, we "expand" $SPM(\mathcal{M})$ to all bays/canals through their gates, in additional $O(n)$ time.

Recall that an SPM is a partition of the free space into many cells; each cell C has a root point r such that for any point p in C, a shortest path from the source point s to p consists of the line segment $\overline{pr}$ and a shortest path from s to r. Further, $\overline{pr}$ is in C (i.e., C is a star-shaped polygon with r in the kernel). Refer to [18, 19] for more details on SPM.

We discuss the bays first. Consider a bay $bay(\overline{cd})$. Since its gate $\overline{cd}$ is also an edge of $\mathcal{M}$, $\overline{cd}$ is adjacent to some cells in $SPM(\mathcal{M})$. If $\overline{cd}$ is in a single cell $C(r)$ of $SPM(\mathcal{M})$ with r as the root, then each point in $bay(\overline{cd})$ has a shortest path to s via r. Thus, to construct an SPM for $bay(\overline{cd})$, it suffices to compute an SPM on $bay(\overline{cd})$ with respect to the point r, which can be done in linear time (in terms of the number of vertices of $bay(\overline{cd})$) since $bay(\overline{cd})$ is a simple polygon[1]. Note that although r may not be a point in $bay(\overline{cd})$, we can, for example, connect r to both c and d with two line segments to form a new simple polygon that contains $bay(\overline{cd})$.

If the gate $\overline{cd}$ is not contained in a single cell of $SPM(\mathcal{M})$, then $\overline{cd}$ intersects multiple cells in $SPM(\mathcal{M})$. When computing an SPM for $bay(\overline{cd})$, we must consider the roots of all such cells and each root has an additive weight that is the length of its shortest path to s. In this case, multiple vertices of $SPM(\mathcal{M})$ (i.e., the intersections of the boundaries of the cells of $SPM(\mathcal{M})$ with $\overline{cd}$) may lie in the interior of $\overline{cd}$. We call the vertices of $SPM(\mathcal{M})$ on $\overline{cd}$ (including its endpoints c and d) the *$SPM(\mathcal{M})$ vertices*. Later in Section 4, we give an algorithm for the following result.

▶ **Theorem 1.** *For a bay of n' vertices with m' $SPM(\mathcal{M})$ vertices on its gate, an SPM of size $O(n'+m')$ for the bay can be computed in $O(n'+m')$ time.*

Since a canal has two gates which are also edges of $\mathcal{M}$, multiple $SPM(\mathcal{M})$ vertices may lie on both its gates. Later in Section 5, we give an algorithm for the following Theorem 2, which uses the algorithm for Theorem 1 as a main procedure.

▶ **Theorem 2.** *For a canal of n' vertices with totally m' $SPM(\mathcal{M})$ vertices on its two gates, an SPM of size $O(n'+m')$ can be computed in $O(n'+m')$ time.*

By Theorems 1 and 2, the total time for computing the SPMs for all bays and canals is linear in terms of the total sum of the numbers of obstacle vertices of all bays and canals (which is $O(n)$) and the total number of the $SPM(\mathcal{M})$ vertices on the gates of all bays and canals (which is also $O(n)$ since the size of $SPM(\mathcal{M})$ is $O(n)$). We hence conclude that after $SPM(\mathcal{M})$ is obtained, an SPM for $\mathcal{F}$ can be computed in additional $O(n)$ time. Together with a planar point location data structure [10, 17], we have the following result.

▶ **Theorem 3.** *After the free space $\mathcal{F}$ is triangulated in $O(T)$ time, an $SPM(\mathcal{F})$ of size $O(n)$ can be built in $O(n+h\log h)$ time and $O(n)$ space.*

4 Expanding the $SPM(\mathcal{M})$ into a Bay (a Sketch)

In this section, we sketch our algorithm for Theorem 1, and all details can be found in [6].

Consider a bay $bay(\overline{cd})$ with gate $\overline{cd}$ (see Fig. 3). Denote by n' the number of vertices of $bay(\overline{cd})$. Let $SPM(bay(\overline{cd}))$ be an SPM for $bay(\overline{cd})$ that we seek to compute. If $\overline{cd}$ lies in a single cell of $SPM(\mathcal{M})$, we have shown $SPM(bay(\overline{cd}))$ can be computed in $O(n')$ time. This section focuses on the case when $\overline{cd}$ is not contained in a single cell of $SPM(\mathcal{M})$. Denote by m' the number of $SPM(\mathcal{M})$ vertices on $\overline{cd}$. Our task is to compute $SPM(bay(\overline{cd}))$ in $O(n'+m')$ time. Let R be the set of roots of the cells of $SPM(\mathcal{M})$ that intersect with $\overline{cd}$.

To obtain $SPM(bay(\overline{cd}))$, we first compute, for each $r \in R$, the *Voronoi region* $VD(r)$ inside $bay(\overline{cd})$ such that for any point $t \in VD(r)$, there is a shortest s-t path going through

[1] As the Euclidean shortest path between two points in a simple polygon is also an L_1 shortest path [13], a Euclidean SPM [12] in a simple polygon is also an L_1 one.

r; we then compute an SPM on $VD(r)$ with respect to the single point r, which can be done in linear time since $VD(r)$ is a simple polygon. Thus, the key is to decompose $bay(\overline{cd})$ into Voronoi regions for the roots of R (which is the challenging subproblem mentioned in Section 1.1). Denote by $VD(bay(\overline{cd}))$ this Voronoi diagram decomposition of $bay(\overline{cd})$. We aim to compute $VD(bay(\overline{cd}))$ in $O(n' + m')$ time.

Without loss of generality (W.l.o.g.), assume that $\overline{cd}$ is positive-sloped, $bay(\overline{cd})$ is on the right of $\overline{cd}$, and the vertex c is higher than d (e.g., $bay(\overline{cd}) = P$ in Fig. 1). Let $R = \{r_1, r_2, \ldots, r_k\}$ be the set of roots of the cells of $SPM(\mathcal{M})$ that intersect with $\overline{cd}$ in the order from c to d along $\overline{cd}$. Note that R may be a multi-set, i.e., two roots r_i and r_j with $i \neq j$ may refer to the same physical point; but this is not important to our algorithm (e.g., we can view each r_i as a physical copy of the same root). Let $c = v_0, v_1, \ldots, v_k = d$ be the $SPM(\mathcal{M})$ vertices on $\overline{cd}$ ordered from c to d (thus $m' = k + 1$). Hence, for each $1 \leq i \leq k$, the segment $\overline{v_{i-1}v_i}$ is on the boundary of the cell $C(r_i)$ of $SPM(\mathcal{M})$. To obtain $VD(bay(\overline{cd}))$, for each $r_i \in R$, we need to compute the Voronoi region $VD(r_i)$.

Our algorithm can be viewed as an incremental one, i.e., it considers the roots in R one by one. It is commonly known that incremental approaches can construct Voronoi diagrams in quadratic time, or may give good randomized results. In contrast, our algorithm is deterministic and takes only linear time. The success of it hinges on that we can find an *order* of the roots in R such that by following this order to consider the roots in R incrementally, we are able to compute $VD(bay(\overline{cd}))$ in linear time. The order is nothing but that of the indices of the roots in R we have defined. With this order, the algorithm is conceptually simple. However, it is quite challenging to argue its correctness and achieve a linear time implementation. Our strategy is to show that the algorithm implicitly maintains a number of *invariants* that assure the correctness of the algorithm. For this purpose, we discover many observations that capture some essential properties of this L_1 problem.

4.1 Algorithm Overview

To compute $VD(bay(\overline{cd}))$, it turns out that we need to deal with the interactions between some horizontal and vertical rays, each of which belongs to the bisector of two roots in R. Further, considering the roots in R incrementally is equivalent to considering the corresponding rays incrementally. We process these rays in a certain order (e.g., as to be proved, their origins somehow form a staircase structure). For each ray considered, if it is vertical, then it is easy (it eventually leads to a ray shooting operation), and its processing does not introduce any new ray. But, if it is horizontal, then the situation is more complicated since its processing may introduce many new horizontal rays and (at most) one vertical ray, also in a certain order along a staircase structure (in addition to causing a ray shooting operation). A stack is used to store certain vertical rays that need to be further processed.

The algorithm needs to perform ray shooting operations for some vertical and horizontal rays. Although there are known data structures for ray shooting queries [3, 4, 12, 14], they are not efficient enough for a linear time implementation of the entire algorithm. Based on observations, we use the horizontal visibility map and vertical visibility map of $bay(\overline{cd})$ [2]. More specifically, we prove that all vertical ray shootings are in a "nice" sorted order (called *target-sorted*). With this property, all vertical ray shootings are performed in totally linear time by using the vertical visibility map of $bay(\overline{cd})$. The horizontal visibility map is used to guide the overall process of the algorithm. During the algorithm, we march into the bay and the horizontal visibility map allows us to keep track of our current position (i.e., in a trapezoid of the map that contains our current position). The horizontal visibility map also allows each horizontal ray shooting to be done in $O(1)$ time. In addition, in the preprocessing

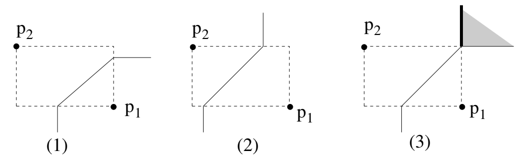


Figure 4 The L_1 bisector $B(p_1, p_2)$ of two weighted points p_1 and p_2. In (3), an entire quadrant (the shaded area) is $B(p_1, p_2)$, but we choose $B(p_1, p_2)$ to be the vertical (solid thick) half-line.

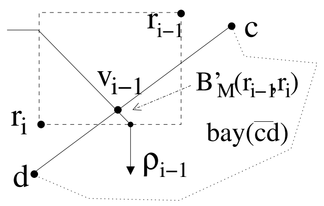


Figure 5 Illustrating the definition of ρ_{i-1}.

of the algorithm, we also need to perform some other ray shootings (for rays of slope -1); our linear time solution for this also hinges on the target-sorted property of such rays.

Our algorithm is conceptually simple. The only data structures we need are linked lists, a stack, and the horizontal and vertical visibility maps. Again, it is much more difficult to argue the correctness of the algorithm, making the presentation of this paper lengthy, technically complicated, or even tedious, for which we ask for the reader's patience.

Below, we sketch how our algorithm works, and the proof of its correctness is omitted. We may also use some terminology of natural meaning without definitions (their formal definitions are can also be found in [6].

4.2 The Algorithm

Each root $r_i \in R$ can be viewed as an additively weighted point whose weight is the length of an L_1 shortest path from s to r_i. For any two weighted points p_1 and p_2 with weights w_1 and w_2, respectively, their bisector $B(p_1, p_2)$ can be an entire quadrant of the plane (e.g., see Fig. 4); in this case, as in [18, 19], we choose a vertical half-line as the bisector. Denote by $Rec(p_1, p_2)$ the rectangle with p_1 and p_2 as its two diagonal vertices. Thus, as illustrated in Fig. 4, $B(p_1, p_2)$ consists of three portions, two rays whose origins are on the boundary of $Rec(p_1, p_2)$ and an open line segment (called *middle segment* and denoted by $B_M(p_1, p_2)$) in $Rec(p_1, p_2)$ connecting the origins the two rays; further each ray is either horizontal or vertical and the middle segment is of slope 1 or -1.

For any pair of consecutive roots r_{i-1} and r_i in R for $2 \leq i \leq k$, since v_{i-1} is on the common boundary of the cells $C(r_{i-1})$ and $C(r_i)$, v_{i-1} lies on the bisector $B(r_{i-1}, r_i)$ of r_{i-1} and r_i. But v_{i-1} may lie on either a ray or the middle segment of $B(r_{i-1}, r_i)$. If v_{i-1} lies on a ray of $B(r_{i-1}, r_i)$, let ρ_{i-1} denote the ray; otherwise, v_{i-1} partitions $B_M(r_{i-1}, r_i)$ into two portions and one portion (denoted by $B'_M(r_{i-1}, r_i)$) intersects the interior of $bay(\overline{cd})$, and we let ρ_{i-1} be the ray of $B(r_{i-1}, r_i)$ connecting to $B'_M(r_{i-1}, r_i)$ (see Fig. 5). In either case, ρ_{i-1} is either vertically going south (downwards) or horizontally going east (rightwards). (If $B_M(r_{i-1}, r_i)$ does not intersect $\overline{cd}$, we let $B'_M(r_{i-1}, r_i) = \emptyset$.) Further, if $v_{i-1} \in B_M(r_{i-1}, r_i)$, $B_M(r_{i-1}, r_i)$ must be (-1)-sloped [6]. Denote by $or(\rho)$ the origin of a ray ρ. We can prove that the origins $or(\rho_1), or(\rho_2), \ldots, or(\rho_{k-1})$ follow the order from *northeast* to *southwest* [6].

Let ∂ denote the boundary of $bay(\overline{cd})$ excluding $\overline{cd}$. The algorithm first determines for each $2 \leq i \leq k$ whether $B'_M(r_{i-1}, r_i)$ intersects ∂, which is done by a set of (-1)-sloped ray shootings. If, say $B'_M(r_{i-1}, r_i)$, intersects ∂ at a point p such that $\overline{v_{i-1}p}$ is inside $bay(\overline{cd})$, then $\overline{v_{i-1}p}$ partitions $bay(\overline{cd})$ into two simple polygons bay_1 and bay_2 such that bay_1 contains $\overline{cv_{i-1}}$ as an edge (see Fig. 6). We show (in [6]) that $\overline{v_{i-1}p}$ must *appear in* $VD(bay(\overline{cd}))$ (which means that $\overline{v_{i-1}p}$ lies on some cell boundaries of $VD(bay(\overline{cd}))$) and the original problem of computing $VD(bay(\overline{cd}))$ on $bay(\overline{cd})$ and R can be broken into two subproblems of computing

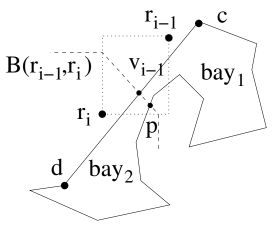


Figure 6 $B_M(r_{i-1}, r_i)$ intersects both $\overline{cd}$ (at v_{i-1}) and ∂ (at p). The line segment $\overline{v_{i-1}p}$ divides $bay(\overline{cd})$ into bay_1 and bay_2.

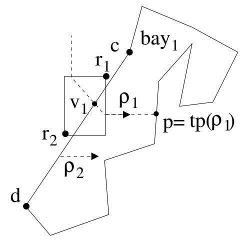


Figure 7 Illustrating an example of ρ_1 being horizontal.

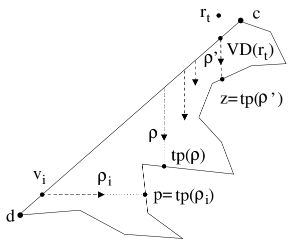


Figure 8 The target points of all rays in S (whose rays are all vertical) are before $p = tp(\rho_i)$. The ray ρ is at the top of S and ρ' is at the bottom of S.

$VD(bay_1)$ on bay_1 and $\{r_1, \ldots, r_{i-1}\}$ and computing $VD(bay_2)$ on bay_2 and $\{r_i, \ldots, r_k\}$. The above procedure is done in the preprocessing of the algorithm, where all (-1)-sloped ray shootings are solved in totally $O(n' + k)$ time. Thus, we only need to focus on each individual subproblem. W.l.o.g., we assume the original problem is one subproblem (i.e., no $B'_M(r_{i-1}, r_i)$ intersects ∂). We can show that each $B'_M(r_{i-1}, r_i)$ appears in $VD(bay(\overline{cd}))$ [6]. To compute $VD(bay(\overline{cd}))$, essentially we need to handle the interactions of all rays $\rho_1, \ldots, \rho_{k-1}$. Let $\Psi = \{\rho_1, \ldots, \rho_{k-1}\}$. Considering the roots in R incrementally is equivalent to considering the corresponding rays in Ψ incrementally. Specifically, our algorithm processes the rays in Ψ from ρ_1 to ρ_{k-1} incrementally. Recall that each $\rho_i \in \Psi$ is either vertically going south or horizontally going east. We use a stack S to store certain vertical rays and $S = \emptyset$ initially. As will be seen later, some rays in S may not be in Ψ. For a ray ρ with its origin in $bay(\overline{cd})$, the point on ∂ hit first by ρ is called the *target point* of ρ, denoted by $tp(\rho)$.

Our algorithm maintains a number of invariants, and the next paragraph lists a subset of them that are related to our discussion in this section. The complete list of invariants (as well as the argument why our algorithm implicitly maintains them) are in [6].

Let $\hat{\rho}$ be the next ray to be considered by the algorithm. Assume $S \neq \emptyset$. **Invariants:** (a) All rays in S are vertically going south. (b) The origins of all rays in S from top to bottom are ordered from southwest to northeast. (c) The origin of $\hat{\rho}$ is to the southwest of the origin of the ray at the top of S. (d) Suppose $\hat{\rho}$ is on a bisector $B(r_j, r_i)$ with $j < i$ and the ray at the top of S is on a bisector $B(r_t, r_{t'})$ with $t < t'$; then $j = t'$. (e) For each ray ρ'' in $S \cup \{\hat{\rho}\}$, suppose ρ'' lies on a bisector $B(r_{j'}, r_{i'})$ of two roots $r_{j'}$ and $r_{i'}$ with $j' < i'$; then the portion of the boundary of the Voronoi region $VD(r_{i'})$ (resp., $VD(r_{j'})$) from $v_{i'-1}$ (resp., $v_{j'}$) to the origin $or(\rho'')$ of ρ'' has already been computed. (f) The target points of all rays in S from bottom to top are ordered clockwise on ∂, i.e., from c to d (this property is called "target-sorted").

Consider the first ray ρ_1. If ρ_1 is vertical, we push it on S and continue to consider the ray $\rho_2 \in \Psi$. If ρ_1 is horizontal, we find the target point $p = tp(\rho_1)$ of ρ_1 (i.e., the first point on ∂ hit by ρ_1) by performing a horizontal ray shooting. Then, $B'_M(r_1, r_2)$ and $\overline{or(\rho_1)p}$ together partition $bay(\overline{cd})$ into two simple polygons, one of which contains $\overline{cv_1}$ as an edge and we denote it by bay_1 (see Fig. 7). We show that bay_1 is the Voronoi region of r_1, i.e., $VD(r_1) = bay_1$. We then continue to consider the ray $\rho_2 \in \Psi$.

Now consider a general step of the algorithm, which processes a ray $\rho_i \in \Psi$. If ρ_i is vertical, we simply push ρ_i on the top of the stack S and continue to consider the next ray $\rho_{i+1} \in \Psi$. Below, we discuss the case when ρ_i is horizontal. Let $p = tp(\rho_i)$ be the target point of ρ_i. If $S = \emptyset$, then as in the case of ρ_1, the Voronoi region $VD(r_i)$ is determined immediately (note that $\rho_i \in B(r_i, r_{i+1})$), and we continue to consider ρ_{i+1}. Below, we assume

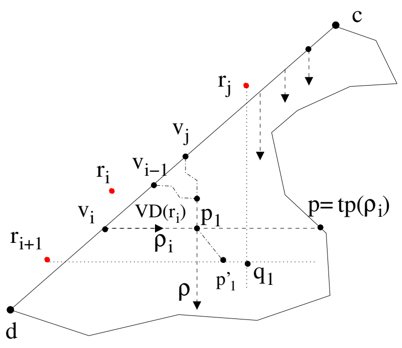


Figure 9 Illustrating an example that the ray ρ at the top of S intersects ρ_i (at p_1) before hitting ∂.

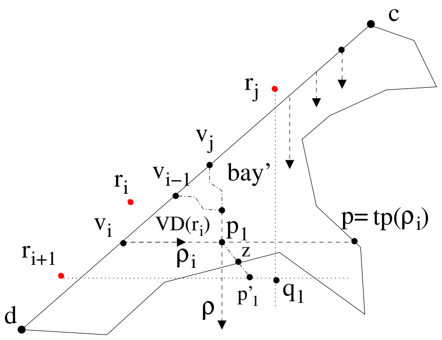


Figure 10 Illustrating an example that $B'_M(r_j, r_{i+1})$ $(= \overline{p_1p'_1})$ intersects ∂ (first at z).

$S \neq \emptyset$. For any two points a and b on ∂ with a lying in the portion of ∂ from the vertex c clockwise to b, we say a is *before* b or b is *after* a. Suppose ρ is the ray on the top of S. By Invariant (b), ρ is the leftmost ray in S. If the target point $tp(\rho)$ is before p, then by Invariants (f), ρ_i does not intersect any vertical ray in S before they hit ∂ (see Fig. 8). We then perform a *splitting procedure* on the rays in S, as follows.

Let ρ' be the ray at the *bottom* of S and $z = tp(\rho')$ (see Fig. 8). Suppose ρ' is on a bisector, say $B(r_t, r_{t'})$ for some $t < t'$. By Invariant (e), the boundary portion of $VD(r_t)$ between v_t and the origin $or(\rho')$ has been computed. The concatenation of the segment $\overline{or(\rho')z}$ and the above boundary portion of $VD(r_t)$ splits the current region of $bay(\overline{cd})$ that needs to be further decomposed for computing $VD(bay(\overline{cd}))$ into two simple polygons. The one containing $\overline{v_{t-1}v_t}$ is the Voronoi region $VD(r_t)$. We then continue to process the second bottom ray in S in a similar fashion. This splitting procedure stops once all rays in S are processed. The target points of all rays in S are found by vertical ray shootings, which is done by a *scanning procedure* that basically scans a portion of ∂. Finally, we pop all rays out of S. We then continue to consider the next ray $\rho_{i+1} \in \Psi$.

If $tp(\rho)$ is after p, then ρ_i intersects ρ before they hit ∂ (e.g., see Fig. 9). Suppose ρ is on $B(r_j, r_{j'})$ with $j < j'$. Recall $\rho_i \in B(r_i, r_{i+1})$. By Invariant (d), $j' = i$, and the Voronoi region $VD(r_i)$ can be determined immediately [6]. Let p_1 be the intersection of ρ_i and ρ (see Fig. 9). Let q_1 be the intersection of the vertical line through r_j and the horizontal line through r_{i+1}. We show (in [6]) that q_1 must be to the southeast of p_1. The line of slope -1 through p_1 intersects the boundary of the rectangle $Rec(p_1, q_1)$ at two points: One is p_1 and denote the other one by p'_1 (see Fig. 9). We show that $\overline{p_1p'_1} \subseteq B(r_j, r_{i+1})$ and $\overline{p_1p'_1} \cap bay(\overline{cd})$ appears in $VD(bay(\overline{cd}))$. Depending on whether $\overline{p_1p'_1}$ intersects ∂, there are two cases.

If $\overline{p_1p'_1}$ intersects ∂, let z be the first intersection point (see Fig. 10). Similarly as before, the line segment $\overline{p_1z}$ appears in $VD(bay(\overline{cd}))$ and partitions the current region of $bay(\overline{cd})$ that needs further decomposition for computing $VD(bay(\overline{cd}))$ into two simple polygons; one of them, say bay', contains the point p. Then, the Voronoi regions of the roots that define the rays in S form a decomposition of bay', and we use a procedure similar to the splitting procedure discussed earlier to compute this decomposition of bay', i.e., consider the rays in S from bottom to top. Finally, we pop all rays out of S, and continue with the next ray $\rho_{i+1} \in \Psi$.

If $\overline{p_1p'_1}$ does not intersect ∂, then depending on whether p'_1 is on the bottom edge or the right edge of the rectangle $Rec(p_1, q_1)$, there are further two subcases. In either subcase, we first pop ρ out of S. If p'_1 is on the bottom edge, then let ρ^*_i be the vertical ray originating at p'_1 and going south (see Fig. 11). We call ρ^*_i the *termination vertical ray* of ρ_i. We push

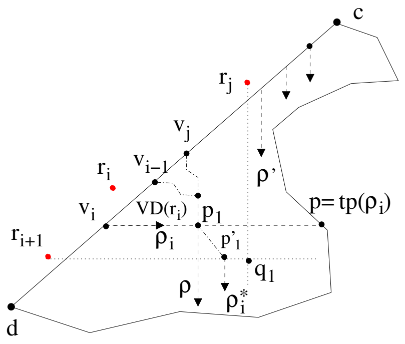


Figure 11 Illustrating an example that the point p'_1 $(= or(\rho^*_i))$ is on the bottom edge of $Rec(p_1, q_1)$.

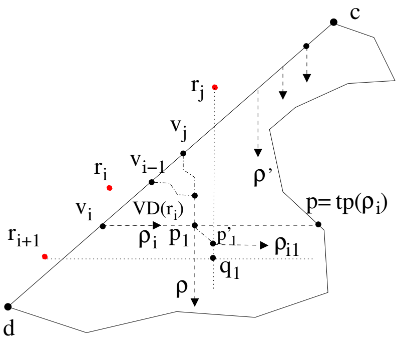


Figure 12 Illustrating an example that the point p'_1 $(= or(\rho_{i1}))$ is on the right edge of $Rec(p_1, q_1)$.

ρ^*_i on the top of S. We then continue to consider the next ray $\rho_{i+1} \in \Psi$. If p'_1 is on the right edge of $Rec(p_1, q_1)$, then let ρ_{i1} be the horizontal ray originating at p'_1 and going east (see Fig. 12). We call ρ_{i1} a *successor horizontal ray* of ρ_i. The ray ρ_{i1} (not ρ_{i+1}) is the next ray that will be considered by the algorithm. Note that $\rho_{i1} \notin \Psi$. We then continue with processing ρ_{i1}. We should mention that although our discussion above is on a ray ρ_i in Ψ, the processing for (the horizontal) ρ_{i1} is very similar to the case when $\rho_i \in \Psi$ is horizontal. In particular, there may also be a termination vertical ray or a successor horizontal ray generated after processing ρ_{i1}. Thus, the processing of a horizontal ray $\rho_i \in \Psi$ may lead to generating multiple successor horizontal rays but at most one termination vertical ray, i.e., a successor horizontal ray may generate another successor horizontal ray (e.g., see Fig. 13), but a termination vertical ray does not generate another ray.

We have discussed all possible cases for processing a ray. The algorithm finishes when the Voronoi regions for all roots in R are computed.

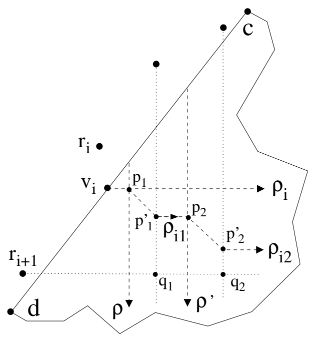


Figure 13 Illustrating the first two successor horizontal rays ρ_{i1} and ρ_{i2} of a horizontal ray $\rho_i \in \Psi$.

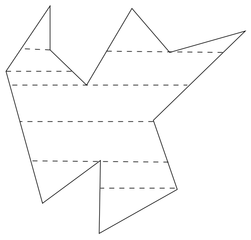


Figure 14 Illustrating the horizontal visibility map of a simple polygon.

The running time of the algorithm is $O(n' + m')$ (recall $m' = k + 1$ and n' is the number of vertices in $bay(\overline{cd})$). For the implementation, in the preprocessing we also compute a horizontal visibility map $HM(bay(\overline{cd}))$ (see Fig. 14) and a vertical visibility map $VM(bay(\overline{cd}))$ [2]. We do not overlap the two maps. In the main algorithm, we use $HM(bay(\overline{cd}))$ to guide the computation, i.e., we keep track of which trapezoid of $HM(bay(\overline{cd}))$ we are in during the algorithm. This allows each horizontal ray shooting to be performed in constant time. We also use $HM(bay(\overline{cd}))$ to compute the first intersection point of $\overline{p_1 p'_1}$ and ∂ (i.e., the point z in Fig. 10). To conduct the vertical ray shootings (i.e., for the rays in S), we utilize the vertical map $VM(bay(\overline{cd}))$ and a scanning procedure. Further, with Invariant (f), we can show that the target points of the vertical rays in the entire algorithm that we need to compute are ordered on ∂ from c to d (i.e., the target-sorted property). In addition, we use a *reference point* p^* to help implement the vertical ray shootings. The reference point p^*, which is at c (resp., d) at the beginning (resp., end) of the algorithm, always moves (forward) on ∂ from c to d during the algorithm but it never moves backward. These components together perform all vertical ray shootings in totally $O(n' + m')$ time. Note that although there are known data structures for general ray shootings [3, 4, 12, 14], they are not efficient enough for our purpose. Also note that although the processing of a horizontal ray ρ_i in Ψ may produce multiple successor horizontal rays, we can show that the total number of horizontal rays in the entire algorithm is at most k and that of vertical rays is also at most k.

5 Expanding $SPM(\mathcal{M})$ into a Canal (a Sketch)

We sketch the idea of computing an SPM for a canal, say $canal(x, y)$. The details are in [6]. A main difference than the bay case is that a canal has two gates, say $\overline{xd}$ and $\overline{yz}$ (e.g., see Fig. 3). Let R_1 (resp., R_2) be the set of roots whose cells in $SPM(\mathcal{M})$ intersect $\overline{xd}$ (resp., $\overline{yz}$). Let $VD(canal(x, y), R_1)$ denote the weighted Voronoi diagram of $canal(x, y)$ with respect to R_1, i.e., we treat $canal(x, y)$ as a bay with the gate $\overline{xd}$. Define $VD(canal(x, y), R_2)$ similarly.

We first compute $VD(canal(x, y), R_1)$ and $VD(canal(x, y), R_2)$ by our algorithm for a bay in Section 4. We then find a "dividing curve" γ in $canal(x, y)$ that divides $canal(x, y)$ into two simple polygons C_1 and C_2, such that each point in C_1 (resp., C_2) has a shortest path from s via a root in R_1 (resp., R_2). We apply our algorithm for a bay on C_1 and R_1 to compute the weighted Voronoi diagram of C_1 with respect to R_1, i.e., $VD(C_1, R_1)$. We similarly compute $VD(C_2, R_2)$. It is easy to see that $SPM(canal(x, y))$ is a concatenation of $VD(C_1, R_1)$ and $VD(C_2, R_2)$. It remains to compute the dividing curve γ.

To compute γ, we first determine a point $p^* \in \gamma$ (e.g., with the help of the corridor path). Then, we trace γ out from p^* by traversing the cells of $VD(canal(x, y), R_1)$ and $VD(canal(x, y), R_2)$, which is similar to the merge procedure of the divide-and-conquer Voronoi diagram algorithm [20].

References

1 R. Bar-Yehuda and B. Chazelle. Triangulating disjoint Jordan chains. *International Journal of Computational Geometry and Applications*, 4(4):475–481, 1994.

2 B. Chazelle. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry*, 6:485–524, 1991.

3 B. Chazelle, H. Edelsbrunner, M. Grigni, L. Gribas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12(1):54–68, 1994.

4 B. Chazelle and L. Guibas. Visibility and intersection problems in plane geometry. *Discrete and Computational Geometry*, 4:551–589, 1989.

5 D.Z. Chen and H. Wang. A nearly optimal algorithm for finding L_1 shortest paths among polygonal obstacles in the plane. In *Proc. of the 19th European Symposium on Algorithms*, pages 481–492, 2011.

6 D.Z. Chen and H. Wang. Computing L_1 shortest paths among polygonal obstacles in the plane. arXiv:1202.5715v1, 2012.

7 D.Z. Chen and H. Wang. Computing the visibility polygon of an island in a polygonal domain. In *Proc. of the 39th International Colloquium on Automata, Languages and Programming*, pages 218–229, 2012.

8 K. Clarkson, S. Kapoor, and P. Vaidya. Rectilinear shortest paths through polygonal obstacles in $O(n \log^2 n)$ time. In *Proc. of the 3rd Annual Symposium on Computational Geometry*, pages 251–257, 1987.

9 P.J. de Rezende, D.T. Lee, and Y.F. Wu. Rectilinear shortest paths in the presence of rectangular barriers. *Discrete and Computational Geometry*, 4:41–53, 1989.

10 H. Edelsbrunner, L. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15(2):317–340, 1986.

11 S. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

12 L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R.E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1-4):209–233, 1987.

13 J. Hershberger and J. Snoeyink. Computing minimum length paths of a given homotopy class. *Computational Geometry: Theory and Applications*, 4(2):63–97, 1994.

14 J. Hershberger and S. Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *Journal of Algorithms*, 18(3):403–431, 1995.

15 R. Inkulu and S. Kapoor. Planar rectilinear shortest path computation using corridors. *Computational Geometry: Theory and Applications*, 42(9):873–884, 2009.

16 S. Kapoor, S.N. Maheshwari, and J.S.B. Mitchell. An efficient algorithm for Euclidean shortest paths among polygonal obstacles in the plane. *Discrete and Computational Geometry*, 18(4):377–383, 1997.

17 D. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

18 J.S.B. Mitchell. An optimal algorithm for shortest rectilinear paths among obstacles. Abstracts of the *1st Canadian Conference on Computational Geometry*, 1989.

19 J.S.B. Mitchell. L_1 shortest paths among polygonal obstacles in the plane. *Algorithmica*, 8(1):55–88, 1992.

20 M.I. Shamos and D. Hoey. Closest-point problems. In *Proc. of the 16th Annual Symposium on Foundations of Computer Science*, pages 151–162, 1975.

21 P. Widmayer. On graphs preserving rectilinear shortest paths in the presence of obstacles. *Annals of Operations Research*, 33(7):557–575, 1991.

Quantifier Alternation in Two-Variable First-Order Logic with Successor Is Decidable*

Manfred Kufleitner and Alexander Lauser

University of Stuttgart, FMI, Germany
`{kufleitner,lauser}@fmi.uni-stuttgart.de`

Abstract

We consider the quantifier alternation hierarchy within two-variable first-order logic $\mathrm{FO}^2[<, \mathrm{suc}]$ over finite words with linear order and binary successor predicate. We give a single identity of omega-terms for each level of this hierarchy. This shows that for a given regular language and a non-negative integer m it is decidable whether the language is definable by a formula in $\mathrm{FO}^2[<, \mathrm{suc}]$ which has at most m quantifier alternations. We also consider the alternation hierarchy of unary temporal logic $\mathrm{TL}[\mathsf{X}, \mathsf{F}, \mathsf{Y}, \mathsf{P}]$ defined by the maximal number of nested negations. This hierarchy coincides with the $\mathrm{FO}^2[<, \mathrm{suc}]$ quantifier alternation hierarchy.

1998 ACM Subject Classification F.4.1 Mathematical Logic, F.4.3 Formal Languages.

Keywords and phrases automata theory, semigroups, regular languages, first-order logic

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.305

1 Introduction

Around 1960, Büchi, Elgot and Trakhtenbrot independently showed that monadic second-order logic (MSO) over finite words defines the class of regular languages [2, 6, 33]. Since then numerous fragments of MSO have been considered. A theoretical motivation for fragments is the study of the rich structure within the regular languages. For this purpose, fragments form the basis of a descriptive complexity theory: The simpler the formula for defining a language is, the simpler this language is. From a practical point of view, simpler fragments often lead to more efficient algorithms for decision problems such as satisfiability.

The most prominent fragment of MSO is first-order logic FO. The atomic predicates of FO are the unary predicate $\lambda(x) = a$ stating that position x is labeled by the letter a, and the binary predicates $x = y$ and $x < y$ with the natural interpretation. The successor predicate $\mathrm{suc}(x, y)$ is easily definable in FO by saying that $x < y$ and that there is no position between x and y. McNaughton and Papert showed that a language is FO-definable if and only if it is star-free [18]. Combined with Schützenberger's characterization of star-free languages in terms of finite aperiodic monoids [21], it follows that a language is FO-definable if and only if its syntactic monoid is aperiodic. The latter property is decidable and one can thus effectively check whether a regular language (given *e.g.* by a nondeterministic automaton or an MSO formula) is definable in FO. The two most famous hierarchies within FO are the Straubing-Thérien hierarchy and Brzozowski's dot-depth hierarchy. The Straubing-Thérien hierarchy coincides with the quantifier alternation inside FO without the successor predicate [25, 29], and Brzozowski's dot-depth hierarchy is captured by quantifier alternation including the successor predicate [3]; see also [20, 31]. Here, quantifier alternation is defined in terms of blocks of quantifiers for formulae in prenex normal form. Note that

* The authors were supported by the German Research Foundation (DFG) under grant DI 435/5-1.

by introducing new variables, every formula is equivalent to a formula in prenex normal form. Deciding membership of level m for these hierarchies is one of the most challenging open problems in automata theory. To date only the very first levels (*i.e.*, $m = 1$) of both hierarchies are known to be decidable [9, 24].

By Kamp's Theorem, first-order logic FO^3 with only three different names for the variables and full first-order logic FO have the same expressive power [8]. However, two variables are not sufficient for defining all first-order definable languages. The fragment FO$^2[<]$ without successor predicate has a huge number of different characterizations; see *e.g.* [5, 28]. One of them is the variety **DA** of finite monoids [22]; *cf.* [30]. For quantifier alternation inside FO^2 one cannot readily rely on prenex normal forms. However, in FO^2 negations can be moved towards the atomic formulae, and hence every formula is equivalent to a negation-free counterpart. The fragment FO^2_m consists of all FO^2-formulae whose negation-free counterpart has at most m blocks of quantifiers on each path of the parse tree. Kufleitner and Weil have shown that for every $m \geq 1$ it is decidable whether a given regular language is definable in FO$^2_m[<]$ without successor predicate [16]. They have given an effective algebraic characterization in terms of levels of the Trotter-Weil hierarchy of finite monoids [34]; see also [15]. In addition, restrictions of many other characterizations of the FO$^2[<]$-definable languages admit algebraic counterparts within this hierarchy [12, 17]. The proof of Kufleitner and Weil's characterization of FO$^2_m[<]$ relies on a combinatorial tool known under the terms *ranker* [35] and *turtle program* [23]. A connection between FO$^2_m[<]$ and rankers was established by Weis and Immerman [35] and further exploited by Kufleitner and Weil [17]. Straubing has given another algebraic characterization of FO$^2_m[<]$ in terms of weakly iterated block products of $\mathcal{J}$-trivial monoids [27]. Recently, Krebs and Straubing [10] were able to use this characterization for giving identities of omega-terms for FO$^2_m[<]$, thereby obtaining another effective characterization of FO$^2_m[<]$.

In this paper, we consider the quantifier alternation hierarchy inside FO$^2[<, \mathrm{suc}]$ with successor predicate. The logic FO$^2[<, \mathrm{suc}]$ is strictly more expressive than FO$^2[<]$ without successor. Thérien and Wilke [30] have given an algebraic characterization of FO$^2[<, \mathrm{suc}]$ which, by a previous result of Almeida, is known to coincide with the decidable variety **LDA** of finite semigroups [1]; see also [4]. For every $m \geq 2$ we give a single identity of omega-terms such that a language is definable in FO$^2_m[<, \mathrm{suc}]$ if and only if its syntactic semigroup satisfies this identity. It is thus decidable whether a given regular language is FO$^2_m[<, \mathrm{suc}]$-definable.

Our proof is by induction on m with Knast's Theorem on dot-depth one [9] as base case. For $m = 1$, there is a small difference between the availability and the absence of min- and max-predicates; this is identical to the situation for dot-depth one [11]. The main ingredients of our proof are *(i)* string rewriting techniques, *(ii)* combinatorial properties of **LDA**, and *(iii)* relativization techniques for FO^2_m. As a byproduct, we show that quantifier alternation in FO$^2[<, \mathrm{suc}]$ coincides with alternation in unary temporal logic TL$[\mathsf{X}, \mathsf{F}, \mathsf{Y}, \mathsf{P}]$ where the latter is based on the nesting depth of negations. This last property can also be seen using a translation from FO^2 to unary temporal logic by Etessami, Vardi, and Wilke [7].

Missing proofs can be found in the technical report [14].

2 Preliminaries

Throughout, A denotes a finite alphabet. The set of all finite words is A^* and the set of all finite, nonempty words is A^+. Let $u = a_1 \cdots a_n$ with $a_i \in A$. The set of *positions* of u is $\mathrm{pos}(u) = \{1, \ldots, n\}$ and its *length* is $|u| = n$. If I is an interval, then $u[I]$ denotes the factor of u covered by the interval of positions $\mathrm{pos}(u) \cap I$. If $I = [i; j]$, then $u[i; j]$ is an abbreviation for $u[I]$. In particular, if $1 \leq i \leq j \leq n$, then $u[i;j] = a_i \cdots a_j$. The *k-factor alphabet* is $\operatorname{alph}_k(u) = \{a_i \cdots a_{i+k-1} \in A^k \mid 1 \leq i \leq n-k+1\}$

First-Order Logic. We consider first-order logic over finite words with order and successor predicates. Atomic first-order formulae are $\top$ for *true*, $\bot$ for *false*, label predicates $\lambda(x) = a$ with $a \in A$, comparisons $x = y$, $x < y$ and successor $\operatorname{suc}(x, y)$ as well as minimum $\min(x)$ and maximum $\max(x)$. Here x and y are variables ranging over positions of a word which forms a model as a labeled, linearly ordered set of positions. Formulae can be composed by the usual Boolean connectives, *i.e.*, if φ and ψ are first-order formulae, then so are the disjunction $\varphi \vee \psi$, the conjunction $\varphi \wedge \psi$, and the negation $\neg\varphi$. Moreover, formulae can be composed by existential quantification $\exists x\, \varphi$ and universal quantification $\forall x\, \varphi$. The semantics is as usual; see *e.g.* [13, 32]. We use the notation $\varphi(x_1, \ldots, x_n)$ to indicate that at most the variables $x_1, \ldots, x_n$ occur freely in φ. We write $u \models \varphi(i_1, \ldots, i_n)$ for $u \in A^*$ and positions $i_j \in \operatorname{pos}(u)$ if φ is true over u with x_j being interpreted by i_j. A formula without free variables is a *sentence* and in this case we simply write $u \models \varphi$. For any class $\mathcal{F}$ of first-order formulae, $\mathcal{F}[\mathcal{C}]$ is the restriction to formulae in $\mathcal{F}$ which, apart from $\top$, $\bot$, label predicates, and equality, only use predicates in $\mathcal{C} \subseteq \{<, \operatorname{suc}, \min, \max\}$.

The fragment $\mathrm{FO}^2 = \mathrm{FO}^2[<, \operatorname{suc}, \min, \max]$ of first-order logic contains all formulae which use at most two different names for variables, say x and y. For FO^2-formulae $\varphi(x)$ with free variable x we stipulate the convention that $\varphi(y)$ is the FO^2-formula obtained by interchanging x and y. Using De Morgan's laws and the usual dualities between existential and universal quantifiers, one can see that every formula in FO^2 is equivalent to a formula with negations only applied to atomic formulae. We call such formulae *negation-free* (since negations could be eliminated by adding negative predicates to an extended signature). The fragment FO^2_m consists of all formulae in FO^2 with quantifier alternation depth at most m, *i.e.*, formulae such that the negation-free counterpart has at most m blocks of quantifiers on every path of the parse tree. Therefore, if we drop the two-variable restriction, every FO^2_m-formula admits a prenex normal form with m blocks of quantifiers. In other words negation-free formulae in FO^2_m have at most $m-1$ alternations of nested existential and universal quantifiers. Note that FO^2_m is closed under negation. The fragment $\mathrm{FO}^2_{m,n}$ contains all formulae in FO^2_m with quantifier depth at most n.

Unary Temporal Logic. Unary temporal logic $\mathrm{TL}[\mathsf{X}, \mathsf{F}, \mathsf{Y}, \mathsf{P}]$ consists of all formulae built from $\top$ for *true*, $\bot$ for *false*, labels a with $a \in A$, compositions using Boolean connectives as in first-order logic, and temporal modalities $\mathsf{X}\,\varphi$, $\mathsf{F}\,\varphi$, $\mathsf{Y}\,\varphi$, and $\mathsf{P}\,\varphi$ for $\varphi \in \mathrm{TL}[\mathsf{X}, \mathsf{F}, \mathsf{Y}, \mathsf{P}]$. Formulae of unary temporal logic are interpreted over a word relative to a current position. The semantics is declared by the following FO^2-formulae in one free variable: We let $a(x) \equiv \big(\lambda(x) = a\big)$ and

$$\begin{aligned}
(\mathsf{X}\,\varphi)(x) &\equiv \exists y\,\big(\operatorname{suc}(x,y) \wedge \varphi(y)\big), & (\mathsf{F}\,\varphi)(x) &\equiv \exists y\,\big(x \leq y \wedge \varphi(y)\big),\\
(\mathsf{Y}\,\varphi)(x) &\equiv \exists y\,\big(\operatorname{suc}(y,x) \wedge \varphi(y)\big), & (\mathsf{P}\,\varphi)(x) &\equiv \exists y\,\big(y \leq x \wedge \varphi(y)\big).
\end{aligned}$$

Here and in the sequel, $\equiv$ means syntactic equality. We often use this symbol instead of equality in order to avoid confusion with the symbol $=$ occurring in atomic predicates. The formulae for the remaining constructs are as usual. The modalities X (neXt) and F (Future) are called *future modalities* whereas the modalities Y (Yesterday) and P (Past) are called *past modalities*. In order to define $u \models \varphi$ without a distinguished position in u, we start evaluation in front (position 0) for future modalities and after (position $|u|+1$) the word u

for past modalities. More formally, for a word $u \in A^*$ we define $u \not\models a$ and

$$u \models \mathsf{X}\,\varphi \text{ if and only if } u \models \varphi(1), \qquad u \models \mathsf{F}\,\varphi \text{ if and only if } u \models \mathsf{F}\,\varphi(1),$$
$$u \models \mathsf{Y}\,\varphi \text{ if and only if } u \models \varphi(|u|), \qquad u \models \mathsf{P}\,\varphi \text{ if and only if } u \models \mathsf{P}\,\varphi(|u|).$$

Boolean connectives and atomic formulae $\top$ and $\bot$ are defined as usual. For example, the formula $\mathsf{X}\,a \wedge \mathsf{Y}\,b$ defines the language aA^*b. Let $\mathrm{TL}_m[\mathsf{X},\mathsf{F},\mathsf{Y},\mathsf{P}]$ be the fragment of unary temporal logic consisting of the Boolean combinations of formulae with at most $m-1$ nested negations. Let $\mathrm{TL}_{m,n}[\mathsf{X},\mathsf{F},\mathsf{Y},\mathsf{P}]$ consist of all formulae in $\mathrm{TL}_m[\mathsf{X},\mathsf{F},\mathsf{Y},\mathsf{P}]$ with operator depth at most n, *i.e.*, there are at most n nested temporal modalities. For a formula φ in first-order logic or in unary temporal logic, let $L(\varphi) = \{u \in A^+ \mid u \models \varphi\}$ be the language defined by φ.

Algebra. Let S be a finite semigroup. An element $x \in S$ is *idempotent* if $x^2 = x$. The set of all idempotents of S is denoted $E(S)$. For every finite semigroup S there exists an integer $\omega \geq 1$ such that each ω-power is idempotent in S. *Green's relations* are an important concept in the structure theory of finite semigroups: For $x, y \in S$ let $x \leq_{\mathcal{R}} y$ if $x = y$ or $x \in yS$ and symmetrically let $x \leq_{\mathcal{L}} y$ if $x = y$ or $x \in Sy$. For $\mathcal{G} \in \{\mathcal{R}, \mathcal{L}\}$ let $x \mathrel{\mathcal{G}} y$ if both $x \leq_{\mathcal{G}} y$ and $y \leq_{\mathcal{G}} x$; and let $x <_{\mathcal{G}} y$ if $x \leq_{\mathcal{G}} y$ but not $y \leq_{\mathcal{G}} x$. We also view S as an alphabet and write $u \in S^+$ for a word with letters from S. For words $u, v \in S^+$ we say that a relation $u \mathrel{\mathcal{G}} v$ "holds in S", if the relation is satisfied after evaluating u and v in S. We use this frequently for equality and Green's relations. All semigroups in this paper are nonempty.

Classes of finite semigroups are often defined by *identities* of omega-terms. An *omega-term* over a set of variables Σ is defined inductively. Every $x \in \Sigma$ is an omega-term, and if u and v are omega-terms, then so are uv and u^ω. A finite semigroup S *satisfies* the identity $u = v$ if for each homomorphism $h : \Sigma^+ \to S$ we have $h(u) = h(v)$. Here, h is extended to omega-terms by letting $h(u^\omega)$ be the idempotent generated by $h(u)$.

For every $e \in E(S)$ the set eSe forms the so-called *local monoid* at e. A semigroup S belongs to $\mathbf{LDA}$ if every local monoid eSe satisfies $(xy)^\omega x (xy)^\omega = (xy)^\omega$. This is equivalent to saying that we have $(exeye)^\omega exe (exeye)^\omega = (exeye)^\omega$ in S for all $x, y \in S$ and all $e \in E(S)$. Note that if S is in $\mathbf{LDA}$ and if $e \in E(S)$ and $x, y \in eSe$ then, $(xy)^\omega = (xy)^{\omega-1} x (yx)^\omega y = (xy)^{\omega-1} x (yx)^\omega y (yx)^\omega y = (xy)^{2\omega} y (xy)^\omega = (xy)^\omega y (xy)^\omega$. Thus despite its asymmetric definition, $\mathbf{LDA}$ is left-right-symmetric.

A homomorphism $h : A^+ \to S$ to a finite semigroup S *recognizes* a language $L \subseteq A^+$ if $h^{-1}(h(L)) = L$. A semigroup S *recognizes* $L \subseteq A^+$ if there exists a homomorphism $h : A^+ \to S$ which recognizes L. For $u, v \in A^+$ let $u \equiv_L v$ if $puq \in L$ is equivalent to $pvq \in L$ for all $p, q \in A^*$. The relation $\equiv_L$ over A^+ is a congruence and the semigroup $A^+/{\equiv_L}$, also denoted by $\mathrm{Synt}(L)$ and called the *syntactic semigroup* of L, is the unique minimal semigroup recognizing L. Moreover, it is effectively computable (*e.g.* from an automaton for L); *cf.* [19].

3 Alternation within Two-Variable First-Order Logic with Successor

We define classes $\mathbf{W}_m$ of finite semigroups which will yield an algebraic characterization of $\mathrm{FO}^2_m[<, \mathrm{suc}]$. To this end, we inductively define sequences of omega-terms U_m, V_m with variables $e, f, x_i, y_i, s, t, p_i, q_i$. For $m = 1$ we define $U_1 = (e^\omega s f^\omega x_1 e^\omega)^\omega s (f^\omega y_1 e^\omega t f^\omega)^\omega$ and $V_1 = (e^\omega s f^\omega x_1 e^\omega)^\omega t (f^\omega y_1 e^\omega t f^\omega)^\omega$ and for $m \geq 2$

$$U_m = (p_m U_{m-1} q_m x_m)^\omega p_m U_{m-1} q_m (y_m p_m U_{m-1} q_m)^\omega,$$
$$V_m = (p_m U_{m-1} q_m x_m)^\omega p_m V_{m-1} q_m (y_m p_m U_{m-1} q_m)^\omega.$$

By definition, a semigroup is in $\mathbf{W}_m$ if it satisfies the identity $U_m = V_m$. The class $\mathbf{W}_1$ is Knast's algebraic characterization of dot-depth one [9]. The only difference between U_1 and V_1 is the central variable in U_1 being s and in V_1 being t. Intuitively, this difference is hidden more and more in U_m and V_m with increasing m.

The following result is the main contribution of this paper. The remainder of this section is dedicated to its proof.

▶ **Theorem 1.** *Let $m \geq 2$ and let $L \subseteq A^+$. The following assertions are equivalent:*

1. *L is definable in $\mathrm{FO}^2_m[<, \mathrm{suc}]$.*
2. *L is definable in $\mathrm{TL}_m[\mathsf{X}, \mathsf{F}, \mathsf{Y}, \mathsf{P}]$.*
3. *$\mathrm{Synt}(L) \in \mathbf{W}_m$.*

Before turning to the proof of Theorem 1 we record the following decidability corollary. For $m = 1$ it relies on a characterization of two-sided ideals inside dot-depth one [11].

▶ **Corollary 2.** *For every positive integer m one can decide whether a given regular language $L \subseteq A^+$ is definable in $\mathrm{FO}^2_m[<, \mathrm{suc}]$.* ◀

We start with the hard part of the proof of Theorem 1, *i.e.*, with the implication from (3) to (1). This is essentially Proposition 13 whose proof requires some preparatory work: We first show that every $\mathbf{W}_m$ is contained in $\mathbf{LDA}$ (Lemma 3) which allows us to use a combinatorial property of $\mathbf{LDA}$ (given in Lemma 6). Then a relativization technique for FO^2_m (Lemma 7) is used for defining a congruence $\approx_{m,n}$ (Definition 8) as a tool for FO^2_m. The connection between this congruence and FO^2_m is established by Lemma 10. Using a string rewriting system, a special factorization (given in Lemma 12) finally leads to an inductive scheme to prove Proposition 13.

In the proof of Theorem 1 at the very end of this section we sketch how to show the reverse implication as well as how to incorporate unary temporal logic.

▶ **Lemma 3.** *For all $m \geq 1$ we have $\mathbf{W}_m \subseteq \mathbf{LDA}$.*

Proof. Let S be a finite semigroup and let $\omega \geq 1$ be an integer such that x^ω is idempotent for all $x \in S$. Let $x, y \in S$ and let $e \in S$ be idempotent. Setting $e_1 = f_1 = s = e$, $x_1 = xey$, $y_1 = x$, $t = y$ we get $U_1 = (exeye)^\omega$ in S and $V_1 = (exeye)^\omega eye(exeye)^\omega$ in S. Setting all other variables occurring in U_m or in V_m to be e, we see $U_m = (exeye)^\omega$ in S and $V_m = (exeye)^\omega eye(exeye)^\omega$ in S. Thus if $S \in \mathbf{W}_m$ and $e \in E(S)$, then eSe satisfies the identity $(xy)^\omega = (xy)^\omega y(xy)^\omega$, *i.e.*, $S \in \mathbf{LDA}$. ◀

The next lemma is an intermediate result for Lemma 5 and Lemma 6 both of which yield important combinatorial properties of semigroups in $\mathbf{LDA}$.

▶ **Lemma 4.** *Let $S \in \mathbf{LDA}$, let $x, y, z \in S$, and let $e \in E(S)$.*

1. *If $xe \mathrel{\mathcal{R}} ye$ in S, then $xe \mathrel{\mathcal{R}} xez$ if and only if $ye \mathrel{\mathcal{R}} yez$.*
2. *If $ex \mathrel{\mathcal{L}} ey$ in S, then $ex \mathrel{\mathcal{L}} zex$ if and only if $ey \mathrel{\mathcal{L}} zey$.*

Proof. Since $\mathbf{LDA}$ is left-right symmetric, it suffices to show (1). Suppose $xe \mathrel{\mathcal{R}} xez$. Since $ye \mathrel{\mathcal{R}} xe \mathrel{\mathcal{R}} xez$ there exist s, t such that $xe = yes$ and $ye = xezt$. We get $ye = ye(esezte)$. Pumping the factor in the parentheses and using $\mathbf{LDA}$ yields $ye = ye(esezte)^\omega = ye(esezte)^\omega ezte(esezte)^\omega \in yezS$. ◀

▶ **Lemma 5.** *Let $S \in \mathbf{LDA}$, let $u, v \in S^+$, let $s, t \in S^*$ with $\mathrm{alph}_{|S|+1}(vs) = \mathrm{alph}_{|S|+1}(vt)$ and $|v| \geq |S|$.*

1. *If $u \mathrel{\mathcal{R}} uv$ in S, then $u \mathrel{\mathcal{R}} uvs$ in S if and only if $u \mathrel{\mathcal{R}} uvt$ in S.*
2. *If $u \mathrel{\mathcal{L}} vu$ in S, then $u \mathrel{\mathcal{L}} svu$ in S if and only if $u \mathrel{\mathcal{L}} tvu$ in S.*

Proof. Since **LDA** is left-right symmetric, it suffices to show (1). Assume $u \mathrel{\mathcal{R}} uv \mathrel{\mathcal{R}} uvs$ in S. We want to show $u \mathrel{\mathcal{R}} uvt$ in S. This is trivial if t is the empty word. Otherwise we factorize $vt = pwz$ such that $|w| < |wz| = |S| + 1$ with $w = we$ in S for some idempotent e of S. Note that every sequence $x_1, \ldots, x_{|S|} \in S$ has a prefix which admits an idempotent stabilizer, *i.e.*, there exists $i \in \{1, \ldots, |S|\}$ and $e \in E(S)$ such that $x_1 \cdots x_i = x_1 \cdots x_i e$ in S; see *e.g.* [11, Lemma 1] for a proof of this claim. Since vs and vt have the same factors of length $|S| + 1$, we find a factorization $vs = s_1 wzs_2$. Let $x = us_1w$ and $y = upw$. By induction $u \mathrel{\mathcal{R}} y$ and thus $xe = x \mathrel{\mathcal{R}} y = ye$ in S. Moreover, $xe \mathrel{\mathcal{R}} xez$ and by Lemma 4 we see $ye \mathrel{\mathcal{R}} yez$ in S. This implies the claim. ◀

Choosing s to be the empty word and $t = a$ immediately yields the following consequence.

▶ **Lemma 6.** *Let $S \in$ **LDA**, let $u, v \in S^+$, let $a \in S$ and let $|v| \geq |S|$.*
1. *If $u \mathrel{\mathcal{R}} uv >_{\mathcal{R}} uva$ in S, then $\mathrm{alph}_{|S|+1}(v) \neq \mathrm{alph}_{|S|+1}(va)$.*
2. *If $u \mathrel{\mathcal{L}} vu >_{\mathcal{L}} avu$ in S, then $\mathrm{alph}_{|S|+1}(v) \neq \mathrm{alph}_{|S|+1}(av)$.* ◀

The next lemma gives the main combinatorial properties of $\mathrm{FO}^2_m[<, \mathrm{suc}]$ for our purpose, namely relativizations of formulae to certain factors of deterministic factorizations.

▶ **Lemma 7.** *Let $\varphi \in \mathrm{FO}^2[<, \mathrm{suc}]$ and let $v, w \in A^+$.*
1. *There exist formulae $\langle\varphi\rangle_{<\mathsf{X}w}$ and $\langle\varphi\rangle_{>\mathsf{X}w}$ such that for all $u = u_1wu_2$ with a unique occurrence of the factor w in the prefix u_1w:*

$$u \models \langle\varphi\rangle_{<\mathsf{X}w}(i,j) \;\; \textit{iff} \;\; u_1 \models \varphi(i,j) \;\; \textit{for all } 1 \leq i,j \leq |u_1|,$$
$$u \models \langle\varphi\rangle_{>\mathsf{X}w}(i,j) \;\; \textit{iff} \;\; u_2 \models \varphi(i - |u_1w|, j - |u_1w|) \;\; \textit{for all } |u_1w| < i,j \leq |u|.$$

2. *There exist formulae $\langle\varphi\rangle_{<\mathsf{Y}v}$ and $\langle\varphi\rangle_{>\mathsf{Y}v}$ such that for all $u = u_1vu_2$ with a unique occurrence of the factor v in the suffix vu_2:*

$$u \models \langle\varphi\rangle_{<\mathsf{Y}v}(i,j) \;\; \textit{iff} \;\; u_1 \models \varphi(i,j) \;\; \textit{for all } 1 \leq i,j \leq |u_1|,$$
$$u \models \langle\varphi\rangle_{>\mathsf{Y}v}(i,j) \;\; \textit{iff} \;\; u_2 \models \varphi(i - |u_1v|, j - |u_1v|) \;\; \textit{for all } |u_1v| < i,j \leq |u|.$$

3. *There exists a formula $\langle\varphi\rangle_{[v;w]}$ such that for all $u = u_1vu_2wu_3$ with a unique occurrence of the factor v in vu_2wu_3 and a unique occurrence of the factor w in u_1vu_2w:*

$$u \models \langle\varphi\rangle_{[v;w]}(i,j) \;\; \textit{iff} \;\; u_2 \models \varphi(i - |u_1v|, j - |u_1v|) \;\; \textit{for all } |u_1v| < i,j \leq |u_1vu_2|.$$

Moreover, if $\varphi \in \mathrm{FO}^2_{m,n}[<, \mathrm{suc}]$, then
1. *$\langle\varphi\rangle_{<\mathsf{X}w} \in \mathrm{FO}^2_{m+1,n+|w|}[<, \mathrm{suc}]$ and $\langle\varphi\rangle_{>\mathsf{X}w} \in \mathrm{FO}^2_{m,n+|w|}[<, \mathrm{suc}]$,*
2. *$\langle\varphi\rangle_{<\mathsf{Y}v} \in \mathrm{FO}^2_{m,n+|v|}[<, \mathrm{suc}]$ and $\langle\varphi\rangle_{>\mathsf{Y}v} \in \mathrm{FO}^2_{m+1,n+|v|}[<, \mathrm{suc}]$, and*
3. *$\langle\varphi\rangle_{[v;w]} \in \mathrm{FO}^2_{m+1,n+N}[<, \mathrm{suc}]$ for $N = \max\{|v|, |w|\}$.* ◀

The relativization of the previous lemma leads to the congruence in the following definition. This congruence is our tool for the combinatorics of FO^2_m in the subsequent proofs.

▶ **Definition 8.** Let $u, v \in A^*$. For $m, n \geq 0$ we let $u \approx_{m,0} v$ and $u \approx_{0,n} v$. For $n \geq 1$ let $u \approx_{1,n} v$ if u and v are contained in the same monomials $w_1A^+w_2 \cdots A^+w_\ell$ with $w_i \in A^+$ and $|w_1 \cdots w_\ell| \leq n$. For $m \geq 2$ and $n \geq 1$ let $u \approx_{m,n} v$ if $\mathrm{alph}_k(u) = \mathrm{alph}_k(v)$ and $\mathrm{pref}_k(u) = \mathrm{pref}_k(v)$ and $\mathrm{suff}_k(u) = \mathrm{suff}_k(v)$ for all $k \leq n$, and all of the following hold:

1. if $u = u_1 w u_2$ and $v = v_1 w v_2$ with $1 \leq |w| \leq n$ such that the factor w has a unique occurrence in the prefixes $u_1 w$ and $v_1 w$, then $u_1 \approx_{m-1,n-|w|} v_1$ and $u_2 \approx_{m,n-|w|} v_2$,
2. if $u = u_1 w u_2$ and $v = v_1 w v_2$ with $1 \leq |w| \leq n$ such that the factor w has a unique occurrence in the suffixes $w u_2$ and $w v_2$, then $u_1 \approx_{m,n-|w|} v_1$ and $u_2 \approx_{m-1,n-|w|} v_2$,
3. if $u = u_1 w u_2 w' u_3$ and $v = v_1 w v_2 w' v_3$ with $|ww'| \leq n$ such that the factor w has a unique occurrence in the suffixes $w u_2 w' u_3$ and $w v_2 w' v_3$ and such that the factor w' has a unique occurrence in the prefixes $u_1 w u_2 w'$ and $v_1 w v_2 w'$, then $u_2 \approx_{m-1,n-|ww'|} v_2$. ◀

An elementary verification shows that $\approx_{m,n}$ is a congruence. Since this fact is not used in this paper, we do not record it as lemma. The following is also straightforward.

▶ **Lemma 9.** *If $m, n \geq 1$ and $u, v \in A^*$ with $u \approx_{m,n} v$, then $u \approx_{m-1,n} v$ and $u \approx_{m,n-1} v$.* ◀

The next lemma connects $\mathrm{FO}^2_{m,n}$ with the combinatorial properties captured by $\approx_{m,n}$. For $u, v \in A^*$ let $u \equiv_{1,n} v$ if u and v model the same formulae in $\mathrm{FO}^2_{1,n}[<, \mathrm{suc}, \min, \max]$. For $m \geq 2$ and $u, v \in A^*$ let $u \equiv_{m,n} v$ if u and v model the same formulae in $\mathrm{FO}^2_{m,n}[<, \mathrm{suc}]$. We have to include min and max predicates at level 1 for technical reasons.

▶ **Lemma 10.** *If $m, n \geq 0$ and $u, v \in A^*$ with $u \equiv_{m,n+1} v$, then $u \approx_{m,n} v$.* ◀

In other words the previous lemma shows that $\equiv_{m,n+1}$ is a refinement of $\approx_{m,n}$. In particular, $\approx_{m,n}$ has finite index. The next lemma is an auxiliary statement used in the proof of Lemma 12. It says that $\approx_{1,n}$ equivalence of u and v allows order comparison for certain factors in the words u and v.

▶ **Lemma 11.** *Let $u, v \in A^+$ and consider factorizations $u = x_1 u_1 \cdots x_k u_k = u'_1 y_1 \cdots u'_\ell y_\ell$ and $v = x_1 v_1 \cdots x_k v_k = v'_1 y_1 \cdots v'_\ell y_\ell$ with $k, \ell \geq 1$ and $u'_1, v'_1, u_k, v_k \in A^*$ and $x_i, y_i \in A^+$ such that*

- *$x_1 u_1 \cdots x_k$ is the shortest prefix of u contained in $x_1 A^+ x_2 \cdots A^+ x_k$ and $x_1 v_1 \cdots x_k$ is the shortest prefix of v contained in $x_1 A^+ x_2 \cdots A^+ x_k$,*
- *$y_1 \cdots u'_\ell y_\ell$ is the shortest suffix of u contained in $y_1 A^+ y_2 \cdots A^+ y_\ell$ and $y_1 \cdots v'_\ell y_\ell$ is the shortest suffix of v contained in $y_1 A^+ y_2 \cdots A^+ y_\ell$.*

Let $\Delta_u = |u| - |x_1 u_1 \cdots u_{k-1}| - |u'_2 \cdots u'_\ell y_\ell|$ and let $\Delta_v = |v| - |x_1 v_1 \cdots v_{k-1}| - |v'_2 \cdots v'_\ell y_\ell|$. If $u \approx_{1,n} v$ for $n = |x_1 \cdots x_k| + |y_1 \cdots y_\ell|$, then the relative order of the occurrences of x_k and y_1 is the same in u and v, i.e., one of the following conditions applies:

1. *$\Delta_u > |x_k y_1|$ and $\Delta_v > |x_k y_1|$.*
2. *$\Delta_u < 0$ and $\Delta_v < 0$.*
3. *$\Delta_u = \Delta_v$.* ◀

The main combinatorial ingredient for the implication from $\mathbf{W}_m$ to FO^2_m is the factorization in the following lemma. It combines properties of **LDA** and $\approx_{m,n}$.

▶ **Lemma 12.** *Let $S \in \mathbf{LDA}$, let $m \geq 2$, let $N = 2|S|^2$ and let $u, v \in S^+$ such that $u \approx_{m,n+N} v$. Then there exist factorizations $u = w_0 s_1 w_1 \cdots s_\ell w_\ell$ and $v = w_0 t_1 w_1 \cdots t_\ell w_\ell$ with $w_i, s_i, t_i \in S^+$ and $|w_0 \cdots w_\ell| \leq N$ such that for all $1 \leq i \leq \ell$ the following hold:*

1. *$s_i \approx_{m-1,n} t_i$,*
2. *$w_0 s_1 \cdots w_{i-1} \mathrel{\mathcal{R}} w_0 s_1 \cdots w_{i-1} s_i$ in S,*
3. *$w_i \cdots t_\ell w_\ell \mathrel{\mathcal{L}} t_i w_i \cdots t_\ell w_\ell$ in S.*

Proof. Let $X' = \{1\} \cup \{i \in \mathrm{pos}(u) \mid 1 < i \leq |u|\,,\ u[1; i-1] >_{\mathcal{R}} u[1; i] \text{ in } S\}$ be the set of positions of u which cause an $\mathcal{R}$-descent when reading u from left to right. Let X be the set of positions j such that there exists $i \in X'$ with $0 \leq i - j \leq |S|$, *i.e.*, we include all $|S|$ positions to the left of each $i \in X'$. Let Y' and Y be defined left-right symmetrically on v, *i.e.*, $Y' = \{|v|\} \cup \{i \in \operatorname{pos}(v) \mid 1 < i \leq |v|\,,\, v[i-1; |v|] >_{\mathcal{L}} u[i; |v|] \text{ in } S\}$ and Y is the set of positions j such that $0 \leq j - i \leq |S|$ for some $i \in Y'$. Let $X = X_1 \cup \cdots \cup X_k$ with $X_i \neq \emptyset$ being maximal subsets of consecutive positions of X such that all positions of X_i are smaller than all positions of X_{i+1}. Symmetrically, let $Y = Y_1 \cup \cdots \cup Y_{k'}$ with $Y_i \neq \emptyset$ being maximal subsets of consecutive positions of Y such that all positions of Y_i are smaller than all positions of Y_{i+1}.

Let $x_i = u[X_i]$ and $y_i = u[Y_i]$ be the factors of u and v covered by the positions of X_i and Y_i, respectively. By construction and Lemma 6 (1), we see that $u[1; \max(X_i)]$ is the shortest prefix of u which is contained in $x_1 S^+ x_2 \cdots S^+ x_i$. Symmetrically, $v[\min(Y_i); |v|]$ is the shortest suffix of v which is contained in $y_i S^+ y_{i+1} \cdots S^+ y_{k'}$ by Lemma 6 (2). We use these properties to transfer the positions of X to v and the positions of Y to u. Specifically we let $Y'' = Y''_1 \cup \cdots \cup Y''_{k'}$ be such that each Y''_i is an interval of positions of u with $u[Y''_i] = y_i$ and $u[\min(Y''_i); |u|]$ is the shortest suffix of u which is contained in $y_i S^+ y_{i+1} \cdots S^+ y_{k'}$. And we let $X = X''_1 \cup \cdots \cup X''_k$ be such that each X''_i is an interval of positions of v with $v[X''_i] = x_i$ and $v[1; \max(X''_i)]$ is the shortest prefix of v which is contained in $x_1 S^+ x_2 \cdots S^+ x_i$. Note that $u \in S^* y_1 S^+ y_2 \cdots S^+ y_{k'}$ and $v \in x_1 S^+ x_2 \cdots S^+ x_k S^*$ because $u \approx_{m,n+N} v$.

Now, consider the factorization $u = w_0 s_1 w_1 \cdots s_\ell w_\ell$ with $s_i \in S^+$ such that the w_i are the factors covered by maximal subsets of consecutive positions in $X \cup Y''$. Intuitively, this means that we merge overlapping and adjacent factors x_i and y_j in u. Lemma 11 shows that the relative order of those concrete occurrences of x_i and y_j is the same in v as in u. Therefore, if we consider the factorization of v which is covered by maximal subsets of consecutive positions in $X'' \cup Y$, then we end up with the same factors in the same order, *i.e.*, we have $v = w_0 t_1 w_1 \cdots t_\ell w_\ell$ for some $t_i \in S^+$. Since the $\mathcal{R}$-class and the $\mathcal{L}$-class can descend at most $|S| - 1$ times, we have $|X' \cup Y'| \leq 2\,|S|$ and thus $|w_0 \cdots w_\ell| \leq |X \cup Y''| \leq 2\,|S|^2$. Moreover, by construction every $\mathcal{R}$-descent when reading prefixes of u as well as every $\mathcal{L}$-descent when reading suffixes of v is covered by some factor w_i showing (2) and (3).

It remains to show $s_i \approx_{m-1,n} t_i$ for all i. An intermediate step is the following claim.

Claim. *If $s_k w_k \cdots s_\ell w_\ell \approx_{m,n+N} t_k w_k \cdots t_\ell w_\ell$ for some $N \geq |w_k \cdots w_\ell|$, then $s_i \approx_{m-1,n} t_i$ for all $i \in \{k, \ldots, \ell\}$.*

The proof of this claim is by induction on $\ell - k$. Every w_i either arises from some x_j or some y_j or both. Therefore, the w_i's inherit the properties of the corresponding x_j's and y_j's of being the first occurrence (respectively being the last occurrence). If there is no w_i arising from an x_j, then every w_i has a unique occurrence in $w_i s_{i+1}$ as well as in $w_i t_{i+1}$. Thus $s_i \approx_{m-1,n} t_i$ for all i by an $(\ell - k)$-fold application of condition (2) in the definition of $\approx_{m,n}$ (from right to left). For $i = k$ this uses Lemma 9.

Fix the first w_i which arises from an x_j. We have $s_j \approx_{m-1,n} t_j$ for all $j > i$ by condition (1) in the definition of $\approx_{m,n}$ and induction. If $i = k$, then $s_k \approx_{m-1,n} t_k$ again by condition (1) in the definition of $\approx_{m,n}$. Assume therefore $i > k$ in the sequel. Let $h \geq i$ be minimal such that w_h arises from some y_j; note that w_ℓ arises from $y_{k'}$. By a repeated application of condition (2) in the definition of $\approx_{m,n}$ we get that $s_k w_k \cdots s_h \approx_{m,n+N'} t_k w_k \cdots t_h$ for $N' = |w_k \cdots w_{h-1}|$. Now w_{i-1} has a unique occurrence in each of the words $w_{i-1} s_i \cdots s_h$ and $w_{i-1} t_i \cdots t_h$. Therefore, by repeatedly applying condition (2) in the definition of $\approx_{m,n}$ we see that $s_j \approx_{m-1,n} t_j$ for all $k \leq j < i$. If $h > i$, then by condition (3) in the definition of $\approx_{m,n}$ we see that $s_i \approx_{m-1,n} t_i$; and if $h = i$, then this follows from condition (2) in the definition of $\approx_{m,n}$. This concludes the proof of the claim.

Now by condition (1) in the definition of $\approx_{m,n}$, we see $s_1 w_1 \cdots s_\ell w_\ell \approx_{m,n+N'} t_1 w_1 \cdots t_\ell w_\ell$

for $N' = N - |w_0|$ and the above claim yields $s_j \approx_{m-1,n} t_j$ for all $1 \leq j \leq \ell$. ◀

The following proposition essentially shows how to pass from $\mathbf{W}_m$ to $\mathrm{FO}^2_m[<, \mathrm{suc}]$. The key to its proof is a string rewriting system which enables induction on the parameter m. Intuitively we consider the maximal quotient of a semigroup in $\mathbf{W}_m$ contained in $\mathbf{W}_{m-1}$. Since the latter is given by an omega-identity, this quotient can be described by a string rewriting system. A single rewriting step of this system corresponds to one application of the omega-identity for $\mathbf{W}_{m-1}$ and can be lifted to $\mathbf{W}_m$ relatively easily.

▶ **Proposition 13.** *For every $S \in \mathbf{W}_m$ with $m \geq 1$ there exists $n \geq 1$ such that $u \approx_{m,n} v$ implies $u = v$ in S for all $u, v \in S^+$.*

Proof. We perform an induction on m. By Knast's Theorem [9], if L is recognized by a semigroup $S \in \mathbf{W}_1$, then the language L is a Boolean combination of monomials $w_1 A^+ w_2 \cdots A^+ w_\ell$. Choosing $n \geq 1$ such that for all these monomials we have $|w_1 \cdots w_\ell| \leq n$ yields the claim for $m = 1$.

Let $\omega > |S|$ be an integer such x^ω is idempotent in S for all $x \in S$. Consider the relation $\to$ on S^+ given by $s \to t$ if $s = t$ in S or if $s = pu_{m-1}q$ and $t = pv_{m-1}q$ for some $p, q \in S^*$ and some $x_i, e, y_i, f, p_i, q_i, z, z' \in S^+$ such that $u_1 = (e^\omega z f^\omega x_1 e^\omega)^\omega z (f^\omega y_1 e^\omega z' f^\omega)^\omega$ and $v_1 = (e^\omega z f^\omega x_1 e^\omega)^\omega z' (f^\omega y_1 e^\omega z' f^\omega)^\omega$ and for $i \geq 2$ we have

$$u_i = (p_i u_{i-1} q_i x_i)^\omega p_i u_{i-1} q_i (y_i p_i u_{i-1} q_i)^\omega, \quad v_i = (p_i u_{i-1} q_i x_i)^\omega p_i v_{i-1} q_i (y_i p_i u_{i-1} q_i)^\omega.$$

Let $\overset{*}{\leftrightarrow}$ be the reflexive, symmetric and transitive closure of $\to$. The relation $\overset{*}{\leftrightarrow}$ is a congruence of finite index (since $S^+/\overset{*}{\leftrightarrow}$ is a quotient of S). Moreover $x^\omega \overset{*}{\leftrightarrow} x^{2\omega}$ for all $x \in S^+$ and $S^+/\overset{*}{\leftrightarrow} \in \mathbf{W}_{m-1}$.

Claim 1. *Let $u, s, t \in S^+$. If $s \to t$, then $u \mathrel{\mathcal{R}} us$ in S if and only if $u \mathrel{\mathcal{R}} ut$ in S.*

Assume without restriction that $s \neq t$ in S. We have $\mathrm{alph}_{|S|+1}(s) = \mathrm{alph}_{|S|+1}(t)$ by construction of u_{m-1} and v_{m-1}. Note that by choice of ω, in particular both words have the same prefix and the same suffix of length $|S| + 1$. Lemma 5 yields Claim 1.

Claim 2. *Let $u, v, s, t \in S^+$ with $s \overset{*}{\leftrightarrow} t$. If $u \mathrel{\mathcal{R}} us$ and $v \mathrel{\mathcal{L}} tv$ in S, then $usv = utv$ in S.*

Since $s \overset{*}{\leftrightarrow} t$, there exists $k \geq 0$ and $w_0, \ldots, w_k \in S^+$ such that $s = w_0$ and $w_k = t$ and such that either $w_{i-1} \to w_i$ or $w_i \to w_{i-1}$ for each $1 \leq i \leq k$. Claim 1 and its left-right dual, yield that $u \mathrel{\mathcal{R}} uw_i$ and $v \mathrel{\mathcal{L}} w_i v$ in S for all i. It therefore suffices to show the claim for $s \to t$. The claim is trivial if $s = t$ in S. Otherwise suppose $s = p_m u_{m-1} q_m$ and $t = p_m v_{m-1} q_m$. Since $u \mathrel{\mathcal{R}} us$ in S, there exists $x_m \in S$ such that $u = usx_m$ in S. Since $v \mathrel{\mathcal{L}} tv$ in S, the left-right dual of Claim 1 implies $v \mathrel{\mathcal{L}} sv$ in S. Hence, there exists $y_m \in S$ such that $v = y_m sv$ in S. Now $u = u(p_m u_{m-1} q_m x_m)^\omega$ in S and $v = (y_m p_m u_{m-1} q_m)^\omega v$ in S and with $S \in \mathbf{W}_m$ we see

$$\begin{aligned} usv &= u(p_m u_{m-1} q_m x_m)^\omega p_m u_{m-1} q_m (y_m p_m u_{m-1} q_m)^\omega v \\ &= u(p_m u_{m-1} q_m x_m)^\omega p_m v_{m-1} q_m (y_m p_m u_{m-1} q_m)^\omega v = utv \text{ in } S, \end{aligned}$$

thus establishing Claim 2.

Since $S^+/\overset{*}{\leftrightarrow} \in \mathbf{W}_{m-1}$, by induction there exists $n \geq 1$ such that $s \approx_{m-1,n} t$ implies $s \overset{*}{\leftrightarrow} t$ for all $s, t \in S^+$. Let $u, v \in S^+$ and suppose $u \approx_{m,n+N} v$ for $N = 2|S|^2$. Let $u = w_0 s_1 w_1 \cdots s_\ell w_\ell$ and $v = w_0 t_1 w_1 \cdots t_\ell w_\ell$ be the factorizations given by Lemma 12; in particular $s_i \approx_{m-1,n} t_i$ and $w_0 s_1 \cdots w_{i-1} \mathrel{\mathcal{R}} w_0 s_1 \cdots w_{i-1} s_i$ in S and $w_i \cdots t_\ell w_\ell \mathrel{\mathcal{L}} t_i w_i \cdots t_\ell w_\ell$. By choice of n we have $s_i \overset{*}{\leftrightarrow} t_i$ for all i and repeated application of Claim 2 yields the following chain of identities valid in S:

$$\begin{aligned} v &= w_0 t_1 w_1 t_2 \cdots t_{\ell-1} w_{\ell-1} t_\ell w_\ell \\ &= w_0 s_1 w_1 t_2 \cdots t_{\ell-1} w_{\ell-1} t_\ell w_\ell \\ &= w_0 s_1 w_1 s_2 \cdots t_{\ell-1} w_{\ell-1} t_\ell w_\ell \\ &\ \vdots \\ &= w_0 s_1 w_1 s_2 \cdots s_{\ell-1} w_{\ell-1} t_\ell w_\ell \\ &= w_0 s_1 w_1 s_2 \cdots s_{\ell-1} w_{\ell-1} s_\ell w_\ell = u. \end{aligned}$$

This concludes the proof. ◀

We are now ready to prove Theorem 1.

Proof of Theorem 1. We shall first show "(3) ⇒ (1)". Afterwards we sketch the proof for the implications "(1) ⇒ (3)" and "(1) ⇒ (2)"; note that the implication "(2) ⇒ (1)" is trivial because the semantics of temporal logic formulae is given by two-variable first-order formulae with quantifier alternations originating in negations. We refer to the technical report [14] for full proofs.

"(3) ⇒ (1)": Suppose $S \in \mathbf{W}_m$ and the homomorphism $h : A^+ \to S$ recognizes $L \subseteq A^+$. Combining Proposition 13 and Lemma 10, we see that there exists an integer $n \geq 1$ such that $u \equiv_{m,n} v$ for $u, v \in S^+$ implies $u = v$ in S. Now if $u \equiv_{m,n} v$ for $u, v \in A^+$, then $h(u) = h(v)$. Thus, by specifying the $\equiv_{m,n}$-classes of A^+ which are contained in L, we obtain a formula $\varphi \in \mathrm{FO}^2_{m,n}[<, \mathrm{suc}]$ such that $L(\varphi) = h^{-1}(h(L)) = L$. Note that the syntactic semigroup of L recognizes L.

Sketch of "(1) ⇒ (3)": The overall proof scheme is reminiscent of a recent proof of Straubing [27] which shows that $\mathrm{FO}^2_m[<]$-definable languages are recognized by a monoids in the so-called weakly iterated two-sided semidirect product $((\mathbf{J} ** \mathbf{J}) ** \mathbf{J}) \cdots ** \mathbf{J}$ where $\mathbf{J}$ appears n times. To avoid technical notation our formulation is not in terms of semidirect products, however. More concretely we show that formulae in FO^2_m up to a certain quantifier depth are unable to disprove the defining identity of $\mathbf{W}_m$; this yields a recognizing semigroup of L and thus the claim since the syntactic semigroup is a divisor of any semigroup recognizing L. To this end, an extended alphabet is used to annotate every position by certain information about sequences of factors occurring in the prefix ending and the suffix starting at this position. This allows to reduce the alternation depth of formulae by replacing so-called *innermost quantified blocks* by alphabet information. Induction then yields the claim. The most technical part of this step is to enable induction by showing that certain central factors of the annotated identities for $\mathbf{W}_m$ are obtained from the identities for $\mathbf{W}_{m-1}$ over the extended alphabet.

Sketch of "(1) ⇒ (2)": This can be seen using the construction in [7, proof of Theorem 1] by means of which Etessami, Vardi, and Wilke showed that FO^2 coincides with $\mathrm{TL}[\mathsf{X}, \mathsf{F}, \mathsf{Y}, \mathsf{P}]$; their statement does not involve the alternation depth explicitly, though. Roughly speaking, for every formula in FO^2_m with one free variable an equivalent formula in $\mathrm{TL}_m[\mathsf{X}, \mathsf{F}, \mathsf{Y}, \mathsf{P}]$ is constructed. The idea is to split up quantifier with respect to the order type. For example, the quantifier $\exists x\, \varphi$ is equivalent to the disjunction

$$(\exists x < y - 1 \colon \varphi) \vee (\exists x = y - 1 \colon \varphi) \vee (\exists x = y \colon \varphi) \vee (\exists x = y + 1 \colon \varphi) \vee (\exists x > y + 1 \colon \varphi).$$

In addition, we make explicit the label of the variable x and use syntactic bookkeeping to keep track of the label and the order type. Under the condition that these information be correct, induction yields temporal logic formula for the subformula φ. Now this presupposition can be ensured using the modalities X, F, Y, and P. For example, the subformula in the first parentheses would be YYP φ' where φ' is the formula for φ with respect to the label and the order type $x < y - 1$ which is obtained by induction. ◀

Conclusion

We showed that quantifier alternation for the logic $\mathrm{FO}^2[<, \mathrm{suc}]$ is decidable by giving a single identity of omega-terms for each level $\mathrm{FO}^2_m[<, \mathrm{suc}]$. The key ingredient in our proof is a rewriting technique which allows us to apply induction on m.

There is an algebraic construction $\mathbf{V} \mapsto \mathbf{V} * \mathbf{D}$ in terms of wreath products, see *e.g.* [26]. For most logical fragments $\mathcal{F}$, whenever $\mathcal{F}$ corresponds to a variety of finite monoids $\mathbf{V}$, then the fragment $\mathcal{F}'$ obtained from $\mathcal{F}$ by adding successor predicates corresponds to the semigroup variety $\mathbf{V} * \mathbf{D}$. This is also the case for $\mathrm{FO}^2_m[<]$ and $\mathrm{FO}^2_m[<, \mathrm{suc}]$. Therefore, if $\mathbf{V}_m$ is the variety of finite monoids corresponding to $\mathrm{FO}^2_m[<]$, then our result implies $\mathbf{V}_m * \mathbf{D} = \mathbf{W}_m$.

In general, decidability of $\mathbf{V}$ is not preserved by the operation $\mathbf{V} \mapsto \mathbf{V} * \mathbf{D}$, but a particularly nice situation occurs if $\mathbf{V} * \mathbf{D} = \mathbf{LV}$. Here, a semigroup S is in $\mathbf{LV}$ if all local monoids of S are in $\mathbf{V}$. For example the variety $\mathbf{DA}$ satisfies $\mathbf{DA} * \mathbf{D} = \mathbf{LDA}$, see [1, 4]. For $\mathbf{W}_1$ however, Knast has given an example showing $\mathbf{V}_1 * \mathbf{D} \neq \mathbf{LV}_1$. In view of this example, we conjecture that $\mathbf{V}_m * \mathbf{D} \neq \mathbf{LV}_m$ for all $m \geq 1$.

Acknowledgments. We thank the anonymous referees for their suggestions which helped to improve the presentation of the paper.

References

1 J. Almeida. A syntactical proof of locality of **DA**. *Int. J. Algebra Comput.*, 6(2):165–177, 1996.

2 J. R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundlagen Math.*, 6:66–92, 1960.

3 R. S. Cohen and J. A. Brzozowski. Dot-depth of star-free events. *J. Comput. Syst. Sci.*, 5(1):1–16, 1971.

4 A. Costa and A. P. Escada. Some operators that preserve the locality of a pseudovariety of semigroups. Technical Report 11–37 DMUC, University of Coimbra, 2011.

5 V. Diekert, P. Gastin, and M. Kufleitner. A survey on small fragments of first-order logic over finite words. *Int. J. Found. Comput. Sci.*, 19(3):513–548, 2008. Special issue DLT 2007.

6 C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–51, 1961.

7 K. Etessami, M. Y. Vardi, and Th. Wilke. First-order logic with two variables and unary temporal logic. *Inf. Comput.*, 179(2):279–295, 2002.

8 J. A. W. Kamp. *Tense Logic and the Theory of Linear Order.* PhD thesis, University of California, 1968.

9 R. Knast. A semigroup characterization of dot-depth one languages. *RAIRO, Inf. Théor.*, 17(4):321–330, 1983.

10 A. Krebs and H. Straubing. An effective characterization of the alternation hierarchy in two-variable logic. In *FSTTCS 2012, Proceedings*, volume 18 of *LIPIcs*, pages 86–98. Dagstuhl Publishing, 2012.

11 M. Kufleitner and A. Lauser. Around dot-depth one. *Int. J. Found. Comput. Sci.*, 23(6):1323–1339, 2012.

12 M. Kufleitner and A. Lauser. The join levels of the Trotter-Weil hierarchy are decidable. In *MFCS 2012, Proceedings*, volume 7464 of *LNCS*, pages 603–614. Springer, 2012.

13 M. Kufleitner and A. Lauser. Lattices of logical fragments over words. In *ICALP 2012, Proceedings Part II*, volume 7392 of *LNCS*, pages 275–286. Springer, 2012.

14 M. Kufleitner and A. Lauser. Quantifier alternation in two-variable first-order logic with successor is decidable. *CoRR*, arXiv:1212.6500 [cs.LO], 2012.

15 M. Kufleitner and P. Weil. On the lattice of sub-pseudovarieties of **DA**. *Semigroup Forum*, 81:243–254, 2010.

16 M. Kufleitner and P. Weil. The FO^2 alternation hierarchy is decidable. In *CSL 2012, Proceedings*, volume 16 of *LIPIcs*, pages 426–439. Dagstuhl Publishing, 2012.

17 M. Kufleitner and P. Weil. On logical hierarchies within FO^2-definable languages. *Log. Methods Comput. Sci.*, 8:1–30, 2012.

18 R. McNaughton and S. Papert. *Counter-Free Automata*. The MIT Press, 1971.

19 J.-É. Pin. *Varieties of Formal Languages*. North Oxford Academic, 1986.

20 J.-É. Pin and P. Weil. Polynomial closure and unambiguous product. *Theory Comput. Syst.*, 30(4):383–422, 1997.

21 M. P. Schützenberger. On finite monoids having only trivial subgroups. *Inf. Control*, 8:190–194, 1965.

22 M. P. Schützenberger. Sur le produit de concaténation non ambigu. *Semigroup Forum*, 13:47–75, 1976.

23 Th. Schwentick, D. Thérien, and H. Vollmer. Partially-ordered two-way automata: A new characterization of DA. In *DLT 2001, Proceedings*, volume 2295 of *LNCS*, pages 239–250. Springer, 2002.

24 I. Simon. Piecewise testable events. In *Autom. Theor. Form. Lang., 2nd GI Conf.*, volume 33 of *LNCS*, pages 214–222. Springer, 1975.

25 H. Straubing. A generalization of the Schützenberger product of finite monoids. *Theor. Comput. Sci.*, 13:137–150, 1981.

26 H. Straubing. Finite semigroup varieties of the form $\mathbf{V} * \mathbf{D}$. *J. Pure Appl. Algebra*, 36(1):53–94, 1985.

27 H. Straubing. Algebraic characterization of the alternation hierarchy in FO$^2[<]$ on finite words. In *CSL 2011, Proceedings*, volume 12 of *LIPIcs*, pages 525–537. Dagstuhl Publishing, 2011.

28 P. Tesson and D. Thérien. Diamonds are forever: The variety DA. In *Semigroups, Algorithms, Automata and Languages 2001, Proceedings*, pages 475–500. World Scientific, 2002.

29 D. Thérien. Classification of finite monoids: The language approach. *Theor. Comput. Sci.*, 14(2):195–208, 1981.

30 D. Thérien and Th. Wilke. Over words, two variables are as powerful as one quantifier alternation. In *STOC 1998, Proceedings*, pages 234–240. ACM Press, 1998.

31 W. Thomas. Classifying regular events in symbolic logic. *J. Comput. Syst. Sci.*, 25:360–376, 1982.

32 W. Thomas. Languages, automata and logic. In *Handbook of Formal Languages*, volume 3, pages 389–455. Springer, 1997.

33 B. A. Trakhtenbrot. Finite automata and logic of monadic predicates (in Russian). *Dokl. Akad. Nauk SSSR*, 140:326–329, 1961.

34 P. Trotter and P. Weil. The lattice of pseudovarieties of idempotent semigroups and a non-regular analogue. *Algebra Univers.*, 37(4):491–526, 1997.

35 Ph. Weis and N. Immerman. Structure theorem and strict alternation hierarchy for FO^2 on words. *Log. Methods Comput. Sci.*, 5:1–23, 2009.

FO^2 with one transitive relation is decidable

Wiesław Szwast[1] and Lidia Tendera*[1]

1 **Institute of Mathematics and Informatics,**
Opole University, Oleska 48, 45-052 Opole, Poland
`[szwast,tendera]@math.uni.opole.pl`

Abstract

We show that the satisfiability problem for the two-variable first-order logic, FO^2, over transitive structures when only one relation is required to be transitive, is decidable. The result is optimal, as FO^2 over structures with two transitive relations, or with one transitive and one equivalence relation, are known to be undecidable, so in fact, our result completes the classification of FO^2-logics over transitive structures with respect to decidability. We show that the satisfiability problem is in 2-NEXPTIME. Decidability of the finite satisfiability problem remains open.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.1 Mathematical Logic, F.4.3 Formal Languages

Keywords and phrases classical decision problem, two-variable first-order logic, decidability, computational complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.317

1 Introduction

FO^2 is the restriction of the classical first-order logic over relational signatures to formulae with at most two distinct variables. It is well-known that FO^2 enjoys the finite model property [20], and its satisfiability (hence also finite satisfiability) problem is NEXPTIME-complete [5].

One particular drawback of FO^2 is that it can neither express transitivity of a binary relation nor say that a binary relation is a partial (or linear) order, or an equivalence relation. These natural properties are important for practical applications, thus research has started to investigate FO^2 over restricted classes of structures in which some distinguished binary symbols are required to be interpreted as transitive relations, orders, equivalences, etc. The idea comes from modal correspondence theory, where various conditions on the accessibility relations allow to restrict the class of Kripke structures considered, e.g. to transitive structures for the modal logic K4 or equivalence structures for the modal logic S5. Orderings, on the other hand, are very natural when considering temporal logics, where they model time flow, but they also are used in different scenarios, e.g. in databases or description logics, to compare objects with respect to some parameters.

Unfortunately, the remarkably robust decidability of modal logics and its various extensions towards greater expressibility does not transfer immediately to extensions of FO^2, and the picture for FO^2 is more complex and to some extent less understood. It appeared that both the satisfiability and the finite satisfiability problems for FO^2 are undecidable in the presence of several equivalence or several transitive relations [6, 7]. These results were later

* Lidia Tendera would like to acknowledge the support of Polish Ministry of Science and Higher Education grant N N206 37133.

strengthened: FO^2 is undecidable in the presence of two transitive relations [11, 9], three equivalence relations [15], one transitive and one equivalence relation [17], or three linear orders [12].

On the positive side it is known that FO^2 with one or two equivalence relations is decidable [16, 17, 14]. The same holds for FO^2 with one linear order [22]. The intriguing questions left open by this research was the case of FO^2 with one transitive relation and FO^2 with two linear orders.

In this paper we answer the first question positively: we prove that the satisfiability problem for the extension of FO^2 where exactly one binary relation is required to be transitive, FO^2_T, is decidable in 2-NExpTime. The result completes the classification of variants of FO^2 over transitive structures with respect to decidability.

For the special case of two linear orders, ExpSpace-completeness of finite satisfiability is shown, subject to certain restrictions on signatures, in [24]. (The case of unrestricted signatures, and decidability of the general satisfiability problem are currently open.)

It is also worth to compare the above results with results concerning GF^2 i.e. the two-variable restriction of the *guarded fragment* GF [1] where quantifiers are guarded by atoms. GF+TG is the restriction of GF^2 with transitive relations where the transitive relation symbols are allowed to appear only in guards. As shown in [26] undecidability of FO^2 with transitivity transfers to GF^2 with transitivity; however, GF+TG is decidable irrespective of the number of transitive symbols. Moreover, as noted in [11], the decision procedure developed for GF^2+TG can be applied to GF^2 with one transitive relation that is allowed to appear also outside guards, giving 2-ExpTime-completeness of the latter fragment.

Also of note in this context is the interpretation of FO^2 over *data words* and *data trees* that appear e.g. in verification and XML processing. Decidability of FO^2 over data words with one additional equivalence relation was shown in [3]. For more results related to FO^2 over data words or data trees see e.g. [18, 24, 4, 21, 2].

It makes sense to also consider more expressive systems in which we may refer to the transitive closure of some relation. In fact, relatively few decidable fragments of first-order logic with transitive closure are known. One exception is the logic GF^2 with a transitive closure operator applied to binary symbols appearing only in guards [19]. This fragment captures the two-variable guarded fragment with transitive guards, GF^2+TG, preserving its complexity [27, 10]. Also decidable is the satisfiability problem for the logic $\exists\forall(\mathrm{DTC}^+[E])$, i.e. the prefix class $\exists\forall$ extended by the positive deterministic transitive closure operator of one binary relation, which is shown to enjoy the exponential model property [8]. Recently, it has been shown that the satisfiability problem for the two-variable universal fragment of first-order logic with constants remains decidable when extended by the transitive closure of a single binary relation [13]. Whether the same holds for full FO^2 is open.

Expressive power of FO^2_T. As has already been mentioned, FO^2 has the finite model property. Adding one transitive relation to GF^2 (even restricted only to guards) we can write infinity axioms, however models for this logic still enjoy the so called tree-like property, i.e. new elements required by $\forall\exists$-conjuncts can be added independently. Below we give an example of an infinity axiom in FO^2_T that enforces models where in some triples all elements depend on each other.

We use the transitive relation symbol T and two unary symbols P and Q. It is not difficult to formalize the following statements by an FO^2_T formula: (a) T is strictly antisymmetric. (b) Elements of P form one infinite chain. (c) Elements of Q are incomparable. (d) Every element of P has an incomparable element in Q. (e) Every element of Q is smaller than some element in P.

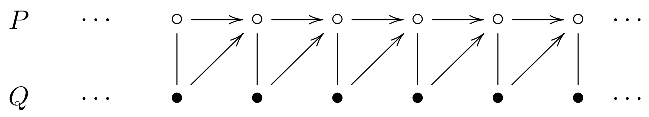


Figure 1 A model satisfying (a)-(e). Arrows depict elements related by T. Lines connect elements required by (d), not connected by T.

In any model satisfying (a)-(e) there is an infinite chain of elements in P that induces an infinite antichain of elements in Q (see Figure 1). Note also, that it suffices that the unique transitive relation is supposed to be a partial ordering.

Outline of the proof. Models for our logic, taking into account the interpretation of the transitive relation, can obviously be seen as partitioned into cliques. As usually for two-variable logics, we first establish a “Scott-type” normal form for FO^2_T: $\forall\forall \;\; \wedge \bigwedge \forall\exists$, allowing us to restrict the nesting of quantifiers to depth two, as well as to concentrate on the $\forall\exists$-conjuncts demanding “witnesses” for all elements in a model. The form of the $\forall\exists$-conjuncts enables to distinguish witnesses required inside cliques (i.e. realizing a 2-type containing both $T(x, y)$ and $T(y, x)$, see Section 2 for a precise definition) from witnesses outside cliques. We also establish a *small clique property* for FO^2_T (practically the same as in [15]), allowing us to restrict attention to models with cliques exponentially bounded in the size of the signature. Further constructions proceed on levels of cliques rather than individual elements. (An alternative approach would be to consider first the satisfiability problem over an antisymmetric relation T and then reduce the general problem to the aforementioned one taking into account the bound on the clique sizes.)

Crucial to our argument is this property: any infinitely satisfiable sentence has an infinite *narrow* model, i.e. a model whose universe can be partitioned into segments (i.e. sets of cliques) $S_0, S_1, \ldots$, each of doubly exponential size, such that every element in $\bigcup_{i=0}^{j-1} S_i$ requiring a witness outside its clique has the witness either in S_0 or in S_j (so, in every S_k, $k \geq j$, Def. 16). This immediately implies that, when needed, every single segment S_j ($j > 0$) can be removed from the structure, to yield a model with new properties.

To prove existence of narrow models, we first make some useful observations. In particular, we show that a single clique can be duplicated, provided its type called *splice* appears at least twice in a model (Claim 7). The property is used to show the main technical result (Claim 10 and Corollary 11). Next, the idea is generalized in Lemma 13 to show that for any finite subset F of elements, the model can be extended by a fixed number of cliques (depending only on the signature, and not depending on the cardinality of F) providing all required witnesses for elements from F.

As the main result of the paper, we show that from any narrow model we can build a *canonical model* where every two segments of the infinite partition (except the first) are isomorphic and they are connected using at most two distinct similarity types (Def. 19). In fact, these constructions can be seen as an application of the infinite Ramsey theorem [23], where segments of the models are considered to be nodes in a colored graph, and similarity types of pairs of segments are colors of edges.

The above properties suffice to obtain the 2-NExpTime decision procedure for the satisfiability problem for FO^2_T given in Theorem 21 and Corollary 22. We note however that the best lower bound coming from GF^2+TG is 2-ExpTime, thus our result leaves a gap in complexity. We also note that our decision procedure cannot be straightforwardly generalized to solve the finite satisfiability problem for FO^2_T and to the best of our knowledge, the latter problem remains open (see Outlook for some discussion).

2 Preliminaries

We denote by FO^2 the two-variable fragment of first-order logic (with equality) over relational signatures. By FO^2_T we understand the set of FO^2_T-formulas over any signature $\sigma = \sigma_0 \cup \{T\}$, where T is a distinguished binary predicate. The semantics for FO^2_T is as for FO^2, subject to the restriction that T is always interpreted as a *transitive* relation.

In this paper, σ-structures are denoted by Gothic capital letters and their universes by corresponding Latin capitals. Where a structure is clear from context, we frequently equivocate between predicates and their realizations, thus writing, for example, R in place of the technically correct $R^{\mathfrak{A}}$. If $\mathfrak{A}$ is a σ-structure and $B \subseteq A$, then $\mathfrak{A} \restriction B$ denotes the substructure of $\mathfrak{A}$ with the universe B.

An (atomic and proper) *k-type* (over a given signature) is a maximal consistent set of atoms or negated atoms over k distinct variables not containing equality atoms $x_i = x_j$ with $i \neq j$. If $\beta(x, y)$ is a 2-type over variables x and y, then $\beta \restriction x$ (respectively, $\beta \restriction y$) denotes the unique 1-type that is obtained from β by removing atoms with the variable y (respectively, the variable x). We denote by $\boldsymbol{\alpha}$ the set of all 1-types and by $\boldsymbol{\beta}$ the set of all 2-types (over a given signature). Note that $|\boldsymbol{\alpha}|$ and $|\boldsymbol{\beta}|$ are bounded exponentially in the size of the signature. We often identify a type with the conjunction of all its elements.

For a given σ-structure $\mathfrak{A}$ and $a \in A$ we say that *a realizes* a 1-type α if α is the unique 1-type such that $\mathfrak{A} \models \alpha[a]$. We denote by $tp^{\mathfrak{A}}(a)$ the 1-type realized by a. Similarly, for distinct $a, b \in A$, we denote by $tp^{\mathfrak{A}}(a, b)$ the unique 2-type *realized* by the pair a, b, i.e. the 2-type β such that $\mathfrak{A} \models \beta[a, b]$. In general, for finite $B, C \subseteq A$, $B \cap C = \emptyset$, by $tp^{\mathfrak{A}}(B, C)$ we denote the similarity type of the substructure $\mathfrak{A} \restriction (B \cup C)$ (or, in other words, its $card(B \cup C)$-type).

Assume $\mathfrak{A}$ is a σ-structure and $B, C \subseteq A$. We denote by $\boldsymbol{\alpha}^{\mathfrak{A}}$ (respectively, $\boldsymbol{\alpha}^{\mathfrak{A}}[B]$) the set of all 1-types realized in $\mathfrak{A}$ (respectively, realized in $\mathfrak{A} \restriction B$), and by $\boldsymbol{\beta}^{\mathfrak{A}}$ (respectively, $\boldsymbol{\beta}^{\mathfrak{A}}[B]$) the set of all 2-types realized in $\mathfrak{A}$ (respectively, realized in $\mathfrak{A} \restriction B$). We denote by $\boldsymbol{\beta}^{\mathfrak{A}}[a, B]$ the set of all 2-types $tp^{\mathfrak{A}}(a, b)$ with $b \in B$, and by $\boldsymbol{\beta}^{\mathfrak{A}}[B, C]$ the set of all 2-types $tp^{\mathfrak{A}}(b, c)$ with $b \in B, c \in C$.

Let γ be a σ-sentence of the form $\forall x\, \exists y\, \psi(x, y)$ and $a \in A$. We say that an element $b \in A$ is a *γ-witness* for a in the structure $\mathfrak{A}$ if $\mathfrak{A} \models \psi(a, b)$; b is a *proper γ-witness*, if b is a γ-witness and $a \neq b$.

Scott normal form. As with FO^2, so too with FO^2_T, analysis is facilitated by the availability of normal forms.

▶ **Definition 1.** An FO^2-sentence Ψ is in *Scott normal form* if it is of the following form: $\forall x \forall y\, \psi_0(x, y) \wedge \bigwedge_{i=1}^{M} \forall x \exists y\, \psi_i(x, y)$, where every ψ_i is quantifier-free and includes unary and binary predicate letters only.

Without loss of generality we suppose that for $i \geq 1$, $\psi_i(x, y)$ entails $x \neq y$ (replacing $\psi_i(x, y)$ with $(\psi_i(x, y) \vee \psi_i(x, x)) \wedge x \neq y$, which is sound over all structures with at least two elements).

Two formulas are said to be *strongly equisatisfiable* if they are satisfiable over the same universe. The following Lemma is typical for two-variable logics.

▶ **Lemma 2** ([25, 5]). *For every formula $\varphi \in \mathrm{FO}^2$ one can compute in polynomial time a strongly equisatisfiable normal form formula $\psi \in \mathrm{FO}^2$ over a new signature whose length is linear in the length of φ.*

Suppose the signature σ consists of predicates of arity at most 2. To define a σ-structure $\mathfrak{A}$, it suffices to specify the 1-types and 2-types realized by elements and pairs of elements from the universe A. In the presence of a transitive relation, we classify 2-types according to the transitive connection between x and y. And so, we distinguish $\boldsymbol{\beta}^{\rightarrow}$, $\boldsymbol{\beta}^{\leftarrow}$, $\boldsymbol{\beta}^{\leftrightarrow}$ and $\boldsymbol{\beta}^{-}$ such that $\boldsymbol{\beta} = \boldsymbol{\beta}^{\rightarrow} \dot\cup \boldsymbol{\beta}^{\leftarrow} \dot\cup \boldsymbol{\beta}^{\leftrightarrow} \dot\cup \boldsymbol{\beta}^{-}$ and for instance: $\beta \in \boldsymbol{\beta}^{\rightarrow}$ iff $(T(x,y) \wedge \neg T(y,x)) \in \beta$, $\beta \in \boldsymbol{\beta}^{\leftrightarrow}$ iff $(T(x,y) \wedge T(y,x)) \in \beta$, etc.

For a quantifier-free formula $\varphi(x,y)$ we use superscripts $^{\rightarrow}$, $^{\leftarrow}$, $^{\leftrightarrow}$ and $^{-}$ to define new formulas that explicitly specify the transitive connection between x and y. For instance, for a quantifier-free formula $\varphi(x,y) \in \mathrm{FO}^2_T$ we let $\varphi^{\rightarrow}(x,y) := \varphi(x,y) \wedge T(x,y) \wedge \neg T(y,x)$.

This conversion of FO^2_T-formulae leads to the the following variant of the Scott normal form:

$$\forall x \forall y\, \psi_0 \wedge \bigwedge_{i=1}^{m} \gamma_i \wedge \bigwedge_{i=1}^{\overline{m}} \delta_i \tag{1}$$

where $\gamma_i = \forall x \exists y\, \psi_i^{d_i}(x,y)$ with $d_i \in \{^{\rightarrow}, ^{\leftarrow}, ^{-}\}$, and $\delta_i = \forall x \exists y\, \psi_i^{\leftrightarrow}(x,y)$.

For a fixed sentence Ψ in normal form (1) we often write $\gamma_i \in \Psi$ to indicate that γ_i is a conjunct of Ψ of the form $\forall x \exists y\, \psi_i^{d_i}(x,y)$.

▶ **Lemma 3.** *Let φ be an FO^2_T-formula over a signature τ. We can compute, in polynomial time, a strongly equisatisfiable FO^2_T-formula in normal form, over a signature σ consisting of τ together with a number of additional unary and binary predicates.*

Sketch. We employ the standard technique of renaming subformulas familiar from [25] and [5], noting that any formula $\exists y \psi$ is logically equivalent to $\exists y \psi^{\rightarrow} \vee \exists y \psi^{\leftarrow} \vee \exists y \psi^{\leftrightarrow} \vee \exists y \psi^{-}$. ◀

The following trivial observation will be very useful in the paper.

▶ **Proposition 4.** *Assume $\mathfrak{A}$ is a σ-structure and Ψ is a FO^2_T-sentence over σ in normal form (1). Then $\mathfrak{A} \models \Psi$ if and only if*

(a) for every $a \in A$, for every γ_i $(1 \leq i \leq m)$ there is a γ_i-witness for a in $\mathfrak{A}$,
(b) for every $a \in A$, for every δ_i $(1 \leq i \leq \overline{m})$ there is a δ_i-witness for a in $\mathfrak{A}$,
(c) for every $a, b \in A$, $tp^{\mathfrak{A}}(a,b) \models \psi_0$,
(d) $T^{\mathfrak{A}}$ is transitive in $\mathfrak{A}$.

A small clique property for FO^2_T. Let $\mathfrak{A}$ be a σ-structure. A subset B of A is called *T-connected* if $\boldsymbol{\beta}[B] \subseteq \boldsymbol{\beta}^{\leftrightarrow}[\mathfrak{A}]$. Maximal T-connected subsets of A are called *cliques*. Note that if $\boldsymbol{\beta}[a,\mathfrak{A}] \cap \boldsymbol{\beta}^{\leftrightarrow}[A] = \emptyset$, for some $a \in A$, then $\{a\}$ is a clique. We prove the following *small clique property*.

▶ **Lemma 5.** *Let Ψ be a satisfiable FO^2_T-sentence in normal form, over a signature σ. Then there exists a model of Ψ in which the size of each clique is bounded exponentially in $|\sigma|$.*

We first show how to replace a single clique in models of normal-form FO^2_T-sentences by an equivalent small one. The idea is not new, it was used in [27] to show that T-cliques in models of GF^2+TG can be replaced by appropriate small structures called T-petals (Lemma 17). Later, in [16] it was proved that for any structure $\mathfrak{A}$ and its substructure $\mathfrak{B}$, one may replace $\mathfrak{B}$ by an alternative structure $\mathfrak{B}'$ of a bounded size in such a way that the obtained structure $\mathfrak{A}'$ and the original structure $\mathfrak{A}$ satisfy exactly the same normal form FO^2 formulas. Due to space limitations, a precise statement of the latter lemma and the proof of the small clique model property will appear in the full version of the paper.

3 Splices and duplicability

In the remainder of the paper we fix a relational signature σ and assume Ψ is an FO^2_T-sentence in normal form (1). By Lemma 5, we may already assume (and we do so) that models of Ψ have the small clique property. In the next two sections we assume that Ψ is satisfiable and, if not stated otherwise, $\mathfrak{A} \models \Psi$.

In this section we analyze properties of models of Ψ on the level of cliques rather than individual elements. We give here the key technical argument of the paper (Corollary 11). It says, roughly speaking, that if $\mathfrak{A} \models \Psi$ and elements of a finite subset F of the universe A have their γ_i-witnesses in several "similar" cliques then $\mathfrak{A}$ can be extended by one new clique, where all the elements of F have their γ_i-witnesses.

First, we need to introduce some new notions and notation. For $a \in A$ denote by $Cl^{\mathfrak{A}}(a)$ the unique clique $C \subseteq A$ with $a \in C$. When $F \subseteq A$, denote $Cl^{\mathfrak{A}}(F) = \{Cl^{\mathfrak{A}}(a) : a \in F\}$ and finally, $Cl^{\mathfrak{A}} = Cl^{\mathfrak{A}}(A)$. Note that whenever $B \in Cl^{\mathfrak{A}}$, and $a \in A$ is an element outside the clique B, then the 2-types between a and any element $b \in B$ belong to the same subset of $\boldsymbol{\beta}$, i.e. either to $\boldsymbol{\beta}^{\rightarrow}$, $\boldsymbol{\beta}^{\leftarrow}$ or $\boldsymbol{\beta}^{-}$. So, we might speak about elements of $\mathfrak{A}$ connected with the clique B using "directed" edges. Similarly, we can identify cliques connected with B using "incoming" and "outgoing" edges.

▶ **Definition 6.** Let $\mathfrak{A}$ be a σ-structure and let $B \in Cl^{\mathfrak{A}}$. Define:
- $In^{\mathfrak{A}}(B) \overset{def}{=} \{tp^{\mathfrak{A}}(C) : C \in Cl^{\mathfrak{A}} \text{ and } \boldsymbol{\beta}[C,B] \subseteq \boldsymbol{\beta}^{\rightarrow}\}$,
- $Out^{\mathfrak{A}}(B) \overset{def}{=} \{tp^{\mathfrak{A}}(C) : C \in Cl^{\mathfrak{A}} \text{ and } \boldsymbol{\beta}[C,B] \subseteq \boldsymbol{\beta}^{\leftarrow}\}$,
- $sp^{\mathfrak{A}}(B) = \langle tp^{\mathfrak{A}}(B), In^{\mathfrak{A}}(B), Out^{\mathfrak{A}}(B)\rangle$,
- $Sp^{\mathfrak{A}} = \{sp^{\mathfrak{A}}(B) : B \in Cl^{\mathfrak{A}}\}$. Elements of $Sp^{\mathfrak{A}}$ are called $\mathfrak{A}$-*splices*.

Splices define cliques reachable from a given clique via T.

We say that two cliques $B, B' \in Cl^{\mathfrak{A}}$ *realize the same splice*, written $B \equiv^{\mathfrak{A}} B'$, if $sp^{\mathfrak{A}}(B) = sp^{\mathfrak{A}}(B')$. When $\mathfrak{A}$ is understood we often omit the superscript in $\equiv^{\mathfrak{A}}$ and write $\equiv$. Note that $\equiv^{\mathfrak{A}}$ is an equivalence relation on $Cl^{\mathfrak{A}}$. Moreover, if we have an a priori upper bound on the size of cliques in $Cl^{\mathfrak{A}}$, then $Cl^{\mathfrak{A}}/_{\equiv}$ is finite (and of bounded cardinality).

Additionally, we distinguish the set $\mathbb{K}(\mathfrak{A})$ of *unique cliques* in $\mathfrak{A}$: $\mathbb{K}(\mathfrak{A}) = \{B \in Cl^{\mathfrak{A}} : card([B]_{\equiv}) = 1\}$ and the corresponding subset $K(\mathfrak{A})$ of the universe of $\mathfrak{A}$, that consists of the elements of the unique cliques: $K(\mathfrak{A}) = \bigcup_{B \in \mathbb{K}(\mathfrak{A})} B$.

Our task is now to show that any model of Ψ containing a non-unique clique B can be extended into a new model of Ψ by adding a copy of B. The copy of B is added in such a way that it also provides, for all conjuncts of the form γ_i, all the witnesses for elements outside the two cliques that have been provided by B. This property will be explored later, when new models will be constructed by removing *segments* (i.e. sets of cliques) from given ones.

For every $\gamma_i \in \Psi$ (recall $\gamma_i = \forall x \exists y\, \psi_i^{d_i}(x,y)$ with $d_i \in \{^{\rightarrow}, ^{\leftarrow}, ^{-}\}$) and for every $a \in A$ we define $W_i^{\mathfrak{A}}(a)$ as the set of all proper γ_i-witnesses for a in $\mathfrak{A}$:

$$W_i^{\mathfrak{A}}(a) \overset{def}{=} \{b \in A : \mathfrak{A} \models \psi_i(a,b),\, b \neq a\}.$$

Similarly, for every $\gamma_i \in \Psi$ and for every $F \subseteq A$ we define $W_i^{\mathfrak{A}}(F) \overset{def}{=} \bigcup_{a \in F} W_i^{\mathfrak{A}}(a)$.

The following claim states that every non-unique clique in a given model can be properly duplicated, as informally described above.

▶ **Claim 7** (Duplicability). *Assume $\mathfrak{A} \models \Psi$, $B_1 \in Cl^{\mathfrak{A}}$ and $B_1 \notin \mathbb{K}(\mathfrak{A})$. There is an extension $\mathfrak{A}'$ of $\mathfrak{A}$ by one new clique D such that*

1. $\mathfrak{A}' \models \Psi$,

2. *for every conjunct γ_i of Ψ, for every $a \in A$ we have:*
$$B_1 \cap W_i^{\mathfrak{A}}(a) \neq \emptyset \quad \textit{iff} \quad D \cap W_i^{\mathfrak{A}'}(a) \neq \emptyset, \textit{ and}$$
3. $sp^{\mathfrak{A}'}(D) = sp^{\mathfrak{A}'}(B_1) = sp^{\mathfrak{A}}(B_1)$.

Proof. Let $\mathfrak{A} \models \Psi$, $B_1 \in Cl^{\mathfrak{A}}$. Since $B_1 \notin \mathbb{K}(\mathfrak{A})$, there exists $B_2 \in Cl^{\mathfrak{A}}$, $B_2 \neq B_1$ such that $sp^{\mathfrak{A}}(B_2) = sp^{\mathfrak{A}}(B_1)$. Assume $\mathfrak{D}$ is a duplicate of $\mathfrak{A} \restriction B_1$, $D \cap A = \emptyset$, $f_1 : D \mapsto B_1$ and $f_2 : D \mapsto B_2$ are appropriate isomorphism functions. Let $\mathfrak{A}'$ be an extension of $\mathfrak{A}$ with the universe $A \dot{\cup} D$ such that:

- $tp^{\mathfrak{A}'}(d, b) \overset{def}{=} tp^{\mathfrak{A}}(b, f_2(d))$, for every $b \in B_1$, $d \in D$ (note that $\boldsymbol{\beta}[d, B_1] = \boldsymbol{\beta}[B_1, f_2(d)]$ and so $\boldsymbol{\beta}[D, B_1] = \boldsymbol{\beta}[B_1, B_2]$),
- $tp^{\mathfrak{A}'}(d, a) \overset{def}{=} tp^{\mathfrak{A}}(f_1(d), a)$, for every $a \in A \setminus (B_1 \cup D)$, $d \in D$ (note that $\boldsymbol{\beta}[d, A \setminus B_1] = \boldsymbol{\beta}[f_1(d), A \setminus B_1]$ and so $\boldsymbol{\beta}[D, A \setminus B_1] = \boldsymbol{\beta}[B_1, A \setminus B_1]$).

To see that $\mathfrak{A}' \models \Psi$ one can show that conditions (a)–(d) of Proposition 4 hold for $\mathfrak{A}'$. ◀

Using the above claim we may build *saturated models* in the following sense.

▶ **Definition 8.** Assume $\mathfrak{A} \models \Psi$. We say that $\mathfrak{A}$ is *witness-saturated*, if $\mathfrak{A}$ has the small clique property and for every $a \in A$, for every $\gamma_i \in \Psi$ $(1 \leq i \leq m)$
$$W_i(a) \subseteq K(\mathfrak{A}) \quad \text{or} \quad W_i(a) \text{ is infinite.}$$
Note that if a witness-saturated model $\mathfrak{A}$ is finite then $A = K(\mathfrak{A})$. By Lemma 5 and by iterative application of Claim 7 we get the following.

▶ **Lemma 9** (Saturated model)**.** *Every satisfiable normal form sentence Ψ has a countable witness-saturated model. Additionally, if $\mathfrak{A}$ is witness-saturated, then the extension $\mathfrak{A}'$ given by Claim 7 is also witness-saturated.*

The above Lemma is essential for the proof of the key technical tool for the paper, Corollary 11, given below. It says that when several elements $a_1, a_2, \ldots, a_n$ of a model $\mathfrak{A}$ have γ_i-witnesses in several distinguished cliques that realize the same splice, one can extend the model $\mathfrak{A}$ by a single clique D (realizing the same splice) in which $a_1, a_2, \ldots, a_n$ have their γ_i-witnesses. The proof is based on a more subtle (than in Claim 7) analysis of models of a normal form sentence Ψ given in Claim 10. In the Claim note that whenever $\boldsymbol{\beta}(C_1, C_2) \in \boldsymbol{\beta}^{\leftarrow}$ then $\boldsymbol{\beta}(C_2, C_1) \notin \boldsymbol{\beta}^{\leftarrow}$.

▶ **Claim 10.** *Assume $\mathfrak{A}$ is countable witness-saturated and $\gamma_i \in \Psi$. Let $C_1, C_2, B_1, B_2 \in Cl^{\mathfrak{A}}$, $\boldsymbol{\beta}(C_1, C_2) \notin \boldsymbol{\beta}^{\leftarrow}$, $B_1 \neq B_2$, $B_1 \cap W_i(C_1) \neq \emptyset$, $B_2 \cap W_i(C_2) \neq \emptyset$, $C_1 \notin \mathbb{K}(\mathfrak{A})$, $C_1 \equiv C_2$ and $B_1 \equiv B_2$. Then, there exists an extension $\mathfrak{A}_1$ of $\mathfrak{A}$ by at least one clique D such that*
(i) $\mathfrak{A}_1 \models \Psi$ *and* $\mathfrak{A}_1$ *is witness-saturated,*
(ii) *for every* $a \in A$: *if* $B_1 \cap W_i^{\mathfrak{A}}(a) \neq \emptyset$ *then* $D \cap W_i^{\mathfrak{A}_1}(a) \neq \emptyset$,
(iii) *for every* $a \in C_2$: *if* $B_2 \cap W_i^{\mathfrak{A}}(a) \neq \emptyset$ *then* $D \cap W_i^{\mathfrak{A}_1}(a) \neq \emptyset$,
(iv) $sp^{\mathfrak{A}_1}(D) = sp^{\mathfrak{A}_1}(B_1) = sp^{\mathfrak{A}}(B_1)$.

Proof. We have several cases. In each case we add a duplicate D of the clique B_1 where both C_1 and C_2 will get their γ_i-witnesses. We sketch only one of the interesting cases.
Case 1. $\boldsymbol{\beta}[C_1, B_1] \subseteq \boldsymbol{\beta}^{-}$ and $\boldsymbol{\beta}[C_1, C_2] \subseteq \boldsymbol{\beta}^{\rightarrow} \cup \boldsymbol{\beta}^{-}$. In this case $C_1 \neq C_2$ and $\gamma_i = \forall x \exists y\, \psi_i^{-}(x, y)$. So, $\boldsymbol{\beta}[C_1, B_1] \subseteq \boldsymbol{\beta}^{-}$ and $\boldsymbol{\beta}[C_2, B_2] \subseteq \boldsymbol{\beta}^{-}$.
Subcase 1.a. $\boldsymbol{\beta}[C_1, C_2] \in \boldsymbol{\beta}^{\rightarrow}$, $\boldsymbol{\beta}[B_1, B_2] \subseteq \boldsymbol{\beta}^{-}$, $\boldsymbol{\beta}[C_1, B_2] \subseteq \boldsymbol{\beta}^{\rightarrow}$ and $\boldsymbol{\beta}[B_1, C_2] \in \boldsymbol{\beta}^{\rightarrow}$.

The construction proceeds in four steps.
Step 1 (copying of B_1). Let $\mathfrak{A}'$ be the witness saturated extension of $\mathfrak{A}$ by one new clique D – a duplicate of the clique B_1 given by Claim 7 and Lemma 9. Observe that all conditions *(i)–(iv)* hold in $\mathfrak{A}'$ except *(iii)* since by construction of $\mathfrak{A}'$ $\boldsymbol{\beta}^{\mathfrak{A}'}[C_2, D] = \boldsymbol{\beta}^{\mathfrak{A}}[C_2, B_1] \subseteq \boldsymbol{\beta}^{\leftarrow}$.

Step 2 (modification of $tp^{\mathfrak{A}'}(C_2, D)$). To ensure that *(iii)* holds, a new structure $\mathfrak{A}_2$ is built by defining $tp^{\mathfrak{A}_2}(C_2, D) \stackrel{def}{=} tp^{\mathfrak{A}'}(C_2, B_2)$.
Step 3 (transitivity correction). To ensure that T is transitive we construct a structure $\mathfrak{A}_3$: for every $X \in Cl^{\mathfrak{A}_2}$, if $\boldsymbol{\beta}[D, X] \subseteq \boldsymbol{\beta}^{\rightarrow}$ and $\boldsymbol{\beta}[X, C_2] \subseteq \boldsymbol{\beta}^{\rightarrow}$ then replace $tp^{\mathfrak{A}_2}(D, X)$ by $tp^{\mathfrak{A}_2}(B_2, X)$. One can observe that $\boldsymbol{\beta}[X, B_2] \subseteq \boldsymbol{\beta}^{-}$, so T is transitive in $\mathfrak{A}_3$.
Step 4 (γ_j-witness in X correction). Let $X \in Cl^{\mathfrak{A}_2}$ be such that the type $tp^{\mathfrak{A}_2}(D, X)$ is changed in Step 3. We show that then there is a clique $X_2 \in Cl^{\mathfrak{A}_3}$ such that $tp^{\mathfrak{A}_3}(X_2) = tp^{\mathfrak{A}_3}(X)$ and $\boldsymbol{\beta}^{\mathfrak{A}_3}[D, X_2] \subseteq \boldsymbol{\beta}^{\rightarrow}$. For, observe that $tp^{\mathfrak{A}_3}(X) \in Out^{\mathfrak{A}_3}(B_1)$ and so $tp^{\mathfrak{A}_3}(X) \in Out^{\mathfrak{A}_3}(B_2)$ since $B_1 \equiv B_2$. Let $X_1 \in Cl^{\mathfrak{A}_3}$, $tp^{\mathfrak{A}_3}(X_1) = tp^{\mathfrak{A}_3}(X)$ and $\boldsymbol{\beta}[B_2, X_1] \subseteq \boldsymbol{\beta}^{\rightarrow}$. Since $\boldsymbol{\beta}[C_1, B_2] \subseteq \boldsymbol{\beta}^{\rightarrow}$ we have $\boldsymbol{\beta}[C_1, X_1] \subseteq \boldsymbol{\beta}^{\rightarrow}$ and so $tp^{\mathfrak{A}_3}(X_1) \in Out^{\mathfrak{A}_3}(C_1)$. Hence, since $C_1 \equiv C_2$ there exist $X_2 \in Cl^{\mathfrak{A}_3}$ such that $tp^{\mathfrak{A}_3}(X_2) = tp^{\mathfrak{A}_3}(X_1)$ and $\boldsymbol{\beta}[C_2, X_2] \subseteq \boldsymbol{\beta}^{\rightarrow}$. Since $\boldsymbol{\beta}[B_1, C_2] \subseteq \boldsymbol{\beta}^{\rightarrow}$, so $\boldsymbol{\beta}[B_1, X_2] \subseteq \boldsymbol{\beta}^{\rightarrow}$, and hence, by construction of $\mathfrak{A}'$, $\boldsymbol{\beta}[D, X_2] \subseteq \boldsymbol{\beta}^{\rightarrow}$. To obtain the required model $\mathfrak{A}_1$ replace $tp^{\mathfrak{A}_3}(D, X_2)$ by $tp^{\mathfrak{A}_2}(D, X)$ $(= tp^{\mathfrak{A}}(B_1, X))$.

Correctness proof of the above construction, as well as other cases, will appear in the full version of the paper. ◀

▶ **Corollary 11.** *Assume $\mathfrak{A}$ is countable witness-saturated, $\gamma_i \in \Psi$ and $X \in Sp^{\mathfrak{A}}$. Let $F \subseteq A \setminus K(\mathfrak{A})$ be a finite set such that for every $a \in F$ there is $b \in W_i^{\mathfrak{A}}(a)$ such that $b \notin K(\mathfrak{A})$ and $sp(Cl^{\mathfrak{A}}(b)) = X$. Then, there exists an extension $\mathfrak{A}'$ of $\mathfrak{A}$ by at least one clique D such that*

(i) $\mathfrak{A}' \models \Psi$ and $\mathfrak{A}'$ is witness-saturated,
(ii) $D \cap W_i^{\mathfrak{A}'}(a) \neq \emptyset$, for every $a \in F$,
(iii) $sp^{\mathfrak{A}'}(D) = X$.

Proof. Let $F = \{a_1, a_2, \ldots, a_p\}$, where a_1 denotes an element of F such that for every $a \in F \setminus \{a_1\}$, $tp^{\mathfrak{A}}(a_1, a) \notin \boldsymbol{\beta}^{\leftarrow}$. (Note that a_1 can always be found since F is finite and T is transitive in $\mathfrak{A}$.) We iteratively apply Claim 10. Denote $\mathfrak{A}^1 = \mathfrak{A}$ and for $k = 2, 3, \ldots, p$ let $\mathfrak{A}^k$ and D^k be the extension of $\mathfrak{A}_1^{k-1}$ by at least one clique D^k given by Claim 10 for a_1 and a_k. Obviously, for every k $(2 \leq k \leq p)$ we have $D^k \cap W_i^{\mathfrak{A}'}(a) \neq \emptyset$, for every $a \in \{a_1, a_2, \ldots, a_k\}$. ◀

4 Canonical models

In this section we analyze properties of models of Ψ on the level of segments which consist of several cliques, and constitute a partition $S_0, S_1, \ldots$ of the universe of a model. Every segment S_j has a fixed (doubly exponential) size and is meant to contain all γ_i-witnesses for elements from earlier segments $S_0, S_1, \ldots, S_{j-1}$. On this level of abstraction cliques and splices of a model become less important.

▶ **Definition 12.** A finite subset $S \subset A$ is a *segment* in $\mathfrak{A}$ if $Cl^{\mathfrak{A}}(a) \subseteq S$ for every $a \in S$.

In the following we reserve the letter S (possibly decorated) to denote segments. Define $s = |Sp(\sigma)|$ and denote by h the bound of the size of each clique in a small-clique σ-structure, given by Lemma 5. Note that s is doubly exponential and h is exponential in $|\sigma|$.

We first prove a generalization of Corollary 11. It says, roughly speaking, that if $\mathfrak{A} \models \Psi$ and F is a finite subset of A, then it is possible to extend $\mathfrak{A}$ by a segment of fixed cardinality in which all elements of F have their γ_i-witnesses, for every i $(1 \leq i \leq m)$.

▶ **Lemma 13** (Witnesses compression). *Assume $\mathfrak{A}$ is a countable witness-saturated model of Ψ and $F \subseteq A \setminus K(\mathfrak{A})$ is finite. There is a witness-saturated extension $\mathfrak{A}'$ of $\mathfrak{A}$ by a segment S such that*

1. $\mathfrak{A}' \models \Psi$,
2. $|S| \leq m \cdot s \cdot h$,
3. *for every* $\gamma_i \in \Psi$, *for every* $a \in F$, *if* $W_i^{\mathfrak{A}}(a) \setminus K(\mathfrak{A}) \neq \emptyset$, *then* $W_i^{\mathfrak{A}'}(a) \cap S \neq \emptyset$.

Proof. First, for given i $(1 \leq i \leq m)$ and $X \in Sp^{\mathfrak{A}}$ let $F_i^X \subseteq S$ be a maximal subset of F such that for every $a \in F_i^X$ there is $b \in W_i^{\mathfrak{A}}(a)$ such that $b \notin K(\mathfrak{A})$ and $sp(Cl^{\mathfrak{A}}(b)) = X$. Now, for every i $(1 \leq i \leq m)$ and for every $X \in Sp^{\mathfrak{A}}$ iteratively apply Corollary 11 for F_i^X and denote each new clique added in the process by D_i^X. Let S be the segment consisting of elements of the new cliques: $S \overset{def}{=} \bigcup_{1 \leq i \leq m} \bigcup_{X \in Sp^{\mathfrak{A}}} D_i^X$.

Condition (i) of Lemma 11 implies that $\mathfrak{A}' \models \Psi$. Obviously, $|S| \leq m \cdot s \cdot h$. To show that condition 3 of our lemma holds, assume $\gamma_i \in \Psi$, $a \in F$ and there exists $b \in W_i^{\mathfrak{A}}(a)$ such that $b \notin K(\mathfrak{A})$. So $a \in F_i^X$, where $X = sp(Cl^{\mathfrak{A}}(b))$. By condition (ii) of Lemma 11 we obtain $D_i^X \cap W_i^{\mathfrak{A}'}(a) \neq \emptyset$, and so, $W_i^{\mathfrak{A}'}(a) \cap S \neq \emptyset$. ◀

▶ **Definition 14.** A segment $S \subsetneq A$ is *redundant in* $\mathfrak{A}$, if for every $a \in A \setminus S$ and for every $\gamma_i \in \Psi$ we have: $W_i^{\mathfrak{A}}(a) \cap S \neq \emptyset$ implies there exists $c \in A \setminus S$ such that $c \in W_i^{\mathfrak{A}}(a)$.

▶ **Claim 15.** *If* $\mathfrak{A} \models \Psi$ *and* $S \subsetneq A$ *is redundant in* $\mathfrak{A}$, *then* $\mathfrak{A} \restriction (A \setminus S) \models \Psi$.

Proof. Every subgraph of a transitive graph is also transitive. Conditions (a)–(c) of Proposition 4 obviously hold for $\mathfrak{A} \restriction (A \setminus S)$. ◀

▶ **Definition 16.** A model $\mathfrak{A}$ of Ψ is *narrow* if there is an infinite partition $P_A = \{S_0, S_1, \ldots\}$ of the universe A such that:
1. $K(\mathfrak{A}) \subset S_0$, $|S_0| \leq (m+1) \cdot s \cdot h$,
2. $|S_j| \leq m \cdot s \cdot h$, for every $j \geq 1$,
3. for every $j \geq 0$, for every $e \in \bigcup_{k=0}^{j} S_k$ and for every $\gamma_i \in \Psi$,
   if $W_i^{\mathfrak{A}}(e) \cap S_0 = \emptyset$, then $W_i^{\mathfrak{A}}(e) \cap S_{j+1} \neq \emptyset$.

▶ **Lemma 17.** *Every infinitely satisfiable sentence* Ψ *has a narrow model.*

Proof. Assume $\mathfrak{A}$ is an infinite witness-saturated model of Ψ that exists by Lemma 13. For $\gamma_i \in \Psi$ and $a \in A$ denote by $\bar{\gamma}_i(a)$ an arbitrarily chosen element $b \in W_i^{\mathfrak{A}}(a)$. Define $\mathfrak{A}_0 = \mathfrak{A}$ and $S_0 = K(\mathfrak{A}) \cup \bigcup_{a \in K(\mathfrak{A})} Cl^{\mathfrak{A}}(\bar{\gamma}_i(a))$.
Now, for $j = 0, 1, 2, \ldots$ define:
- $\mathfrak{A}_{j+1} = \mathfrak{A}'_j$, where $\mathfrak{A}'_j$ is the extension of $\mathfrak{A}_j$ given by Lemma 13 for $F = \bigcup_{k=0}^{j} S_k$,
- $S_{j+1} = B$, where B is the finite set given by Lemma 13; B extends $\mathfrak{A}_j$ to $\mathfrak{A}_{j+1}$ in such a way, that:
  - $\mathfrak{A}_{j+1} \models \Psi$,
  - $|B| \leq m \cdot s \cdot h$,
  - for every $\gamma_i \in \Psi$, for every $a \in F$, if $W_i^{\mathfrak{A}}(a) \setminus K(\mathfrak{A}_j) \neq \emptyset$, then $W_i^{\mathfrak{A}_{j+1}}(a) \cap B \neq \emptyset$.

Now, define $\mathfrak{A}' = (\bigcup_{k=0}^{\infty} \mathfrak{A}_k) \restriction \bigcup_{k=0}^{\infty} S_k$. By Claim 15 and Lemma 13, it is easy to see, that $\mathfrak{A}'$ is a narrow model of Ψ with partition $P_{A'} = \{S_0, S_1, \ldots\}$. ◀

▶ **Definition 18.** Assume $\mathfrak{A}$ is a σ-structure, $x, y \in \mathbb{N}^+$ and B, B', C, C' are finite subsets of A with fixed orderings: $B = \{b_1 < \ldots < b_x\}$, $B' = \{b'_1 < \ldots < b'_y\}$, $C = \{c_1 < \ldots < c_x\}$, $C' = \{c'_1 < \ldots, c'_y\}$ such that $B \cap B' = \emptyset$, $C \cap C' = \emptyset$.

A *connection type* of B and B' in $\mathfrak{A}$ is the structure $\langle B, B' \rangle \overset{def}{=} \mathfrak{A} \restriction (B \cup B')$. Two connection types $\langle B, B' \rangle$ and $\langle C, C' \rangle$ are *the same connection types in* $\mathfrak{A}$, denoted $\langle B, B' \rangle \cong^{\mathfrak{A}} \langle C, C' \rangle$, if the function $f : B \cup B' \mapsto C \cup C'$ defined by $f(b_j) = c_j$ and $f(b'_j) = c'_j$ is an isomorphism of $\langle B, B' \rangle$ and $\langle C, C' \rangle$.

▶ **Definition 19.** Assume $\mathfrak{A}$ is a narrow model of Ψ and $P_A = \{S_0, S_1, \ldots\}$ is any partition satisfying conditions 1-3 of Definition 16. We say that $\mathfrak{A}$ is *canonical* if for every $j, k \in \mathbb{N}^+$, $0 < j < k$, we have $\langle S_j, S_0\rangle \cong^{\mathfrak{A}} \langle S_1, S_0\rangle$, and $\langle S_k, S_j\rangle \cong^{\mathfrak{A}} \langle S_2, S_1\rangle$.

▶ **Lemma 20.** *Every infinitely satisfiable sentence Ψ has a canonical model.*

Proof. Let $\mathfrak{A}$ be a narrow model of Ψ with partition $P_A = \{S_0, S_1, \ldots\}$ given by Definition 16. Additionally, assume that in every segment S_j, $j \geq 0$, there is a fixed linear ordering.

Observe first that for every $p > 0$, S_p is redundant in $\mathfrak{A}$. For, assume (cf. Definition 14) $b \in S_p$, $a \in A \setminus S_p$ and $b \in W_i^{\mathfrak{A}}(a)$. Assume $a \in S_q$ and take $j \in \mathbb{N}^+$ such that $j > \max\{p, q\}$. By Definition 16, we have that if $W_i^{\mathfrak{A}}(a) \cap S_0 = \emptyset$, then $W_i^{\mathfrak{A}}(a) \cap S_{j+1} \neq \emptyset$. So, there is $c \in S_0 \cup S_{j+1}$ such that $c \in W_i^{\mathfrak{A}}(a)$. Similarly, for every infinite $V \subset \mathbb{N}^+$, the segment $\bigcup_{j \in \mathbb{N}^+ \setminus V} S_j$ is redundant in $\mathfrak{A}$ and, by Claim 15, $\mathfrak{A} \restriction \bigcup_{j \in V \cup \{0\}} S_j \models \Psi$.

To construct the canonical model we first find an infinite set $V \subset \mathbb{N}^+$ such that for every $j,\ l \in V$, $j \neq l$, $\langle S_l, S_0\rangle \cong^{\mathfrak{A}} \langle S_j, S_0\rangle$. Observe that the set V does exist (by the infinite Ramsey theorem, there are infinitely many segments in P_A and only a finite number of similarity types).

Secondly, we find an infinite set $W \subseteq V$ such that for every $i,\ j,\ k,\ l \in W$ with $i < j$ and $k < l$ we have $\langle S_j, S_i\rangle \cong^{\mathfrak{A}} \langle S_l, S_k\rangle$. (Again, W exists by the infinite Ramsey theorem).

Finally, we define $\mathfrak{A}' = \mathfrak{A} \restriction \bigcup_{j \in W \cup \{0\}} S_j$. By Claim 15, $\mathfrak{A}' \models \Psi$. Obviously, $\mathfrak{A}'$ is canonical with partition $P_{A'} = \{S_j : j \in \mathbb{N}^+, j = \#p\}$, where $\#p$ is the position number of $p \in W \cup \{0\}$. ◀

5 Decidability and complexity

From Lemma 20 we get immediately the following theorem.

▶ **Theorem 21.** *An FO^2_T-sentence Ψ is satisfiable if and only if there exist a σ-structure $\mathfrak{A}$ and $S_0, S_1, S_2, S_3 \subseteq A$, such that:*
1. *$|A| \leq (4m + 1) \cdot s \cdot h$,*
2. *either $S_1 = S_2 = S_3 = \emptyset$, or $\{S_0, S_1, S_2, S_3\}$ is a partition of A and then*
   - **a.** *$\langle S_1, S_0\rangle \cong^{\mathfrak{A}} \langle S_2, S_0\rangle \cong^{\mathfrak{A}} \langle S_3, S_0\rangle$,*
   - **b.** *$\langle S_2, S_1\rangle \cong^{\mathfrak{A}} \langle S_3, S_2\rangle \cong^{\mathfrak{A}} \langle S_3, S_1\rangle$,*
3. *for every $a, b \in A$, $tp^{\mathfrak{A}}(a, b) \models \psi_0$,*
4. *$T^{\mathfrak{A}}$ is transitive in $\mathfrak{A}$,*
5. *for every $j = 0, 1, 2$, for every $e \in S_j$ and for every $\gamma_i \in \Psi$,*
   if $W_i^{\mathfrak{A}}(e) \cap S_0 = \emptyset$, then $W_i^{\mathfrak{A}}(e) \cap S_{j+1} \neq \emptyset$.

Proof. ($\Rightarrow$) There are two cases. Either Ψ has only finite models and then, by Lemma 9, Ψ has a witness-saturated model $\mathfrak{A}$ with $A = K(\mathfrak{A})$. In this case, we put $S_0 = A$ and $S_1 = S_2 = S_3 = \emptyset$. Or, Ψ has infinite models, and then, by Lemma 20, Ψ has a canonical model $\mathfrak{A}'$ with partition $P_{A'} = \{S_0, S_1, \ldots\}$. In this case, we define $\mathfrak{A} \overset{def}{=} \mathfrak{A}' \restriction (S_0 \dot\cup S_1 \dot\cup S_2 \dot\cup S_3)$. Note that in either case $|S_0| \leq s \cdot h + m \cdot s \cdot h = (m + 1) \cdot s \cdot h$.
($\Leftarrow$) Define a structure $\mathfrak{A}'$ such that $A' \overset{def}{=} S_0 \,\dot\cup\, S_1 \,\dot\cup\, S_2 \,\dot\cup\, S_3 \,\dot\cup\, \dot{\bigcup}_{j=4}^{\infty} S_j$ and, for every $j, k \in \mathbb{N}^+$ $(0 < j < k) : \langle S_j, S_0\rangle \cong^{\mathfrak{A}'} \langle S_1, S_0\rangle$ and $\langle S_k, S_j\rangle \cong^{\mathfrak{A}'} \langle S_2, S_1\rangle$. It is obvious that $\mathfrak{A}'$ satisfies conditions (a)–(c) of Proposition 4. To show that T is transitive in $\mathfrak{A}'$ it suffices to prove that for every j, k, l $(0 \leq j \leq k \leq l)$, $T^{\mathfrak{A}' \restriction (S_j \cup S_k \cup S_l)}$ is transitive in $\mathfrak{A}' \restriction (S_j \cup S_k \cup S_l)$. The latter condition can be easily verified; hence, $\mathfrak{A}' \models \Psi$. ◀

▶ **Corollary 22.** *SAT(FO^2_T) $\in$ 2-NExpTime.*

Proof. To check whether a given FO^2_Tsentence Ψ is satisfiable we follow Theorem 21 and we obtain a nondeterministic double exponential time procedure, as described below.

```
1. Guess a $\sigma$-structure $\mathfrak{A}$ of cardinality $|A| \le (4m+1) \cdot s \cdot h$ and partition
   $P_A = \{S_0, S_1, S_2, S_3\}$;
2. Guess enumerations of every segment $S_0, S_1, S_2, S_3$;
3. If not:
   a. $\langle S_1, S_0\rangle \cong^{\mathfrak{A}} \langle S_2, S_0\rangle \cong^{\mathfrak{A}} \langle S_3, S_0\rangle$ and
   b. $\langle S_2, S_1\rangle \cong^{\mathfrak{A}} \langle S_3, S_2\rangle \cong^{\mathfrak{A}} \langle S_3, S_1\rangle$
   then reject;
4. For every $a, b \in A$, if $tp^{\mathfrak{A}}(a,b) \not\models \psi_0$ then reject;
5. For every $a, b, c \in A$, if not $(T^{\mathfrak{A}}(a,b) \wedge T^{\mathfrak{A}}(b,c) \Rightarrow T^{\mathfrak{A}}(a,c))$ then reject;
6. For every $j = 0, 1, 2$, for every $e \in S_j$, for every $\gamma_i \in \Psi$ such that $W_i^{\mathfrak{A}}(e) \cap S_0 = \emptyset$
   if $W_i^{\mathfrak{A}}(e) \cap S_{j+1} = \emptyset$ then reject;
7. Accept;
```

◀

6 Outlook

Since the finite model property fails for FO^2_T, an interesting question is whether the finite satisfiability problem is also decidable. Immediately from Lemma 17 we have the following observation.

▶ **Corollary 23.** *An FO^2_T-sentence Ψ is satisfiable if and only if*
Ψ has a model of cardinality $\le s \cdot h$, or Ψ has an infinite model.

The $s \cdot h$ bound on the size of the finite model of Ψ depends on the number of different σ-splices and the size of cliques in a structure with the small clique property. Unfortunately, this observation does not suffice to answer the finite satisfiability problem, as in general, one can imagine that a finite model contains several realizations of the same splice. So, to the best of our knowledge, the finite satisfiability problem for FO^2_T remains open. We believe that the latter problem is decidable; however, we suppose that an essential extension of the above approach is required to get the proof.

We also note that the 2-NExpTime bound for the satisfiability problem leaves a gap in complexity, as the best lower bound coming from the two-variable guarded logic with transitive guards is 2-ExpTime [10]. We believe that the upper bound proved in our paper can be improved; however, as FO^2_T does not enjoy the tree-like property, standard techniques using alternating machines cannot be applied directly.

Acknowledgements The authors acknowledge insightful comments from the anonymous referees that helped to improve the presentation and simplify some proofs in Section 4.

References

1 H. Andréka, J. van Benthem, and I. Németi. Modal languages and bounded fragments of predicate logic. *J. Philos. Logic*, 27:217–274, 1998.
2 M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and xml reasoning. In *Proc. of PODS-06*, pages 10–19, New York, NY, USA, 2006. ACM.
3 M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on words with data. In *LICS-06*, pages 7–16, 2006.
4 C. David, L. Libkin, and Tony Tan. On the satisfiability of two-variable logic over data words. In Christian G. Fermüller and Andrei Voronkov, editors, *LPAR (Yogyakarta)*, volume 6397 of *Lecture Notes in Computer Science*, pages 248–262. Springer, 2010.

5 E. Grädel, P. Kolaitis, and M. Vardi. On the decision problem for two-variable first-order logic. *Bull. of Symb. Logic*, 3(1):53–69, 1997.

6 E. Grädel and M. Otto. On Logics with Two Variables. *Theoretical Computer Science*, 224:73–113, 1999.

7 E. Grädel, M. Otto, and E. Rosen. Undecidability results on two-variable logics. *Arch. Math. Log.*, 38(4-5):313–354, 1999.

8 N. Immerman, A. M. Rabinovich, T. W. Reps, S. Sagiv, and G. Yorsh. The boundary between decidability and undecidability for transitive-closure logics. In *CSL*, pages 160–174, 2004.

9 Y. Kazakov. *Saturation-based decision procedures for extensions of the guarded fragment.* PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, 2006.

10 E. Kieroński. The two-variable guarded fragment with transitive guards is 2EXPTIME-Hard. In *Proc. of FOSSACS*, volume LNCS 2620, pages 299–312, 2003.

11 E. Kieroński. Results on the guarded fragment with equivalence or transitive relations. In *Computer Science Logic*, volume 3634, pages 309–324. Springer Verlag, 2005.

12 E. Kieroński. Decidability issues for two-variable logics with several linear orders. In *CSL-11*, pages 337–351, 2011.

13 E. Kieroński and J. Michaliszyn. Two-variable universal logic with transitive closure. In *CSL-12*, LIPIcs, pages 337–351, 2012.

14 E. Kieroński, J. Michaliszyn, I. Pratt-Hartmann, and L. Tendera. Two-variable first-order logic with equivalence closure. In *Proc. of LICS2012*, pages 431–440. IEEE, 2012.

15 E. Kieroński and M. Otto. Small substructures and decidability issues for first-order logic with two variables. In *Proc. of LICS2005*, pages 448–457, 2005.

16 E. Kieroński and M. Otto. Small substructures and decidability issues for first-order logic with two variables. *Journal of Symbolic Logic*, 77 (3):729–765, 2012.

17 E. Kieroński and L. Tendera. On finite satisfiability of two-variable first-order logic with equivalence relations. In *Proc. of LICS2009*, pages 123–132, 2009.

18 A. Manuel. Two variables and two successors. In Petr Hlinený and Antonín Kucera, editors, *MFCS*, volume 6281 of *Lecture Notes in Computer Science*, pages 513–524. Springer, 2010.

19 J. Michaliszyn. Decidability of the guarded fragment with the transitive closure. In *Proc. of ICALP2009*, pages 261–272, 2009.

20 M. Mortimer. On languages with two variables. *Zeitschr. f. Logik und Grundlagen d. Math.*, 21:135–140, 1975.

21 M. Niewerth and T. Schwentick. Two-variable logic and key constraints on data words. In Tova Milo, editor, *ICDT*, pages 138–149. ACM, 2011.

22 M. Otto. Two-variable first-order logic over ordered domains. *Journal of Symbolic Logic*, 66:685–702, 2001.

23 F. Ramsey. On a problem of formal logic. *Proc. London Math. Soc. series 2*, 30:264–286, 1930.

24 T. Schwentick and T. Zeume. Two-variable logic with two order relations - (extended abstract). In *CSL*, pages 499–513, 2010.

25 D. Scott. A decision method for validity of sentences in two variables. *J. Symb. Logic*, 27:477, 1962.

26 W. Szwast and L. Tendera. On the decision problem for the guarded fragment with transitivity. In *Proc. of LICS2001*, pages 147–156, 2001.

27 W. Szwast and L. Tendera. The guarded fragment with transitive guards. *Annals of Pure and Applied Logic*, 128:227–276, 2004.

Two-variable first order logic with modular predicates over words[*]

Luc Dartois[1] **and Charles Paperman**[1]

1 **LIAFA, Université Paris-Diderot and CNRS,**
Case 7014, 75205 Paris Cedex 13, France
luc.dartois,charles.paperman@liafa.univ-paris-diderot.fr

Abstract

We consider first order formulae over the signature consisting of the symbols of the alphabet, the symbol $<$ (interpreted as a linear order) and the set MOD of modular numerical predicates. We study the expressive power of $\mathbf{FO}^2[<, \mathrm{MOD}]$, the two-variable first order logic over this signature, interpreted over finite words. We give an algebraic characterization of the corresponding regular languages in terms of their syntactic morphisms and we also give simple unambiguous regular expressions for them. It follows that one can decide whether a given regular language is captured by $\mathbf{FO}^2[<, \mathrm{MOD}]$. Our proofs rely on a combination of arguments from semigroup theory (stamps), model theory (Ehrenfeucht-Fraïssé games) and combinatorics.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases First order logic, automata theory, semigroup, modular predicates

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.329

Following the pioneering work of Büchi [3], McNaughton and Papert [11] and Thomas [21], the study of the expressive power of fragments of first order logic has grown up to an important topic of automata theory. Part of the main results for finite words are summarized in the table below. They are concerned with the signature $[<]$ (the "sequential calculus" first considered by Büchi) and $[<, \mathrm{MOD}]$, where MOD stands for the set of modular predicates. The fragments of interest include $\boldsymbol{\Sigma}_1$, the set of existential formulae, its Boolean closure $\mathcal{B}\boldsymbol{\Sigma}_1$, the set $\mathbf{FO}$ of first order formulae and its restriction $\mathbf{FO}^2$ to two-variable formulae. As shown in the table below, all the corresponding fragments are already known to be decidable except for the class $\mathbf{FO}^2[<, \mathrm{MOD}]$, which is the topic of this paper.

	$\boldsymbol{\Sigma}_1$	$\mathcal{B}\boldsymbol{\Sigma}_1$	$\mathbf{FO}^2$	$\mathbf{FO}$
$[<]$	Decidable [12, 21]	Decidable [17, 21]	Decidable [20]	Decidable [11, 15]
$[<, \mathrm{MOD}]$	Decidable [4]	Decidable [4]	Decidable **New result**	Decidable [18, 2]

We also give an algebraic characterization of $\mathbf{FO}^2[<, \mathrm{MOD}]$ (Theorem 6), a description of the corresponding languages as unambiguous regular expressions (Proposition 31) and

* The authors are supported by the project ANR 2010 BLAN 0202 02 FREC, the second author is supported by Fondation CFM.

an equivalent definition in terms of a suitable variant of temporal logic (Proposition 30). Our algebraic characterization $\mathbf{QDA} = \mathbf{FO}^2[<, \mathrm{MOD}]$ can be viewed as an extension of two known results (a) $\mathbf{QA} = \mathbf{FO}[<, \mathrm{MOD}]$ proved in [2, 18] and (b) $\mathbf{DA} = \mathbf{FO}^2[<]$ proved in [5, 20]. However, it is not easy to extend the proofs of these equalities to our case. For instance, the proof of (a) makes use of the successor relation, which is not expressible in $\mathbf{FO}^2[<]$. Therefore our proof is closer to the proof of (b) but some technical difficulties still have to be worked out (See Section 5).

1 Preliminaries

1.1 Words and logic

Let A be a finite alphabet. We denote by A^* the set of all finite words over A and 1 the empty word. Given a word $u = a_0 \cdots a_{n-1}$ of length n, we denote by $\alpha(u)$ the set of letters of A occurring in u. We associate to u the *relational structure* $M_u = \{[0, n-1], \sigma\}$, where $[i, j]$ is the set of integers between i and j and σ is the truth table of the predicates over u. Basic examples of predicates are the binary predicate $<$, which is the usual order on integers, and $(\mathbf{a})_{a \in A}$ that are disjoint monadic predicates marking the positions of the letters over the structure. For instance, if $u = aabbab$, then $\mathbf{a} = \{0, 1, 4\}$ and $\mathbf{b} = \{2, 3, 5\}$. We also consider the modular predicate MOD_i^d, which holds at all positions equal to i modulo d, and the 0-ary predicate D_i^d which is true if the word has length equals to i modulo d. For $u = aabbab$, we have $MOD_0^2 = \{0, 2, 4\}$, and D_1^3 is *false* whereas D_0^3 is *true*. We denote by MOD the set of these *modular* predicates.

First order formulae are interpreted on words in the usual way (see [18]). For instance the formula $\exists x\ \exists y\ \exists z\ a(x) \wedge b(y) \wedge a(z) \wedge x < y \wedge y < z$ defines the language $A^*aA^*bA^*aA^*$.

In this article, we focus on the first order formulae containing only two different variables. The subsequent logic is denoted by $\mathbf{FO}^2[<]$. For instance the two-variable formula $\exists x\ \exists y\ a(x) \wedge b(y) \wedge x < y \wedge (\exists x \wedge a(x) \wedge y < x)$ also defines the language $A^*aA^*bA^*aA^*$ of the previous example. The first order logic with the order predicate can be enriched with modular predicates. We denote by $\mathbf{FO}[<, \mathrm{MOD}]$ (resp. $\mathbf{FO}^2[<, \mathrm{MOD}]$) the logic built with the same atomic propositions that $\mathbf{FO}[<]$ (resp. $\mathbf{FO}^2[<]$) except that we allow the modular predicates. For instance the formula $\exists x\ \exists y\ \exists z\ a(x) \wedge MOD_0^2(x) \wedge b(y) \wedge a(z) \wedge x < y \wedge y < z$ defines the language $(A^2)^*aA^*bA^*aA^*$.

Note that if required by context, we will specify the alphabet, denoting it between parentheses. For instance $\mathbf{FO}[<](B^*)$ denotes the set of the languages of B^* definable by a formula of $\mathbf{FO}[<]$.

1.2 Algebraic notions

We recall in this section the algebraic notions used in this paper.

1.2.1 Semigroups and recognizable languages

We refer to [13] for the standard definitions of semigroup theory. A *semigroup* is a set equipped with a binary associative operation, which we will denote multiplicatively. A *monoid* is a semigroup with a neutral element 1. Given a semigroup S, we denote by S^1 either S if S is already a monoid or the monoid obtained by adding a neutral element 1 to S otherwise. Recall that a monoid M *divides* another monoid N if M is a quotient of a submonoid of N. This defines a partial order on finite monoids.

A *stamp* is a surjective monoid morphism from A^* onto a finite monoid. A language L is *recognized* by a finite monoid M if there exists a stamp $\varphi : A^* \to M$ and a subset P of M such that $L = \varphi^{-1}(P)$. A language is *recognizable* if it is recognized by a finite monoid. Kleene's theorem states that the set of recognizable languages is exactly the set of rational (or regular) languages. The syntactic congruence of a regular language L of A^* is the equivalence relation $\equiv_L$ defined as follow:

$$u \equiv_L v \text{ if and only if for all } w, w' \in A^*, wuw' \in L \Leftrightarrow wvw' \in L.$$

The monoid $A^*/{\equiv_L}$ is the *syntactic monoid* of L and the morphism $\varphi : A^* \to A^*/{\equiv_L}$ is the *syntactic stamp*.

1.2.2 Stability index, stable semigroup, stable automaton

For a stamp $\varphi : A^* \to M$, the set $\varphi(A)$ is an element of the powerset monoid of M. As such it has an idempotent power. The *stability index* of a stamp is the least positive integer s such that $\varphi(A^s) = \varphi(A^{2s})$. This set is therefore a semigroup called the *stable semigroup* of φ. Stable semigroups are strongly related to *stable automata*, defined as follows. Let $\mathcal{A} = (Q, A, \cdot)$ be a deterministic automaton and let k be a positive integer. The *k-automaton* of $\mathcal{A}$ is the deterministic automaton $\mathcal{A}_k = (Q, A^k, \cdot^k)$ where $q \cdot^k (a_1 a_2 \cdots a_k) = (\cdots(q \cdot a_1) \cdot a_2) \cdots) \cdot a_k)$. Note that if M is the transition monoid of $\mathcal{A}$, and M_k the transition monoid of $\mathcal{A}_k$, then M_k is the submonoid of M generated by the image elements of words of length k in M.

▶ **Definition 1.** Let $\mathcal{A} = (Q, B, \cdot)$ be a deterministic automaton. We say that $\mathcal{A}$ is *stable* if for any two-letter word, there exists a letter that has the same action over the set Q, and conversely for any letter of B, there exists a word of B^2 that has the same action over Q.

As shown in the next proposition, this definition is a compatible translation of the stable semigroup for an automaton.

▶ **Proposition 2.** *Let $\mathcal{A}$ be a deterministic automaton. Then, there is an integer k such that the associated k-automaton is stable.*

The least k which satisfies this proposition is called the *stability index* of the automaton. It is equal to the stability index of the associated stamp.

1.2.3 Stamps and varieties

A (pseudo) variety of (finite) monoids is a class of monoids closed under division and finite products. According to Eilenberg [6], a *variety of languages* $\mathcal{V}$ is a class of languages closed under finite union, intersection and complementation, and closed under inverse of monoid morphism. This means that, for any monoid morphism $\varphi : A^* \to B^*$, $X \in \mathcal{V}(B^*)$ implies $\varphi^{-1}(X) \in \mathcal{V}(A^*)$. Furthermore Eilenberg [6] proved that there is a one-to-one correspondence between varieties of monoids and varieties of languages.
The class of languages $\mathbf{FO}^2[<, \mathrm{MOD}]$ is not closed under inverse morphisms, and the Eilenberg's varieties theory does not apply. Still, this class is closed under inverse of length-multiplying morphisms (shortened as *lm*-morphisms), and an algebraic characterization can be obtained by considering a more general framework : the theory of $\mathcal{C}$-varieties independently introduced by Esik and Ito [7] and Straubing [19] and developed by Pin and Straubing [14].

Let us now recall the notion of variety of stamps. A morphism $\alpha : A^* \to B^*$ is *length-multiplying* if there exists an integer n such that for any letter a of A, $\varphi(a)$ is a word of B^n. Given two stamps $\varphi : A^* \to M$ and $\psi : A^* \to N$, the product stamp is the stamp $\eta : A^* \to M \times N$ defined by $\eta(a) = (\varphi(a), \psi(a))$. A stamp $\varphi : A^* \to M$ lm-divides another stamp $\psi : B^* \to N$ if and only if there exists a pair (α, β) such that α is a lm-morphism from A^* to B^*, $\beta : N \to M$ is a partial onto monoid morphism and $\varphi = \beta \circ \psi \circ \alpha$. The couple (α, β) is called an lm-division.
Then a lm-variety of stamps is a class of stamps containing the trivial stamp and closed under lm-division and finite product. Note that if $\mathbf{V}$ is a variety of monoids, then the class of all stamps whose image is a monoid in $\mathbf{V}$ forms a lm-variety of stamps, also denoted $\mathbf{V}$. Moreover, given a lm-variety of stamps $\mathbf{V}$, the class $\mathcal{V}$ of all languages recognized by a stamp in $\mathbf{V}$ is a lm-variety of languages. The correspondence $\mathbf{V} \to \mathcal{V}$ is one-to-one and onto [19]. These notions are very useful to decide membership problems for regular languages. Let us recall a few examples.

▶ **Example 3.** A monoid M is *aperiodic* if there exists an integer n such that for any $x \in M$, $x^n = x^{n+1}$. It has been proved by Schützenberger [15] and McNaughton and Papert [11] that the class of aperiodic monoids forms a variety called $\mathbf{A}$ and the corresponding variety of languages is exactly the first-order definable languages, with the order and letter predicates.

▶ **Example 4.** Let $\mathbf{DA}$ be the variety of monoids satisfying the equation
$(xy)^\omega = (xy)^\omega x (xy)^\omega$ where ω is the idempotent power of the monoid. Alternatively $\mathbf{DA}$ is the variety of monoids whose regular $\mathcal{D}$-classes are aperiodic semigroups. The corresponding variety of languages $\mathcal{DA}$ is the class of $\mathbf{FO}^2[<]$-definable languages [20] or equivalently the unambiguous star-free languages [16].

▶ **Example 5.** Given a variety $\mathbf{V}$, the set of all stamps whose stable semigroup is in $\mathbf{V}$ forms a lm-variety of stamps denoted by $\mathbf{QV}$. A language L has its syntactic stamp in $\mathbf{QV}$ if and only if there is an automaton $\mathcal{A}$ recognizing L and a positive integer k such that the k-automaton of $\mathcal{A}$ has its transition monoid in $\mathbf{V}$. Straubing proved in [18] that a language is definable in $\mathbf{FO}[<, \mathrm{MOD}]$ if and only if its syntactic stamp belongs to the lm-variety of stamps $\mathbf{QA}$. We always denote by $\mathcal{QV}$ the lm-variety of languages associated to $\mathbf{QV}$.

2 Main result

Our main result extends the algebraic characterization of $\mathbf{FO}^2[<]$-definable languages by Thérien and Wilke [20] to $\mathbf{FO}^2[<, \mathrm{MOD}]$-definable languages. The next theorem states that the languages definable in $\mathbf{FO}^2[<, \mathrm{MOD}]$ are exactly the languages whose syntactic stamp is in $\mathbf{QDA}$.

▶ **Theorem 6.** $\mathbf{FO}^2[<, \mathrm{MOD}] = \mathbf{QDA}$

Given a regular language (given by a regular expression or by some finite automaton), one can effectively compute the stable semigroup of its syntactic stamp. Since membership in $\mathbf{DA}$ is decidable we get the following corollary.

▶ **Corollary 7.** *Given a regular language L, one can decide whether L is $\mathbf{FO}^2[<, \mathrm{MOD}]$-definable.*

In Section 3 we will give intuition of the power of the modular predicates. The first inclusion $\mathbf{FO}^2[<, \mathrm{MOD}] \subseteq \mathbf{QDA}$ will be proved in Section 4, using general arguments on automata and logic. The second inclusion is proved in Section 5, using Ehrenfeucht-Fraïssé games and algebraic tools. We will extend our main result to several other characterizations in Section 6.

3 $\mathrm{FO}^2[<]$ over an enriched alphabet

Given an integer $d > 1$, let us denote by $\mathbf{FO}^2[<, \mathrm{MOD}_d]$ the fragment of $\mathbf{FO}^2[<, \mathrm{MOD}]$ restricted to congruences modulo d. For a given language, this restriction does not lead to any loss of generality.

▶ **Lemma 8.** *Let L be a language of $\mathbf{FO}^2[<, \mathrm{MOD}]$. Then there exists an integer d such that L is in $\mathbf{FO}^2[<, \mathrm{MOD}_d]$.*

We now fix a positive integer d.

▶ **Definition 9** (Enriched alphabet)**.** Let A be an alphabet. We call the set $A_d = A \times (\mathbb{Z}/d\mathbb{Z})$ the *enriched alphabet* of A, and we denote by $\pi : A_d^* \to A^*$ the projection defined by $\pi(a, i) = a$ for each $(a, i) \in A_d$.

For example, the word $(a, 2)(b, 1)(b, 2)(a, 0)$ is an enriched word of $abba$ for $d = 3$. We say that $abba$ is the *underlying word* of $(a, 2)(b, 1)(b, 2)(a, 0)$.

▶ **Definition 10** (Well-formed words)**.** A word $(a_0, i_0)(a_1, i_1) \cdots (a_n, i_n)$ of A_d is *well-formed* if for $0 \leqslant j \leqslant n$, $i_j \equiv j \bmod d$. We denote by K the set of all well-formed words of A_d^*.

▶ **Definition 11.** For a word $u = a_0 a_1 \cdots a_n \in A^*$, the word $\overline{u} = (a_0, 0)(a_1, 1) \cdots (a_i, i \bmod d) \cdots (a_n, n \bmod d)$ is called the *well-formed word attached* to u.

▶ Remark. On well-formed structures, the projection π is a one-to-one application.

The enriched word $(a, 0)(b, 1)(b, 2)(a, 0)$ is a well-formed word for $d = 3$. Thanks to the previous remark, it is the unique well-formed word having the word $abba$ as underlying word.

▶ Remark. The operation $u \to \overline{u}$ is not a morphism. Indeed, if $|u| \not\equiv 0 \bmod d$ then $\overline{uv} \neq \overline{u}\,\overline{v}$. Thus we define the *$k$-shift operation*, denoted by $\overline{u}^k$, which maps the word $u = u_0 \cdots u_n$ to the enriched word $(u_0, k \bmod d)(u_1, k + 1 \bmod d) \cdots (u_n, n + k \bmod d)$. Note that, if $|u| \equiv k \bmod d$, then $\overline{uv} = \overline{u}\,\overline{v}^k$.

▶ **Proposition 12.** *Let d be a positive integer. Then*

$$\mathbf{FO}^2[<, \mathrm{MOD}_d](A^*) = \pi(\mathbf{FO}^2[<](A_d^*) \cap K).$$

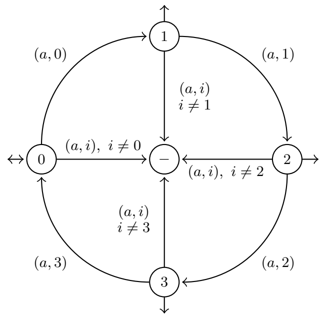
The proof relies on a syntactic transformation of the formulae. We replace MOD_i^d by a conjunction of enriched letters predicates. This can be done in the opposite direction as well, as we consider only well-formed words.

We recall (see [10]) that two words are separated by a formula of $\mathbf{FO}^2[<]$ with quantifier depth n if and only if Spoiler wins the n rounds Ehrenfeucht-Fraïssé game with two coloured pebbles. Thus one can state, in light of Proposition 12, the following assertion:

▶ **Proposition 13.** *Let u, v be words of A^*. Then there exists a formula of $\mathbf{FO}^2[<, \mathrm{MOD}_d]$ of quantifier depth n that separates them if, and only if, Spoiler wins the n rounds Ehrenfeucht-Fraïssé game for $\mathbf{FO}^2[<]$ over the well-formed pair $(\overline{u}, \overline{v})$.*

4 The inclusion $\mathrm{FO}^2[<, \mathrm{MOD}] \subseteq \mathbf{QDA}$

In this section, we prove one direction of the main theorem, using the enriched alphabet and the well-formed words. Let us first study the language K of well-formed words.



	0	1	2	3	$-$
$(a, 0)$	1	$-$	$-$	$-$	$-$
$(a, 1)$	$-$	2	$-$	$-$	$-$
$(a, 2)$	$-$	$-$	3	$-$	$-$
$(a, 3)$	$-$	$-$	$-$	0	$-$

Figure 1 Minimal automaton and transition monoid of K (for $d = 4$).

Consider the semigroup $B_d = (\mathbb{Z}/d\mathbb{Z} \times \mathbb{Z}/d\mathbb{Z}) \cup \{\perp\}$ where $\perp$ is a zero of B_d and for all (i, j) and (k, ℓ) in $\mathbb{Z}/d\mathbb{Z} \times \mathbb{Z}/d\mathbb{Z}$,

$$(i, j)(k, \ell) = \begin{cases} (i, \ell) & \text{if } j = k \\ \perp & \text{otherwise.} \end{cases}$$

The monoid B_d^1 is the transition monoid of the minimal automaton of K for $d \geqslant 2$. Let us denote by $\mathbf{J_1}$ the variety of idempotent and commutative monoids.

▶ **Proposition 14.** *The set of all well-formed words is recognized by a stamp in* $\mathbf{QJ_1}$.

▶ **Lemma 15.** *Let L be a language of $\mathcal{DA}(A_d^*)$. Then the language $L \cap K$ is in $\mathcal{QDA}(A_d^*)$.*

Proof. This comes from the fact that $L \in \mathcal{DA}(A_d^*) \subseteq \mathcal{QDA}(A_d^*)$, and $K \in \mathcal{QJ}_1(A_d^*) \subseteq \mathcal{QDA}(A_d^*)$. ◀

Now, we can use the previous result on well-formed words over modular predicates and prove the inclusion $\mathbf{FO}^2[<, \mathrm{MOD}] \subseteq \mathbf{QDA}$.

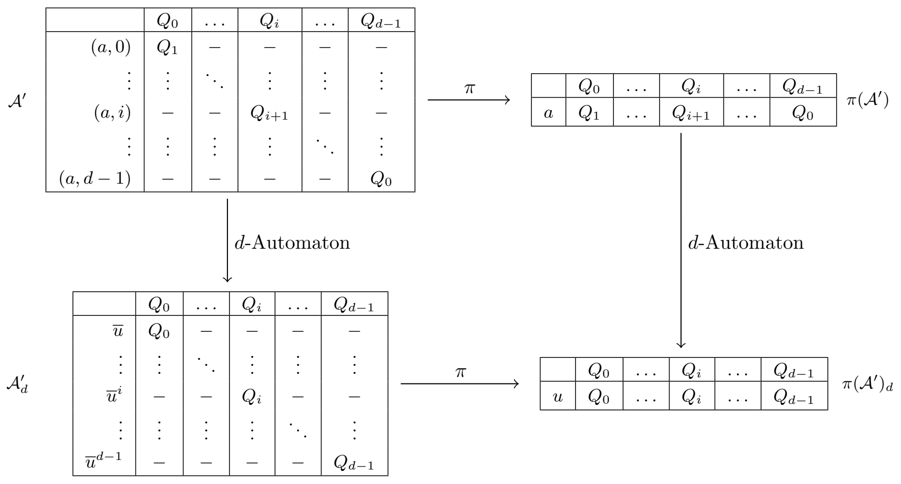
▶ **Theorem 16.** *The syntactic stamp of a $\mathbf{FO}^2[<, \mathrm{MOD}]$-definable language belongs to* $\mathbf{QDA}$.

As suggested by one the referees, this result can be proved by using Ehrenfeucht-Fraïssé games. The proof given below relies on finite automata and could easily be modified to recover the inclusion $\mathbf{FO}[<, \mathrm{MOD}] \subseteq \mathbf{QA}$ [18] and similar results for other fragments of logic.

Proof. Let L be a regular language definable in $\mathbf{FO}^2[<, \mathrm{MOD}](A^*)$. Then by Lemma 8, there exists an integer d such that L is defined in $\mathbf{FO}^2[<, \mathrm{MOD}_d](A^*)$. By Proposition 12, there exists a formula φ in $\mathbf{FO}^2[<](A_d^*)$ such that, $L = \pi(L')$ with $L' = L(\varphi) \cap K$. Since $\mathbf{FO}^2[<] = \mathbf{DA}$ (see [20]), and thanks to Lemma 15, the language L' is in $\mathcal{QDA}(A_d^*)$. Let $\mathcal{A}' = (Q, A_d, \cdot, i, F)$ be its minimal trim deterministic automaton. Since π is one-to-one, the automaton $\pi(\mathcal{A}')$, obtained by dropping the integer component on the transitions of $\mathcal{A}'$, recognizes L. As $\mathcal{A}'$ is trim and recognizes only well-formed words, the labels of all the outgoing edges from a given state have the same second component. For $0 \leqslant i < d$, let

$$Q_i = \{q \in Q \mid \text{ there exists } a \in A \text{ such that } q \cdot (a, i) \text{ is defined } \}$$

and let Q_E be the set of all states of fanout 0. Then Q is a disjoint union of the sets Q_i $(0 \leqslant i < d)$ and Q_E. Observing that a word of length k can only send a state of Q_i to a state of $Q_{i+k \bmod d} \cup Q_E$, the transition function of the d-automaton $\mathcal{A}'_d$ is a subset of $\bigcup\limits_{0\leqslant i<d} \big(Q_i \times A_d^d \times (Q_i \cup Q_E)\big)$. Then each set Q_i induces a monoid M_i, which is a submonoid of the transition monoid of $\mathcal{A}'_d$. Now, going back to the projected d-automaton $\pi(\mathcal{A}')_d$, one can see that the action of a word $u \in A^d$ on the set Q_i is the action of the word $(u_0, i)\cdots(u_d, i-1)$ on Q_i in the automaton $\mathcal{A}'_d$, described in M_i.



$\mathcal{A}'$
Q_0 … Q_i … Q_{d-1}
$(a,0)$ Q_1 – – – –
(a,i) – – Q_{i+1} – –
$(a,d-1)$ – – – – Q_0
π
Q_0 … Q_i … Q_{d-1}
a Q_1 … Q_{i+1} … Q_0
$\pi(\mathcal{A}')$
d-Automaton
d-Automaton
$\mathcal{A}'_d$
Q_0 … Q_i … Q_{d-1}
$\overline{u}$ Q_0 – – – –
$\overline{u}^i$ – – Q_i – –
$\overline{u}^{d-1}$ – – – – Q_{d-1}
π
Q_0 … Q_i … Q_{d-1}
u Q_0 … Q_i … Q_{d-1}
$\pi(\mathcal{A}')_d$

Figure 2 Transitions monoids.

Thus the full action of the word u over Q is described in each M_i, and hence the transition monoid of $\pi(\mathcal{A}')_d$ is a submonoid of the product monoid $\prod\limits_{i=0}^{d} M_i$ (full picture on Figure 2). Finally, as **DA** is a variety and $\mathcal{A}'_d$ has its transition monoid in **DA**, each submonoid M_i is also in **DA** and so is the transition monoid of $\pi(\mathcal{A}')_d$. We can conclude as L is recognized by an automaton whose d-automata has its transition monoid in **DA**. ◀

5 The inclusion QDA $\subseteq$ $\mathbf{FO}^2[<, \mathrm{MOD}]$

We now come to the second part of the proof of Theorem 6. We first enrich the congruences defined in [20] to take the modular predicates into account.

5.1 Congruence and syntactic operations over $\mathbf{FO}^2[<, \mathrm{MOD}]$

▶ **Definition 17.** Let $u \in A^*$ be a word, and let $a \in A$ be a letter of u. We call *left a-decomposition* of u the unique triple (u_0, a, u_1) such that $u = u_0 a u_1$ and u_0 does not contain any a. We define the *right decomposition* in a symmetrical way.

We recall the definition of the congruence $\equiv_n$ on A^* from [20].

▶ **Definition 18.** [20] Let $u, v \in A^*$ be words. Then we have $u \equiv_0 v$.
Moreover, $u \equiv_n v$ if and only if the following conditions hold:
1. $\alpha(u) = \alpha(v)$, the two words have the same alphabet,
2. For each a occurring in u, if (u_0, a, u_1) is the left a-decomposition of u and (v_0, a, v_1) that of v, then $u_0 \equiv_n v_0$ and $u_1 \equiv_{n-1} v_1$,
3. For each a occurring in u, if (u_0, a, u_1) is the right a-decomposition of u and (v_0, a, v_1) that of v, then $u_0 \equiv_{n-1} v_0$ and $u_1 \equiv_n v_1$.

The termination of these inductive definitions has to be verified. Let suppose that $u \equiv_n v$ for some words u and v and some positive integer n. Then, thanks to the first condition, the parameter $n+|\alpha(u)|$ is equal to $n+|\alpha(v)|$. For any left or right decomposition we decompose the words in two parts for which the parameter strictly decreases.

▶ **Proposition 19.** *[20] The relation $\equiv_n$is a congruence.*

This definition can be extended to the enriched alphabet and well-formed words as follows. We say that $u \equiv_n^d v$ if and only if $\overline{u} \equiv_n \overline{v}$.

▶ **Lemma 20.** *Let n, d be two positive integers, and u and v two words such that $u \equiv_n^d v$. Then the following statements hold:*

1. *if u is the empty word, then so is v,*
2. $|u| \equiv |v| \bmod d$,
3. *if $u = u_0au_1$, $v = v_0bv_1$ with $|u_0a| \equiv |v_0b| \bmod d$ and $|u_1| < d$, $|v_1| < d$, then $a = b$, $u_1 = v_1$ and $u_0 \equiv_{n-1}^d v_0$,*
4. *if $u = u_0au_1$, $v = v_0bv_1$ with $|u_0| < d$, $|v_0| < d$ and $|au_1| \equiv |bv_1| \bmod d$, then $a = b$, $u_0 = v_0$ and $u_1 \equiv_{n-1}^d v_1$,*
5. *for any word w, $uw \equiv_n^d vw$ and $wu \equiv_n^d wv$.*

▶ **Corollary 21.** *The relation $\equiv_n^d$ is a congruence on A^*.*

We will now connect our congruence to the logic $\mathbf{FO}^2[<, \mathrm{MOD}_d]$ through the Ehrenfeucht-Fraïssé games for $\mathbf{FO}^2[<](A_d^*)$ (cf. Proposition 13).

▶ **Theorem 22.** *Let $u, v \in A^*$ be words. If $u \not\equiv_n^d v$ then there is a formula of $\mathbf{FO}^2[<, \mathrm{MOD}_d]$ of quantifier depth at most $n + |\alpha(\overline{u})|$ that separates u from v.*

The proof makes use of Ehrenfeucht-Fraïssé games following the arguments of [20].

5.2 Congruence and algebraic operations over QDA

We now define a slightly modified version of the Green's preorders adapted to the stable semigroup. Let $h : A^* \to M$ be a stamp and let S be its stable semigroup. For any elements x and y in M let us write:

- $x \leqslant_{\mathcal{R}_{st}} y$ if and only if $xM \cap S \subseteq yM \cap S$
- $x \leqslant_{\mathcal{L}_{st}} y$ if and only if $Mx \cap S \subseteq My \cap S$
- $x \leqslant_{\mathcal{H}_{st}} y$ if and only if $x \leqslant_{\mathcal{R}_{st}} y$ and $x \leqslant_{\mathcal{L}_{st}} y$.

We also extend our definitions to modified versions of the Green's relations.

- $x \, \mathcal{R}_{st} \, y$ if and only if $x \leqslant_{\mathcal{R}_{st}} y$ and $y \leqslant_{\mathcal{R}_{st}} x$
- $x \, \mathcal{L}_{st} \, y$ if and only if $x \leqslant_{\mathcal{L}_{st}} y$ and $y \leqslant_{\mathcal{L}_{st}} x$
- $x \, \mathcal{H}_{st} \, y$ if and only if $x \leqslant_{\mathcal{H}_{st}} y$ and $y \leqslant_{\mathcal{H}_{st}} x$

We say that the stamp h is *length faithful* if $h^{-1}(S^1) = (A^d)^*$. This notion is shown to be necessary in the next lemma and does not involve a loss of generality, as shown in the proof of Corollary 29.

▶ **Lemma 23.** *Let $h : A^* \to M$ be a stamp and let S be its stable semigroup. If h is length faithful, then the restriction of $\leqslant_{\mathcal{R}_{st}}$ (resp. $\leqslant_{\mathcal{L}_{st}}$) to S is the usual Green relation $\leqslant_{\mathcal{R}}$ (resp. $\leqslant_{\mathcal{L}}$) over S.*

Proof. Let x be an element of S, and y an element of M such that xy is in S. Then, since h is length faithful, $h^{-1}(xy)$ is contained in $(A^d)^*$. Moreover, as x belongs to S, we also have $h^{-1}(x) \subseteq (A^d)^*$. Thus for any word u such that $h(u) = x$, and any word v such that $h(v) = y$, we have $|u| \equiv |uv| \equiv 0 \bmod d$, so $|v| \equiv 0 \bmod d$. Therefore y is an element of S.
This proves that for any x in S, $xM \cap S = xS$, and consequently for any x, y in S, $x \leqslant_{\mathcal{R}_{st}} y$ if and only if $x \leqslant_{\mathcal{R}} y$ in the Green relation over S.
The result for the $\leqslant_{\mathcal{L}_{st}}$ relation is obtained with a symmetric proof. ◀

▶ **Corollary 24.** *Let $h : A^* \to M$ be a length faithful stamp of* **QDA***. Then, the restriction of the $\mathcal{H}_{st}$-classes to S are trivial.*

We also define the $\mathcal{R}_{st}$-*decomposition* :

▶ **Definition 25.** Let u be a word and let $h : A^* \to M$ be a stamp. We call the $\mathcal{R}_{st}$-decomposition of u the tuple $(u_0, a_1, u_1, \ldots, a_s, u_s)$ such that $u = u_0 a_1 u_1 \cdots a_s u_s$ and:
1. $|u_0 a_1 u_1 \cdots a_i u_i| \equiv 0 \bmod d$ for all $0 \leqslant i < s$
2. $h(u_0 a_1 u_1 \cdots u_{i-1} a_i) >_{\mathcal{R}_{st}} h(u_0 \cdots u_i a_{i+1})$
3. For every prefix v of u_i of length multiple of d, $h(u_0 \cdots u_{i-1} a_i) \,\mathcal{R}_{st}\, h(u_0 \cdots a_i v)$
4. For every prefix v and v' of u_0 of length multiple of d, $h(v) \,\mathcal{R}_{st}\, h(v')$

The positions occurring in the $\mathcal{R}_{st}$-decomposition are the first positions multiple of d after falling in the $\leqslant_{\mathcal{R}_{st}}$-order. The two next lemmas will link our congruence $\equiv_d^n$ to the $\mathcal{R}_{st}$-decomposition of the lm-morphisms of **QDA**.

▶ **Lemma 26.** *Let $h : A^* \to M$ be a length faithful stamp in* **QDA***, let S be its stable semigroup. Let $u \in S$ and $a, x \in M$. If $ax \in S$, then $uax \,\mathcal{R}_{st}\, u$ implies $uaxa \,\mathcal{R}_{st}\, u$.*

Proof. The elements u and uax are $\mathcal{R}_{st}$-equivalent and h is length faithful. So thanks to Lemma 23 there is an element t of S such that $u = uaxt$. By iteration, we obtain $u = u(axt)^\omega$. But S belongs to **DA**, hence it satisfies the equation $(xy)^\omega x (xy)^\omega = (xy)^\omega$. Thus, $(axt)^\omega ax (axt)^\omega = (axt)^\omega$, then $u = u(axt)^\omega ax (axt)^\omega$. Shall we rewrite this last equation, we finally get $u = uaxa(xt(axt)^{\omega-1})$. And finally $u \in uaxaM \cap S$, proving that $u \,\mathcal{R}_{st}\, uaxa$. ◀

▶ **Corollary 27.** *Let $h : A^* \to M$ be a length faithful stamp in* **QDA** *and let u be a word. Then if $(u_0, a_1, u_1, \ldots, a_s, u_s)$ is the $\mathcal{R}_{st}$-decomposition of u then $(a_{i+1}, 0) \notin \alpha(\overline{a_i u_i})$ for $i < s$.*

Proof. Let $(u_0, a_1, u_1, \ldots, a_s, u_s)$ be the $\mathcal{R}_{st}$-decomposition of u. Suppose now that there exists i such that $(a_{i+1}, 0) \in \alpha(\overline{a_i u_i})$ for $i < s$. Then, thanks to the preceding Lemma, $h(a_i u_i a_{i+1}) \,\mathcal{R}_{st}\, h(a_i u_i)$ which is in contradiction with the definition of the $\mathcal{R}_{st}$-decomposition of u. ◀

We now have all the tools to prove the following theorem.

▶ **Theorem 28.** *Let $h : A^* \to M$ be a length faithful stamp of* **QDA** *and let d be its stability index. Then there exists an integer n such that for every words u and v, $u \equiv_n^d v$ implies $h(u) = h(v)$.*

Proof. Thanks to Lemma 20, if two words are equivalent for the congruence $\equiv_{n+1}^d$, then their suffixes of length smaller than d are equal and the associated prefixes are equivalent for the congruence $\equiv_n^d$. Therefore it is sufficient to prove the result for words of length multiple of d.

Let u and v be two words of length multiple of d, and an integer $n > |\alpha(\overline{u})||S|$ such that $u \equiv_n^d v$. Let us prove by induction on $|\alpha(\overline{u})|$ that $h(u) = h(v)$. If $|\alpha(\overline{u})| = 0$, then $u = v = 1$.

Consider the result to be true up to the rank $k-1$ and let u be such that $|\alpha(\overline{u})| = k$. We write $(u_0, a_1, u_1, \ldots, a_\ell, u_\ell)$ the $\mathcal{R}_{st}$-decomposition of u. One can remark that $\ell \leqslant |S|$, as each a_i makes the word go down in the $\mathcal{R}_{st}$-classes, whose number is bounded by the size of S. Using the preceding corollary, $(u_i, a_{i+1}, u_{i+1} \cdots u_\ell)$ is a left decomposition of $x_i = u_i \cdots u_\ell$ for $i < \ell$. As $u \equiv_n^d v$, there also exists a decomposition $(v_0, a_1, \ldots, a_\ell, v_\ell)$ of v such that $a_i u_i \equiv_{n-i}^d a_i v_i$ where $(a_{i+1}, 0) \notin \alpha(\overline{a_i u_i})$ and hence $|\alpha(\overline{a_i u_i})| \leqslant |\alpha(\overline{u})| - 1$. As $i < \ell$, we have $n - i \geqslant (k-1)|S| \geqslant |\alpha(\overline{a_i u_i})||S|$. Using the induction hypothesis, for $i < \ell$, $h(a_i u_i) = h(a_i v_i)$. And hence $h(u) \, \mathcal{R}_{st} \, h(u_1 \cdots a_\ell) = h(v_1 \cdots a_\ell) \geqslant_{\mathcal{R}_{st}} h(v)$. Symmetrically, we obtain that $h(v) \geqslant_{\mathcal{R}_{st}} h(u)$ and thus $h(u) \, \mathcal{R}_{st} \, h(v)$. Using the left/right symmetry, we also get that $h(v) \, \mathcal{L}_{st} \, h(u)$ and hence $h(v) \, \mathcal{H}_{st} \, h(u)$. By Corollary 24, the $\mathcal{H}_{st}$-classes are trivial in **QDA** over words of length multiple of d and hence $h(u) = h(v)$. ◀

▶ **Corollary 29.** $\mathbf{QDA} \subseteq \mathbf{FO}^2[<, \mathrm{MOD}]$

Proof. Let $\eta : A^* \to M$ be the syntactic stamp of L and S be the stable semigroup of η. Assume that η is in **QDA**. We claim that the morphism $h : A^* \to M \times \mathbb{Z}/d\mathbb{Z}$ defined, for all words u, by $h(u) = (\eta(u), |u| \bmod d)$ is length faithful. Indeed, the stable semigroup of h is equal to $S \times \{0\}$ and $h^{-1}(S \times \{0\}) = (A^d)^*$.

By Theorem 28, there exists an integer n such that the congruence $\equiv_n^d$ is thinner than the congruence induced by h which is itself thinner than the syntactic congruence of L. Therefore L is a finite union of $\equiv_n^d$- classes, each of them being, according to Theorem 22, definable by a formula of $\mathbf{FO}^2[<, \mathrm{MOD}_d]$ of quantifier-depth at most $n + |A|^d$. ◀

6 Other characterizations

Several other characterizations of **DA** are known (see [5] for a survey). For example, consider the fragment $\mathbf{TL}[X_a, Y_a]$ of the *linear temporal logic* defined inductively as follow:

$$\varphi \;\equiv\; \top \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid X_a\varphi \mid Y_a\varphi.$$

The unary operator X_a stands for ne**X**t a, and Y_a stands for **Y**esterday a. For a word u and one of its positions x, we have $(u, x) \models X_a\varphi$ if φ is true at the next a after x. We say that the word u satisfies $X_a\varphi$ if $(u, -1) \models X_a\varphi$. Symmetrically, we say that u satisfies $Y_a\varphi$ if $(u, |u|) \models Y_a\varphi$. It is a well known fact that the fragment $\mathbf{TL}[X_a, Y_a]$ has the same expressiveness power as the variety **DA**. Therefore, it is natural to look at $\mathbf{TL}[X_a^{r \bmod d}, Y_a^{r \bmod d}]$, where each predicate $X_a^{r \bmod d}$ is defined as follows. For a word u and one of its position x, we have $(u, x) \models X_a^{r \bmod d}\varphi$ if φ is true at the next a whose position is equal to r modulo d. As in Proposition 12 we can transfer a modular information from the predicates to the letters by changing the size of the alphabet.

▶ **Proposition 30.** *Let d be a non-zero integer. Then,*

$$\mathbf{TL}[X_a^{r \bmod d}, Y_a^{r \bmod d}](A^*) = \pi(\mathbf{TL}[X_{(a, r \bmod d)}, Y_{(a, r \bmod d)}](A_d^*) \cap K).$$

In [16], Schützenberger defined the *monomials* as the set of languages of the form $B_0^* a_1 B_1^* \cdots a_n B_n^*$, with $a_i \in A$ and $B_i \subseteq A$. A monomial L is said to be *unambiguous* if for every word u in L, there exists only one decomposition $u = u_0 a_1 u_1 \cdots a_n u_n$ with $\alpha(u_i) \subseteq B_i$. Finally, Schützenberger proved in [16] that a language is in **DA** if and only if it is a disjoint union of unambiguous monomials. We now give a similar definition adapted to the modular predicates. We define the *modular monomials* as the languages of the form

$$(A_0^0 \cdots A_{d-1}^0)^* a_1 (A_0^1 \cdots A_{d-1}^1)^* \cdots a_n (A_0^n \cdots A_{d-1}^n)^*$$

with d an integer, $A_k^i \subseteq A$ and $a_i \in A$.

▶ **Proposition 31.** *A language L is in $\mathcal{QDA}(A^*)$ if and only if L is a disjoint union of unambiguous modular monomials.*

Proof. We know by Theorem 6 and Proposition 12 that a language L is in $\mathcal{QDA}(A^*)$ if and only if there exists an integer d such that L is the projection of a set of well-formed words of a language L' in $\mathcal{DA}(A_d^*)$. Then L' is a disjoint union of unambiguous monomials. As the projection over well-formed words preserves disjoint union, it suffices to show that each unambiguous monomial projects into a disjoint union of modular monomials. Let $B_0^* b_1 B_1^* \cdots b_n B_n^*$ be an enriched unambiguous monomial with $b_i = (a_i, r_i)$. Then the projection of its well-formed words is the rational expression

$$(A_0^0 \cdots A_{d-1}^0)^* A_0^0 \cdots A_{r_1}^0 a_1 (A_{i+1}^1 \cdots A_i^1)^* A_{i+1}^1 \cdots A_{r_2}^1 a_2 \cdots$$

with $A_j^i = \{a \mid (a,j) \in B_i\}$, which can be rewritten as a disjoint union of unambiguous modular monomials. ◀

7 Conclusion

Our main results can now be summarized in a single statement, a consequence of Propositions 12, 30, 31 and Theorem 6.

▶ **Theorem 32.** *Let L be a regular language. Then, the following assertions are equivalent:*
- *L has its syntactic stamp in* **QDA**,
- *L is definable in* $\mathbf{FO}^2[<, \mathrm{MOD}]$,
- *L is definable in* $\mathbf{TL}[X_a^{r \bmod d}, Y_a^{r \bmod d}]$,
- *L is a disjoint union of unambiguous modular monomials.*

Our results are an instance of a more general problem: given a fragment **F** of **FO**, what is the expressive power of $\mathbf{F}[<, \mathrm{MOD}]$. In particular, if $\mathbf{F}[<]$ has an algebraic characterization, is there also a natural algebraic description of $\mathbf{F}[<, \mathrm{MOD}]$? Further if $\mathbf{F}[<]$ is decidable, does it imply that $\mathbf{F}[<, \mathrm{MOD}]$ is also decidable?
These questions are related to non-trivial questions of semigroup theory [1]. There is some hope that, for some sufficiently well-behaved fragment, $\mathbf{F}[<]$ corresponds to some variety of monoids **V** and that $\mathbf{F}[<, \mathrm{MOD}]$ corresponds to the semidirect product $\mathbf{V} * \mathbf{MOD}$ where **MOD** denotes the variety of all stamps onto a cyclic group. This is the case for instance for the fragment $\mathbf{\Sigma}_1$ and $\mathcal{B}\mathbf{\Sigma}_1$, as shown in [4]. The decidability of $\mathbf{V} * \mathbf{MOD}$ (given that of **V**) leads to another series of problems. When $\mathbf{V} * \mathbf{MOD}$ is equal to **QV** the decidability follows immediately but this is not always the case. For instance, $\mathcal{B}\mathbf{\Sigma}_1[<]$ corresponds to the variety **J** but $\mathcal{B}\mathbf{\Sigma}_1[<, \mathrm{MOD}]$ does not correspond to **QJ** and more sophisticated tools using derived categories have to be used [22]. Another possible route would be to follow a model theoretic approach as in [8, 9].
Acknowledgements We would like to thank the anonymous referees for very useful suggestions and Olivier Carton and Jean-Éric Pin for their helpful advice.

References

1 J. Almeida, Hyperdecidable pseudovarieties and the calculation of semidirect products, *Internat. J. Algebra Comput.* **9**,3-4 (1999), 241–261.

2 D. A. M. Barrington, K. Compton, H. Straubing and D. Thérien, Regular languages in NC^1, *J. Comput. System Sci.* **44**,3 (1992), 478–499.

3 J. R. Büchi, Weak second-order arithmetic and finite automata, *Z. Math. Logik Grundlagen Math.* **6** (1960), 66–92.

4 L. Chaubard, J.-É. Pin and H. Straubing, First order formulas with modular predicates, in *21st Annual IEEE Symposium on Logic in Computer Science (LICS 2006)*, pp. 211–220, IEEE, 2006.

5 V. Diekert, P. Gastin and M. Kufleitner, A survey on small fragments of first-order logic over finite words, *Internat. J. Found. Comput. Sci.* **19**,3 (2008), 513–548.

6 S. Eilenberg, *Automata, languages, and machines. Vol. B*, Academic Press [Harcourt Brace Jovanovich Publishers], New York, 1976. With two chapters by Bret Tilson, Pure and Applied Mathematics, Vol. 59.

7 Z. Ésik and M. Ito, Temporal logic with cyclic counting and the degree of aperiodicity of finite automata, *Acta Cybernet.* **16**,1 (2003), 1–28.

8 C. Glasser and H. Schmitz, The Boolean structure of dot-depth one, *J. Autom. Lang. Comb.* **6**,4 (2001), 437–452. 2nd Workshop on Descriptional Complexity of Automata, Grammars and Related Structures (London, ON, 2000).

9 C. Glasser, H. Schmitz and V. Selivanov, Efficient algorithms for membership in Boolean hierarchies of regular languages, in *STACS 2008*, pp. 337–348, *LIPIcs. Leibniz Int. Proc. Inform.* vol. 1, Schloss Dagstuhl. Leibniz-Zent. Inform., Wadern, 2008.

10 N. Immerman, Upper and lower bounds for first order expressibility, *J. Comput. System Sci.* **25**,1 (1982), 76–98.

11 R. McNaughton and S. Papert, *Counter-free automata*, The M.I.T. Press, Cambridge, Mass.-London, 1971.

12 D. Perrin and J.-É. Pin, First-order logic and star-free sets, *J. Comput. System Sci.* **32**,3 (1986), 393–406.

13 J.-É. Pin, Syntactic semigroups, in *Handbook of formal languages, Vol. 1*, pp. 679–746, Springer, Berlin, 1997.

14 J.-É. Pin and H. Straubing, Some results on $\mathcal{C}$-varieties, *Theor. Inform. Appl.* **39**,1 (2005), 239–262.

15 M. P. Schützenberger, On finite monoids having only trivial subgroups, *Information and Control* **8** (1965), 190–194.

16 M. P. Schützenberger, Sur le produit de concaténation non ambigu, *Semigroup Forum* **13**,1 (1976/77), 47–75.

17 I. Simon, Piecewise testable events, in *Automata theory and formal languages (Second GI Conf., Kaiserslautern, 1975)*, pp. 214–222., *Lect. Notes Comp. Sci.* vol. 33, Springer, Berlin, 1975.

18 H. Straubing, *Finite automata, formal logic, and circuit complexity*, Birkhäuser Boston Inc., Boston, MA, 1994.

19 H. Straubing, On logical descriptions of regular languages, in *LATIN 2002: Theoretical informatics*, pp. 528–538, *Lect. Notes Comp. Sci.* vol. 2286, Springer, Berlin, 2002.

20 D. Thérien and T. Wilke, Over words, two variables are as powerful as one quantifier alternation, in *STOC '98 (Dallas, TX)*, pp. 234–240, ACM, New York, 1999.

21 W. Thomas, Classifying regular events in symbolic logic, *J. Comput. System Sci.* **25**,3 (1982), 360–376.

22 B. Tilson, Categories as algebra: an essential ingredient in the theory of monoids, *J. Pure Appl. Algebra* **48**,1-2 (1987), 83–198.

Abusing the Tutte Matrix: An Algebraic Instance Compression for the K-set-cycle Problem

Magnus Wahlström[1]

1 Max-Planck-Institut für Informatik, Saarbrücken, Germany
wahl@mpi-inf.mpg.de

Abstract

We give an algebraic, determinant-based algorithm for the K-Cycle problem, i.e., the problem of finding a cycle through a set of specified elements. Our approach gives a simple FPT algorithm for the problem, matching the $\mathcal{O}^*(2^{|K|})$ running time of the algorithm of Björklund et al. (SODA, 2012). Furthermore, our approach is open for treatment by classical algebraic tools (e.g., Gaussian elimination), and we show that it leads to a *polynomial compression* of the problem, i.e., a polynomial-time reduction of the K-Cycle problem into an algebraic problem with coding size $\mathcal{O}(|K|^3)$. This is surprising, as several related problems (e.g., k-Cycle and the Disjoint Paths problem) are known not to admit such a reduction unless the polynomial hierarchy collapses. Furthermore, despite the result, we are not aware of any *witness* for the K-Cycle problem of size polynomial in $|K| + \log n$, which seems (for now) to separate the notions of polynomial compression and polynomial kernelization (as a polynomial kernelization for a problem in NP necessarily implies a small witness).

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Parameterized complexity; graph theory; kernelization; algebraic algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.341

1 Introduction

Parameterized complexity [19, 21] is one of the major approaches for dealing with NP-hard problems. In this setting, the input is associated with a *parameter* k, usually (but not exclusively) either a parameter related to the solution size, or a structural parameter such as treewidth; the fundamental assumption is that problems with a smaller parameter value will be easier than general instances. The critical notion is that of an *FPT* algorithm, which runs in time $f(k) \cdot \text{poly}(n)$ for some $f(k)$ where $\text{poly}(n)$ is independent of k, i.e., the combinatorial explosion is confined to the parameter k. This notion has lead to a large number of interesting algorithmic principles; for some surveys, see, e.g., the Festschrift of Mike Fellows [7].

One of the most vibrant parts of parameterized complexity in recent years is the subfield of *kernelization*. A kernelization is one of the basic approaches for creating FPT algorithms: It is an algorithm which runs in time polynomial in both k and n, which reduces the size of the instance (e.g., via reduction rules such as "remove a vertex shown not to be required by the solution") if the size is larger than some $f(k)$. Additionally, beyond being a design paradigm for FPT algorithms, it has been observed that the notion can be a good way to formalize effective *instance simplification*, e.g., preprocessing with a performance guarantee. A *polynomial kernel*, then, is a polynomial-time procedure which takes an input instance, with parameter k, and produces an output instance of size at most $\text{poly}(k)$, regardless of the value of n, without changing the problem status. Great interest has been taken in recent years in the question of which problems (and which problem parameterizations) admit polynomial kernels. This was sparked by the creation of a lower bounds framework by Bodlaender et al. [8] and Fortnow and Santhanam [23]. These results provided a way to exclude the existence of a polynomial kernel, under the hypothesis that the polynomial hierarchy does not collapse. Later refinements and applications of this framework can be found in, e.g., [17, 10, 16, 25, 20, 18, 14]. Significant progress has also been made on the positive side; for a few examples, see [9, 40, 22]. A recent trend, relevant to the current paper, is the application of *algebraic* tools to kernelization, e.g., [29, 30]. (See related work, below.)

Sometimes, the results found by these investigations can be quite surprising. As an example, consider the problems Vertex Cover (find a set of at most k vertices in a graph which covers all edges, i.e., a *vertex cover* of size at most k) and Connected Vertex Cover (find a vertex cover of size at most k which additionally is connected). The former is one of the most well-studied problems in theoretical computer science. In terms of parameterized complexity, it can be solved in time $\mathcal{O}^*(2^k)$ by a very simple algorithm, and in time $\mathcal{O}^*(1.28^k)$ by more involved means [12]. It has a simple 2-approximation, and a kernel of $2k$ vertices by the famous Nemhauser-Trotter theorem [36]. On the other hand, if the vertex cover is required to be connected, then the problem still has a simple greedy 2-approximation, an $\mathcal{O}^*(2^k)$-time FPT algorithm [15], and, as shown by Dom et al. [18], no polynomial kernel unless the polynomial hierarchy collapses.

As another example, consider the following three problems. Given a graph G, find (a) a cycle with at least k vertices (the k-Cycle problem); (b) a cycle passing through every element of a given set K, $|K| = k$ (the K-Cycle problem); or (c) a cycle passing through every element of K, which furthermore passes the elements in a specified order (which we may dub the Ordered K-Cycle problem). Which of these seem more or less general? Which, if any, seems most likely do admit efficient instance simplification?

Let us make a quick review of known FPT and kernelization results for these problems. All are NP-hard; in the first two problems, setting $k = n$ yields the Hamiltonian Cycle problem. The k-Cycle problem is closely related to k-Path (the problem of finding a path of length at least k), and there is by now a variety of interesting techniques that can be used to solve it in $2^{\mathcal{O}(k)}\text{poly}(n)$ time, from the seminal color-coding technique of Alon et al. [2], via the multilinear detection of Koutis [28] (see also Williams [43]), to the recent $\mathcal{O}^*(1.66^n)$-time Hamiltonian Cycle algorithm of Björklund [3], which was adapted to a parameterized setting in [5]. Ordered K-Cycle is equivalent to the well-known problem Disjoint Paths, where the input is k pairs of vertices (s_i, t_i), and the question is if we can connect all pairs with pairwise vertex-disjoint paths. This problem seems much more challenging. Robertson and Seymour, in the context of the graph minors programme, showed that it is FPT; more specifically, that it can be solved in time $\mathcal{O}(n^3)$ for every fixed k [37]. Kawarabayashi et al. improved this to $\mathcal{O}(n^2)$ for every fixed k [27]. However, the algorithms are in both cases very involved, and the dependency of the running time on k is hard to pin down exactly, but at the very least multiply exponential. As for polynomial kernelization, both are infeasible: k-Path and k-Cycle were among the first problems to which the lower bounds framework was applied, and Disjoint Paths was addressed in [11]. In both cases, the conclusion is that neither problem allows a polynomial kernelization (or even a polynomial-time compression into size poly(k)) unless the polynomial hierarchy collapses.

The K-Cycle problem, in turn, may intuitively seem to be closer in nature to the latter problem than the former – e.g., it is a terminal connectivity problem, parameterized by the number of terminals, and there is no obvious relation between the parameter and the size of the solution. Indeed, the problem can be solved via applications of the Disjoint Paths algorithm, and Kawarabayashi solved the problem in time $2^{2^{k^{10}}}$ poly(n) using graph minors-type graph structural reasoning [26]. However, recently, Björklund et al. [6] solved K-Cycle using an approach much closer to those of the cited k-Path algorithms: they define a large polynomial, which can be evaluated in $2^k \cdot$ poly(n) time, and which, when evaluated over a field of characteristic two, is non-zero if and only if the instance is positive. The result then follows from an application of the Schwartz-Zippel lemma. (In fact, they solved the more general variant of finding a *shortest* K-cycle.) For kernelization, the status of K-Cycle is so far unknown, but there are several factors – the lack of a small witness, the status of the related problems given above, the apparent difficulty of the problem – which would suggest that the answer should be negative (i.e., that K-Cycle should have no polynomial kernel). As the present paper shows, this conclusion may well be mistaken.

Our results. We give an alternative algebraic algorithm for the K-cycle problem, also with a running time of $\mathcal{O}^*(2^{O(k)})$, by encoding the problem into a variant of the Tutte matrix. More concretely, given G and K we construct a matrix M_G over GF(2^ℓ), whose entries are polynomials, and show that G has a K-cycle if and only if the determinant polynomial of M_G contains a certain type of term. Further minor modifications of the matrix yield an algorithm with running time $\mathcal{O}^*(2^k)$, and a matrix structure such that careful application of partial random evaluation and Gaussian elimination can reduce M_G to a matrix A with total coding length $\mathcal{O}(k^3)$, such that it can be decided from the determinant polynomial of A whether G has a K-cycle. All in all, this yields a randomized polynomial compression of K-Cycle into space $\mathcal{O}(k^3)$. The construction, and all proofs, are simple, and we need only basic arguments about determinants and cycle covers to complete them.

We note that our approach so far fails to provide a polynomial kernel, in the strict sense; the reason being that the output is an instance of a different problem (of deciding a particular property of $\det A$) which is not known to be in NP, while a kernelization requires that the output is an instance of the same problem. This is closely related to the issue of the *witness size* required for K-Cycle; we are not aware of a witness for either K-Cycle or for our artificial algebraic output problem, of size poly$(k + \log n)$. We consider these results quite surprising.

Related work. The Tutte matrix (see Section 2) is a skew-symmetric matrix of indeterminates, created from the adjacency matrix of a graph G, which is non-singular if and only if G has a perfect matching [41]. This can be used to determine the size of a maximum matching in randomized time $\mathcal{O}(n^\omega)$ [32, 34], where $\omega < 2.3727$ is the matrix multiplication exponent [42, 39, 13]. Mucha and Sankowski [34] showed how to find a maximum matching in the same time. Geelen [24] gave a deterministic polynomial-time procedure which finds a maximum rank evaluation of the Tutte matrix (which does not lead to a competitive deterministic matching algorithm, but may be of interest for the general question of removing randomness due to applications of Schwartz-Zippel). For non-algebraic algorithms for matching, Micali and Vazirani [33] gave an algorithm that finds a maximum matching in general graphs in time $\mathcal{O}(m\sqrt{n})$.

Algebraic FPT algorithms, beyond those cited for k-Path above, have been used by, e.g., Lokshtanov and Nederlof [31] and Cygan et al. [14]. See also Nederlof's PhD thesis [35]. More specifically, algorithms based around determinant computations have been used by Björklund [4, 3]. However, we argue that the approach of the present paper leads to significantly simpler algorithms and correctness proofs than before. Algebraically based *kernelizations*, in particular using tools of matroid theory, have been given in [29, 30].

Related to the present work, it is interesting to note that the result of [29] was a pure compression, albeit within NP (the problem Odd Cycle Transversal was encoded into matroid, represented by a matrix of total coding length poly(k)), while [30] gave graph-based reduction rules, significantly broadening the applicability of the tools. A similar improvement on the tools of the present work would be highly interesting.

Organization. We review some basic definitions in the next section, then Section 3 gives, in turn, a very simple $\mathcal{O}^*(4^k)$-time for K-Cycle; an improvement to an $\mathcal{O}^*(2^k)$-time algorithm; and the Gaussian polynomial compression.

2 Preliminaries

Parameterized complexity. A *parameterized problem* is a language $\mathcal{Q} \subseteq \Sigma^* \times \mathbb{N}$; the second component of instances (x, k) is called the parameter (cf. [19]). A parameterized problem is *fixed-parameter tractable* (FPT) if there is an algorithm A and a computable function $f \colon \mathbb{N} \to \mathbb{N}$ such that A decides $(x, k) \in \mathcal{Q}$ in time $f(k)|x|^{\mathcal{O}(1)}$. A *kernelization of* $\mathcal{Q}$ is a polynomial-time computable mapping $K \colon \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N} \colon (x, k) \mapsto (x', k')$ such that $(x, k) \in \mathcal{Q}$ if and only if $(x', k') \in \mathcal{Q}$ and with $|x'|, k' \leq h(k)$ where h is a computable function; h is called the *size* of the kernel and K is a *polynomial kernelization* if $h(k)$ is polynomially bounded. A *polynomial compression* is a polynomial kernelization relaxed so that the output may be an instance of a (fixed) different language than the input language. This has also been called *bikernel* [1] and *generalized kernelization* [8]).

The Tutte matrix. Let $G = (V, E)$ be a simple undirected graph with $V = \{v_1, \ldots, v_n\}$. The *Tutte matrix* A_G is the $n \times n$ matrix of indeterminates such that

$$A_G(i, j) = \begin{cases} x_{ij} & \text{if } v_i v_j \in E \text{ and } i < j, \\ -x_{ji} & \text{if } v_i v_j \in E \text{ and } i > j, \\ 0 & \text{otherwise,} \end{cases}$$

where x_{ij} are distinct commuting variables. Tutte [41] showed that $\det A_G \neq 0$ (viewed as a polynomial) if and only if G has a perfect matching. Lovasz [32] showed the applications of this type of result to randomized algorithms.

Determinants and cycle covers. We recall a few basic facts. Let $D = (V, E)$ be a directed graph, which may contain loops. A *cycle cover* of D is a set $\mathcal{C} \subseteq E$ of arcs such that every vertex in D has in- and out-degree exactly one in $\mathcal{C}$. We allow loops to be present in the cycle cover. For an undirected simple graph G, which again may contain loops, an *oriented cycle cover* of G is a cycle cover of the bidirectional graph corresponding to G. (Note that this implies that loops and isolated edges are permitted in the cycle cover, corresponding to cycles of length one respectively two.)

Let A be an $n \times n$ matrix over a field of characteristic two. Then the determinant and the permanent of A coincide:

$$\det A = \operatorname{perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A(i, \sigma(i)), \tag{1}$$

where S_n is the set of all permutations of $[n]$. Let D be a directed graph on vertex set $V = \{v_1, \ldots, v_n\}$ such that $v_i v_j \in E(D)$ if and only if $A(i, j) \neq 0$ (where $v_i v_i$ denotes a loop on the vertex v_i). There is a well-known bijection between terms of the naïve summation (1) of the determinant and cycle covers of D, as follows.

▶ **Proposition** 1. Let $D = (V, E)$ and A be as above. For a permutation $\sigma \in S_n$, let $\mathcal{C}_\sigma = \{v_i v_{\sigma(i)} : i \in [n]\}$. If $\mathcal{C}_\sigma \subseteq E$, then $\mathcal{C}_\sigma$ is a cycle cover of D; furthermore, this describes a bijection between cycle covers of D and non-zero terms in the summation (1).

Let C be a simple cycle in a cycle cover $\mathcal{C}_\sigma$. We call C *reversible* if the cycle has length at least three and for every edge $v_i v_j \in C$, we have $A(i, j) = A(j, i)$; further, we call $\mathcal{C}_\sigma$ reversible if it contains at least one reversible cycle. A critical observation, both in previous and present work, is that reversible cycle covers cancel in (1).

▶ **Proposition** 2. If (1) is computed over a field of characteristic two, then the terms corresponding to reversible cycle covers cancel each other.

Proof. Let $\mathcal{C}_\sigma$ be a reversible cycle cover, and let C be the first reversible cycle of $\mathcal{C}_\sigma$, counted by vertex incidence (i.e., the cycles of $\mathcal{C}_\sigma$ are sorted according to the number of the earliest incident vertex). Let $\mathcal{C}' = \mathcal{C}_{\sigma'}$ be the cycle cover resulting from reversing C. Then this operation creates a fix-point-free involution among the reversible cycle covers. Further, as the terms of (1) corresponding to σ and σ' are identical by definition, all terms of (1) corresponding to reversible cycle covers will cancel each other out. ◀

Thus, when reasoning about the surviving terms of $\det A$, we only need to concern ourselves with non-reversible cycle covers. (In particular, if $A = A_G$ is the Tutte matrix of a graph G over a field of characteristic two, then every cycle of length more than two is reversible, and there are no cycles of length one; thus the non-reversible cycle covers are exactly the perfect matchings of G.)

Schwartz-Zippel. We also recall the Schwartz-Zippel lemma.

▶ **Lemma 1** (Schwartz-Zippel [38, 44]). *Let $P(x_1, \ldots, x_n)$ be a multivariate polynomial of total degree at most d over a field $\mathbb{F}$, and assume that P is not identically zero. Pick $r_1, \ldots, r_n$ uniformly at random from $\mathbb{F}$. Then $\Pr(P(r_1, \ldots, r_n) = 0) \leq d/|F|$.*

We will use this mostly, though not exclusively, for the case that P is the determinant of a matrix over $\mathrm{GF}(2^\ell)$.

Detecting monomials in a polynomial. Our final generic ingredient is an application of inclusion-exclusion to finding certain monomials in a polynomial over a field of characteristic two. For a polynomial P and a monomial m, we let $P(m)$ denote the coefficient of m in P. We need a way to extract from P only those monomials divided by a certain term.

▶ **Lemma 2.** *Let $P(x_1, \ldots, x_n)$ be a polynomial over a field of characteristic two, and $T \subseteq [n]$ a set of target indices. For a set $I \subseteq [n]$, define $P_{-I}(x_1, \ldots, x_n) = P(y_1, \ldots, y_n)$ where $y_i = 0$ for $i \in I$ and $y_i = x_i$ otherwise. Define*

$$Q(x_1, \ldots, x_n) = \sum_{I \subseteq T} P_{-I}(x_1, \ldots, x_n).$$

Then for any monomial m such that $t := \prod_{i \in T} x_i$ divides m we have $Q(m) = P(m)$, and for every other monomial we have $Q(m) = 0$.

Proof. Consider a monomial m with non-zero coefficient in P. Observe first that for every $I \subseteq [n]$, we have $P_{-I}(m) = P(m)$ if no variable x_i with $i \in I$ occurs in m, and $P_{-I}(m) = 0$ otherwise. Now, if t divides m, then out of the $2^{|T|}$ evaluations, the monomial m occurs in exactly one (namely, $I = \emptyset$). Thus, $Q(m) = P(m)$. If t does not divide m, let $J = \{i \in I : x_i \text{ does not divide } m\}$, and observe that $P_{-I}(m) = P(m)$ for every $I \subseteq J$. Since $J \neq \emptyset$, this is an even number of occurrences of the same monomial with the same coefficient, which implies that they sum to zero. Applying this argument individually to every monomial in P accounts for all occurrences of monomials in the sum defining Q; the result follows. ◀

We remark that we do not require P to be multilinear (although we do require T to be a set rather than a multiset).

3 An Algebraic FPT Algorithm

We now give our algorithm and compression for K-Cycle. Let us first fix a definition.

▶ **Definition 3.** For a vertex $v \in V$, a v-cycle is a cycle that passes through v. For a set $T \subseteq V$, a T-cycle is a cycle that passes through all vertices of T. In both cases, the cycle may pass through further vertices, but this is not required.

The problem is then formally defined as follows.

K-Cycle
Input: A graph $G = (V, E)$; a set $K \subseteq V$ of terminal vertices
Parameter: $k := |K|$
Question: Is there a K-cycle in G?

We will show two algebraic FPT algorithms for this problem, giving two ways of encoding it into the determinant of a matrix. We then show how this implies a polynomial compression via Gaussian elimination, into space $\mathcal{O}(k^3)$.

3.1 Graph preprocessing

We begin with a simple preprocessing of the graph (reducing the terminals to degree two).

▶ **Lemma 4.** *Let (G, K) be an instance of K-Cycle with $|K| > 1$. We can reduce (G, K) to an equivalent instance (G', K'), $|K'| = |K|$, where $d(v) = 2$ for every $v \in K'$, and where K' is an independent set with no common neighbours.*

Proof. We assume that K is an independent set in G (by subdividing edges within K, if necessary). Construct G' from G by replacing every terminal $v \in K$ by two non-adjacent copies v', v'' (with neighbourhoods identical to that of v). Create a new vertex v with $N(v) = \{v', v''\}$. The new terminal set K' consists of these new vertices v.

It is easy to show that this reduction maintains the solution status. On the one hand, for any K-cycle in G, we may replace each portion $u - v - w$ of the cycle, with $v \in K$ and hence $u, w \notin K$, by a path $u - v' - v - v'' - w$, hitting the new terminal v. On the other hand, any K'-cycle in G' must pass through both neighbours v', v'' of each terminal $v \in K'$, and these neighbours are distinct for all terminals. Thus if each segment $v' - v - v''$ of the K'-cycle in G' is contracted into v, we get a valid K-cycle in G. ◀

The requirement that $|K| > 1$ comes from the consideration of whether a single edge uv should be considered a K-cycle with $K = \{u\}$ (but the case $|K| = 1$ is in either case easily solvable in polynomial time).

For the rest of the paper, for convenience, we will let $G = (V, E)$ be a graph, on vertex set $V = \{v_1, \ldots, v_n\}$ and terminal set $K = \{v_1, \ldots, v_k\}$, to which the above reduction has already been applied. We also assume $N(v_i) = \{v_{k+2i-1}, v_{k+2i}\}$ for $i \in [k]$.

3.2 Matrix construction

We now show the matrix which will encode the existence of a K-cycle. We begin with a more intuitive construction, that implies a running time of $\mathcal{O}^*(4^k)$, then modify it to arrive at the $\mathcal{O}^*(2^k)$-time algorithm and polynomial compression.

Given a graph G, reduced as per the previous subsection, we define the matrix A_G as follows. We start from the Tutte matrix A_G of G (although, as the field is of characteristic two, we will not observe the signs), and adjust so that $A(i,i) = 1$ for $i > 3k$ (effectively adding self-loops to all vertices except $N[K]$). Finally, we orient the edges incident to v_1 to make v_1-cycles non-reversible: let $A(1, k+1)$ and $A(k+2, 1)$ be unmodified, but set $A(1, k+2) = A(k+1, 1) = 0$. This can be done safely, as any K-cycle of G can be oriented in either direction.

Let M_G denote the resulting matrix. We can detect a K-cycle in G as follows.

▶ **Theorem 5.** *Let $T = \{x_{i,k+2i-1}, x_{i,k+2i} : i \in [k]\}$ and $t = \prod_{x \in T} x$. Then G has a K-cycle if and only if $\det M_G$, viewed as a polynomial, contains a monomial m with non-zero coefficient such that t divides m.*

Proof. Recall the summation (1) and the notion of a reversible cycle from Section 2. We claim a one-to-one correspondence between non-zero monomials of $\det M_G$ and non-reversible cycle covers.

This follows from basic observations, but we prove it for completeness. Since (1) is already in sum-product form, every non-zero monomial of $\det M_G$ corresponds to a non-empty set of summands from (1). By Prop. 2, we get the same result if we restrict ourselves to those summands corresponding to non-reversible cycle covers. We show that two summands, corresponding to distinct non-reversible cycle covers $\mathcal{C}, \mathcal{C}'$, always produce distinct monomials: if $\mathcal{C}$ and $\mathcal{C}'$ use distinct sets of underlying undirected, non-loop edges, then the claim is clear, and the set of loop edges of a cycle cover is a function of the set of non-loop edges. In the remaining case, $\mathcal{C}'$ must be attainable by a reorientation of $\mathcal{C}$. However, there are by construction only three types of non-reversible cycles: loops, isolated edges, and v_1-cycles, where a v_1-cycle cannot be reversed, and loops and isolated cycles are invariant under reversal. Thus $\mathcal{C}$ and $\mathcal{C}'$ must produce distinct monomials.

The result is now simple. First, if C is a K-cycle in G, then it contributes all factors in t, and by padding C using self-loops we produce a non-reversible cycle cover, which produces a non-zero monomial of $\det M_G$. On the other hand, if a non-zero monomial in $\det M_G$ contains the factor $x_{i,k+2i-1}x_{i,k+2i}$, then in the corresponding cycle cover, the v_1-cycle most also pass through v_i, as such a factor cannot be contributed by loops and isolated edges. By induction, if a non-zero monomial in $\det M_G$ is divided by t, then the corresponding cycle cover contains a v_1-cycle which passes through every vertex of K, i.e., a K-cycle. ◀

As $|T| = 2k$, this implies an $\mathcal{O}(2^{2k})$-time randomized algorithm for the problem, via Lemma 2 and by evaluating the resulting polynomial Q randomly over $\mathrm{GF}(2^\ell)$ for $\ell = \Omega(\log n)$. We will improve this in two ways: by introducing a modification which will let us match the $\mathcal{O}^*(2^k)$ running time of [6], and by showing how to use Gaussian elimination and partial random evaluation to produce a polynomial compression.

3.3 A 2^k Algorithm

We now show a different way to determine the existence of a K-cycle from M_G. Let an *orientation of* M_G be the result of, for every $v_i \in K$, $i > 1$, either setting $A(k+2i-1, i) = A(i, k+2i) = 0$ or $A(k+2i, i) = A(i, k+2i-1) = 0$, i.e., orienting the edges incident to v_i either as $v_i' \to v_i \to v_i''$ or as $v_i' \leftarrow v_i \leftarrow v_i''$. We claim the following.

▶ **Theorem 6.** *Let Q' be the sum of* $\det M_G'$ *over all 2^{k-1} orientations M_G' of M_G. Then G has a K-cycle if and only if Q' is not identically zero.*

Proof. Let M_G' be an arbitrary orientation of M_G. As in Theorem 5, monomials of $\det M_G'$ correspond to non-reversible cycle covers, but now, every cycle incident on some $v_i \in K$ counts as non-reversible (and again, attempting to reverse such a cycle produces a zero-term). On the other hand, if two orientations M_G' and M_G'' contain non-reversible cycle covers $\mathcal{C}'$ and $\mathcal{C}''$ such that $\mathcal{C}''$ can be obtained by reorienting $\mathcal{C}'$, then $\mathcal{C}'$ and $\mathcal{C}''$ contribute identical monomials to the sum, and their contribution may cancel. Thus, let $\mathcal{C}^*$ be an *unoriented* cycle cover, such that every cycle in $\mathcal{C}^*$ is either a loop, an isolated edge, or a cycle incident on K, and such that K is covered entirely by the latter type of cycles. We will count the number of contributions of orientations of $\mathcal{C}^*$ to the sum.

For this, simply observe that in a single cycle C of $\mathcal{C}^*$, as soon as the orientation of at least one vertex of C has been determined, the direction taken through every other vertex of C is fixed as a consequence. Thus, if $\mathcal{C}^*$ contains a K-cycle C, then only one orientation M_G' is possible, as the vertex v_1 enforces a direction already in M_G. On the other hand, if $\mathcal{C}^*$ contains at least two cycles incident on K, then all cycles *not* incident on v_1 may be oriented arbitrarily, making for an even number of orientations, each one of which contributes the same monomial to the sum.

Thus non-zero monomials of Q' correspond to K-cycles in G, as promised. ◀

This construction brings our algorithm closer in spirit to the determinant sums of Björklund [4, 3], or the algebraic FPT algorithms of Cygan et al. [15]. However, as the next subsection shows, by bringing the algorithm back into the structure of deciding properties of the determinant polynomial of a single matrix, we get a randomized polynomial compression for K-Cycle via Gaussian elimination.

3.4 Polynomial Compression

Now, we finally show how to use the above for a polynomial compression of the K-Cycle problem.

We describe one final modification of the matrix M_G. For every $v_i \in K$, $i > 1$, we introduce a new variable a_i, and multiply $A(k+2i-1, i)$ and $A(i, k+2i)$ by a_i, and $A(k+2i, i)$ and $A(i, k+2i-1)$ by $1 - a_i$. Observe that this implies that the algorithm of Theorem 6 can be executed by iteratively setting each a_i to either 1 or 0, and computing the determinant each time. Strictly speaking, each individual determinant computation would then seem to require a fresh dose of randomness, via the Schwartz-Zippel evaluation step, making the approach inappropriate for kernelization. We show that it is possible to perform this in the alternate direction, first randomly evaluating every variable x_e for $e \in E(G)$, then performing Gaussian elimination into a compressed output, and finally (at some future time) performing the 2^{k-1} assignments to the variables a_i and computing the resulting determinants.

▶ **Theorem 7.** *The K-*Cycle* problem has a randomized polynomial compression of size $\mathcal{O}(k^3)$.*

Proof. We get the result in two steps, first showing that we can randomly evaluate the variables $\mathbf{x}$ while leaving $\mathbf{a}$ as indeterminates, then applying Gaussian elimination to produce a smaller matrix with the same determinant (viewed as a polynomial in $\mathbf{a}$).

Let $P(\mathbf{x}, \mathbf{a})$ be the determinant polynomial of M_G. Define $Q(\mathbf{x})$ to be the sum over the 2^{k-1} instantiations of $\mathbf{a}$ necessary to emulate the algorithm of Theorem 6; observe that $Q(\mathbf{x})$ is a polynomial of degree n, and that $Q(\mathbf{x})$ is identically zero if and only if G has no K-cycle. Thus, again by Schwartz-Zippel, we may instantiate $\mathbf{x}$ randomly from $\mathrm{GF}(2^\ell)$, and with probability at least $n/2^\ell$ the resulting values are such that the 2^{k-1}-sized evaluation of $Q(\mathbf{x})$ would return non-zero. Picking $\ell = \Theta(\log n)$ is sufficient for this step to succeed with polynomial probability in n (and with $\ell = \Theta(\log n + k)$, we get a failure rate still polynomial in n, but exponentially small in k). Note that by standard observations we may assume $k \geq \log n$, as otherwise the $\mathcal{O}^*(2^k)$-algorithm runs in polynomial time.

Now, observe that we only need to know the existence of the polynomial Q for the above correctness argument. Thus, by replacing $\mathbf{x}$ randomly by values from $\mathrm{GF}(2^\ell)$, we get a matrix M' with mostly concrete values, and indeterminates in the top-left $3k \times 3k$ corner, such that preserving $\det M'$ is sufficient (up to the failure probability in the previous step) for preserving the information of whether G has a K-cycle.

Next, recall that row and column operations preserve the determinant of a matrix exactly. We show that we can reduce M' to a blocks form

$$M' = \begin{pmatrix} A & 0 \\ 0 & C \end{pmatrix},$$

where C is a matrix without indeterminates. Thus we will have $\det M' = (\det A)(\det C)$ where $\det C$ is a constant.

This is easy. For sets $R, C \subseteq [n]$, let $M[R, C]$ denote the induced submatrix of M with rows R and columns C. Observe that the submatrix $M_G[[3k+1, n], [3k+1, n]]$ is non-singular, as the diagonal contributes the term 1 to the determinant and every other term will contain at least one indeterminate. Thus (up to the failure probability), $M'[[3k+1, n], [3k+1, n]]$ is non-singular, and can be reduced to diagonal form with a non-zero diagonal, without introducing any new indeterminate entries in M'. Now we can use further row and column operations to reduce $M'[[1, 3k], [3k+1, n]]$ and $M'[[3k+1, n], [1, 3k]]$ to all-zero matrices (thereby modifying the contents of $M'[[1, 3k], [1, 3k]]$, but not $M'[[3k+1, n], [3k+1, n]]$). This creates the desired blocks form, and every step preserves the determinant precisely and is performed without further failure probability or growth of the individual entries (since we are working over a finite field).

Finally, we consider the resulting contents of the matrix A. Initially, the entries of $M'(i, j)$ for $i, j \leq 3k$ are either constants, or expressions $a_i \cdot c + c'$ for some constants c, c'. Every further row or column operation that modifies these entries adds some concrete value c' to the entry, meaning that we can maintain these entries in the form $a_i \cdot c + c'$ where c, c' are concrete values from $\mathrm{GF}(2^\ell)$; thus the coding length remains $\mathcal{O}(\ell)$ bits per entry. We then multiply one arbitrary row of A by $\det C$, which again only has the effect of modifying the values c, c'. This gives us a $3k \times 3k$ matrix A', with entries encoded into $\mathcal{O}(\ell) = \mathcal{O}(k)$ bits, such that $\det A' = \det M'$, where M' is the matrix produced by randomly instantiating $\mathbf{x}$ in M_G. ◀

Finally, we remark that, unusually, the output problem is not trivially in NP (as it is a question about the outcome of an exponentially large computation). Thus in terms of parameterized complexity, we do not strictly speaking get a polynomial kernel, as we know of no way of getting back from the matrix A' above to an instance of K-Cycle.

4 Conclusions

We have shown an alternate algebraic algorithm for the K-Cycle problem, recasting the original problem into a question about the existence of certain terms in the determinant polynomial of a matrix with indeterminate entries. By careful application of partial evaluation and Gaussian elimination, we have shown that this leads to a polynomial compression of a K-Cycle instance into space $\mathcal{O}(|K|^3)$. This partially answers the question of the kernelizability of K-Cycle, in a perhaps surprising direction.

Although we are not able to produce a proper kernel, since we are not able to get back to an instance of the K-Cycle problem, such kernel-like polynomial compressions have been previously considered in parameterized complexity [1], and in fact all existing frameworks for excluding polynomial kernelization (e.g., [23, 20]) also exclude polynomial compressions. Thus, for the sake of a smooth theory, we hope that the K-Cycle problem can also be shown to have a polynomial kernel (e.g., a compression within NP).

Another interesting improvement would be a more direct kernel, e.g., based on reduction rules which make direct modifications to G and K. The tools required for finding such rules may well have further applications (perhaps analogously to the two previous works [29, 30]).

It would also be interesting to consider further related problems, perhaps starting with the problems of finding a *shortest* K-cycle, and a K-cycle with a prescribed parity, as these problems can also be solved by the approach in [6]. While it seems that our algorithm can be adapted for this setting, it is not clear to us at the moment whether this can be done in a way that allows for a polynomial compression.

Acknowledgements The author is grateful to Thore Husfeldt and Stefan Kratsch for rewarding discussions, and to an anonymous reviewer for suggesting improvements to the paper.

References

1 N. Alon, G. Gutin, E. Kim, S. Szeider, and A. Yeo. Solving MAX-r-SAT above a tight lower bound. *Algorithmica*, pages 1–18, 2010.
2 N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.
3 A. Björklund. Determinant sums for undirected hamiltonicity. In *FOCS*, pages 173–182, 2010.
4 A. Björklund. Exact covers via determinants. In *STACS*, pages 95–106, 2010.
5 A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Narrow sieves for parameterized paths and packings. *CoRR*, arXiv:1007.1161, 2010.
6 A. Björklund, T. Husfeldt, and N. Taslaman. Shortest cycle through specified elements. In *SODA*, pages 1747–1753, 2012.
7 H. L. Bodlaender, R. Downey, F. V. Fomin, and D. Marx, editors. *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, volume 7370 of *Lecture Notes in Computer Science*. Springer, 2012.
8 H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.
9 H. L. Bodlaender, F. V. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh, and D. M. Thilikos. (Meta) kernelization. In *FOCS*, pages 629–638, 2009.

10 H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch. Cross-composition: A new technique for kernelization lower bounds. In *STACS*, pages 165–176, 2011.
11 H. L. Bodlaender, S. Thomassé, and A. Yeo. Kernel bounds for disjoint cycles and disjoint paths. *Theor. Comput. Sci.*, 412(35):4570–4578, 2011.
12 J. Chen, I. A. Kanj, and G. Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010.
13 D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.*, 9(3):251–280, 1990.
14 M. Cygan, S. Kratsch, M. Pilipczuk, M. Pilipczuk, and M. Wahlström. Clique cover and graph separation: New incompressibility results. In *ICALP (1)*, pages 254–265, 2012.
15 M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J. M. M. van Rooij, and J. O. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *FOCS*, pages 150–159, 2011.
16 H. Dell and D. Marx. Kernelization of packing problems. In *SODA*, pages 68–81, 2012.
17 H. Dell and D. van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In *STOC*, pages 251–260, 2010.
18 M. Dom, D. Lokshtanov, and S. Saurabh. Incompressibility through colors and IDs. In *ICALP (1)*, pages 378–389, 2009.
19 R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, November 1998.
20 A. Drucker. New limits to classical and quantum instance compression. In *FOCS*, pages 609–618, 2012.
21 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, March 2006.
22 F. V. Fomin, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Bidimensionality and kernels. In M. Charikar, editor, *SODA*, pages 503–510. SIAM, 2010.
23 L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011.
24 J. F. Geelen. An algebraic matching algorithm. *Combinatorica*, 20(1):61–70, 2000.
25 D. Hermelin and X. Wu. Weak compositions and their applications to polynomial lower bounds for kernelization. In *SODA*, pages 104–113, 2012.
26 K. Kawarabayashi. An improved algorithm for finding cycles through elements. In *IPCO*, pages 374–384, 2008.
27 K. Kawarabayashi, Y. Kobayashi, and B. A. Reed. The disjoint paths problem in quadratic time. *J. Comb. Theory, Ser. B*, 102(2):424–435, 2012.
28 I. Koutis. Faster algebraic algorithms for path and packing problems. In *ICALP (1)*, pages 575–586, 2008.
29 S. Kratsch and M. Wahlström. Compression via matroids: a randomized polynomial kernel for odd cycle transversal. In *SODA*, pages 94–103, 2012.
30 S. Kratsch and M. Wahlström. Representative sets and irrelevant vertices: new tools for kernelization. In *FOCS*, pages 450–459, 2012.
31 D. Lokshtanov and J. Nederlof. Saving space by algebraization. In *STOC*, pages 321–330, 2010.
32 L. Lovász. On determinants, matchings, and random algorithms. In *FCT*, pages 565–574, 1979.
33 S. Micali and V. V. Vazirani. An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs. In *FOCS*, pages 17–27, 1980.
34 M. Mucha and P. Sankowski. Maximum matchings via Gaussian elimination. In *FOCS*, pages 248–255, 2004.
35 J. Nederlof. *Space and Time Efficient Structural Improvements of Dynamic Programming Algorithms*. PhD thesis, University of Bergen, Norway, 2011. Available at `http://folk.uib.no/jne061/PhDthesisJesper.pdf`.

36 G. Nemhauser and L. Trotter. Vertex packing: structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975.

37 N. Robertson and P. D. Seymour. Graph minors. XIII. The disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.

38 J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

39 A. Stothers. *On the Complexity of Matrix Multiplication.* PhD thesis, University of Edinburgh, 2010.

40 S. Thomassé. A $4k^2$ kernel for feedback vertex set. *ACM Transactions on Algorithms*, 6(2), 2010.

41 W. T. Tutte. The factorization of linear graphs. *J. London Math. Soc.*, s1-22(2):107–111, 1947.

42 V. Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. In *STOC*, pages 887–898, 2012.

43 R. Williams. Finding paths of length k in $O^*(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, 2009.

44 R. E. Zippel. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation (EUROSAM)*, pages 216–226, 1979.

Subexponential-Time Parameterized Algorithm for Steiner Tree on Planar Graphs*

Marcin Pilipczuk[1], Michał Pilipczuk[2], Piotr Sankowski[3], and Erik Jan van Leeuwen[4]

1,3 Institute of Informatics, University of Warsaw, Poland
`{malcin,sank}@mimuw.edu.pl`

2 Department of Informatics, University of Bergen, Norway
`michal.pilipczuk@ii.uib.no`

4 Department of Computer, Control, and Management Engineering, Sapienza University of Rome, Italy
`E.J.van.Leeuwen@dis.uniroma1.it`

Abstract

The well-known bidimensionality theory provides a method for designing fast, subexponential-time parameterized algorithms for a vast number of NP-hard problems on sparse graph classes such as planar graphs, bounded genus graphs, or, more generally, graphs with a fixed excluded minor. However, in order to apply the bidimensionality framework the considered problem needs to fulfill a special density property. Some well-known problems do not have this property, unfortunately, with probably the most prominent and important example being the Steiner Tree problem. Hence the question whether a subexponential-time parameterized algorithm for Steiner Tree on planar graphs exists has remained open. In this paper, we answer this question positively and develop an algorithm running in $\mathcal{O}(2^{\mathcal{O}((k \log k)^{2/3})} n)$ time and polynomial space, where k is the size of the Steiner tree and n is the number of vertices of the graph. Our algorithm does not rely on tools from bidimensionality theory or graph minors theory, apart from Baker's classical approach. Instead, we introduce new tools and concepts to the study of the parameterized complexity of problems on sparse graphs.

1998 ACM Subject Classification G.2.2 Graphs Algorithms

Keywords and phrases planar graph, Steiner tree, subexponential-time algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.353

1 Introduction

It is widely believed that no NP-hard problem can be solved in polynomial time. Interestingly, it turns out that NP-hard problems differ greatly in their exact exponential-time complexity. A majority of NP-complete problems admits algorithms that run in single-exponential time with respect to the natural parameters of the problem (see Section 2 for formal definitions of parameterized complexity). In many cases it is known that going below exponential time would violate the so-called *Exponential Time Hypothesis* [17]. However, a limited set of problems admits subexponential-time algorithms, which are significantly more efficient than the 'standard' exponential-time algorithms.

* The second author is supported by the ERC grant "Rigorous Theory of Preprocessing" no. 267959, whereas the remaining authors are partially supported by the ERC grant PAAl no. 259515. Additionally, the first and third authors acknowledge the support of the Foundation for Polish Science.

The main source of subexponential-time algorithms in parameterized complexity is the theory of bidimensionality. This framework applies to sparse graph classes, such as planar graphs and graphs with a fixed excluded minor. Without going into technical details, the framework provides subexponential-time parameterized algorithms [8] and linear kernels [14] for problems that fulfill a special density property. The problems that fall into this category include Dominating Set, Vertex Cover, and Feedback Vertex Set.

However, for some well-known problems the bidimensionality framework fails, with probably the most prominent and important example being the Steiner Tree problem. In this problem we are given a n-vertex graph G, an integer k, and a set of terminals $S \subseteq V(G)$. We are to find a tree T in G with at most k edges such that $S \subseteq V(T)$. The computational complexity of the Steiner Tree problem on general graphs is by now fully understood. It has a parameterized algorithm working in $\mathcal{O}(2^{|S|}n^{\mathcal{O}(1)})$ time, which is tight under Strong Exponential Time Hypothesis [21, 7], and it is unlikely to admit a polynomial kernel [10] (i.e., a polynomial-time preprocessing algorithm reducing the instance size to $|S|^{O(1)}$). It has a constant-factor approximation algorithm [6], but is APX-hard [3]. On the other hand, on planar graphs, the problem is known to be NP-hard [18, 15] and to possess an EPTAS [5]. It has remained unclear, however, whether topological assumptions on the graph (such as planarity) could speed up parameterized algorithms or provide a polynomial kernel.

In this work, we aim to bridge this gap by providing the first known subexponential-time parameterized algorithm for Steiner Tree on planar graphs. The algorithm is parameterized by the size k of the tree. We call this problem Planar Steiner Tree. Tazari [22] showed that this problem admits an $\mathcal{O}(2^{\mathcal{O}(\sqrt{k \log n})}\, n^{\mathcal{O}(1)})$-time algorithm by applying Baker's classical approach [1]. He explicitly posed the question whether a subexponential-time parameterized algorithm for Planar Steiner Tree exists. We answer this question positively and develop an algorithm running in $\mathcal{O}(2^{\mathcal{O}((k \log k)^{2/3})}\, n)$ time and polynomial space.

Our approach starts with the observation that to obtain a subexponential-time parameterized algorithm for Planar Steiner Tree it suffices to give a subexponential kernel and apply Tazari's algorithm to it. That is, we only need to develop an algorithm that reduces the size of the graph to be subexponential in k. To achieve this goal, we take a path that differs significantly from previous approaches, which were based on bidimensionality theory and which relied on some sort of grid minor theorem. Instead, we bring the strip and brick decomposition (developed by Klein and Borradaile et al. [19, 5] to obtain the EPTAS for Steiner Tree on planar graphs) to the parameterized setting. Roughly speaking, we partition the graph into a polynomial (in k) number of *bricks*, each of polynomial perimeter, such that within each brick the interaction between the solution and the perimeter of the brick is bounded *sublinearly* in k. Then the number of partial solutions within a brick is subexponential. We can restrict the graph to contain only the partial solutions, and thus obtain a graph of subexponential size. However, we were not able to obtain a true subexponential kernel for the problem. Our decomposition algorithm instead relies on branching, and in fact we obtain a subexponential number of subexponential kernels for the input instance.

We believe that our approach, which brings significantly new techniques to the parameterized complexity community, is of independent interest and will inspire further research on subexponential-time algorithms on sparse graph classes.

2 Preliminaries

For the standard graph notation used throughout this paper, we refer to [9]. We additionally need the following notation. If C is a simple cycle in a planar graph with fixed embedding, then $C[x, y]$ denotes the unique counter-clockwise path on C between x and y. When we refer to an order of vertices on a cycle, we mean their counter-clockwise order along the cycle. For a path or a cycle X, $|X|$ denotes the length of X in terms of the number of its edges.

2.1 Parameterized complexity

Parameterized complexity aims at explaining time and space complexities of various, usually NP-hard problems with the help of multivariate analysis. Instead of looking at the instance only from the side of the classical input size measure, we seek relevant parameters that are responsible for the exponential blow-up in the complexity of an algorithm. These parameters generally correspond to natural aspects of the input instance, such as the solution size or some structural parameter of the input graph. In other words, we try to distinguish easy and hard instances of an NP-hard problem by introducing an appropriate measure.

Formally, an instance comes with an integer parameter k. A *parameterized problem* Q then is a subset of $\Sigma^* \times \mathbb{N}$ for some finite alphabet Σ. We say that the problem is *fixed-parameter tractable* (*FPT*) if there exists an algorithm solving any instance (x, k) in time $f(k)\,\mathrm{poly}(|x|)$, for some computable (usually exponential) function f. It is known that a problem is FPT if and only if it is *kernelizable*: a *kernelization algorithm* for a problem Q takes an instance (x, k) and in time polynomial in $|x| + k$ produces an equivalent instance (x', k') (i.e., $(x, k) \in Q$ if and only if $(x', k') \in Q$) such that $|x'| + k' \leq g(k)$ for some computable function g. The function g is the *size of the kernel*, and if it is polynomial (linear), then we say that Q admits a *polynomial (linear) kernel*. For more detailed expositions, see e.g. [11, 13].

3 Bricks and strip decompositions

In this section, we recall the decomposition framework of Klein and Borradaile et al. [19, 5] and modify it to suit our purposes. The original framework operates on weighted graphs and uses approximate distances, as it was designed to be the main tool in an approximation algorithm (the EPTAS for Planar Steiner Tree). However, for our parameterized algorithm it is crucial to use exact distances between vertices. At the same time, we may use the fact that our graphs are unweighted. With this in mind, we develop a modified framework.

Let (G, S, k) be a Planar Steiner Tree instance. We start by manipulating the graph in such a way that all terminals lie on the outer face, in a similar manner as in the work of Borradaile et al. [5]. Intuitively, we find an approximate Steiner tree along which we cut the graph open and then make the resulting face the outer face (see Figure 1). Formally, we first compute an approximate Steiner tree T_{apx} connecting S. We could use an $O(n \log n)$-time 2-approximation algorithm [20, 23, 24] for this. However, since we aim for a linear dependency on n in the running time, we take a different approach. We compute a breadth-first search tree from a fixed terminal, and iteratively remove any nonterminal leafs from this tree. This takes linear time. The resulting tree is a (minimal) Steiner tree connecting S, which we use as T_{apx}. Note that the distance in T_{apx} between any two terminals must be at most k, or we may return NO. Hence we verify that T_{apx} has at most k^2 edges; otherwise, we return NO.

Given the tree T_{apx}, we cut the plane open along the tree, duplicating every edge of T_{apx}. Let $\hat{G}$ be the resulting graph. From here on, we fix a plane embedding of $\hat{G}$ that has the interior of the tree T_{apx} as the outer face (see Figure 1). Observe that now the terminals S lie only on the outer face. Moreover, each vertex or edge of $\hat{G}$ can be mapped to a vertex or edge in G; we denote this map by $\pi : V(\hat{G}) \cup E(\hat{G}) \to V(G) \cup E(G)$. Note that only vertices and edges of T_{apx} possibly appear more than once in the domain of this map.

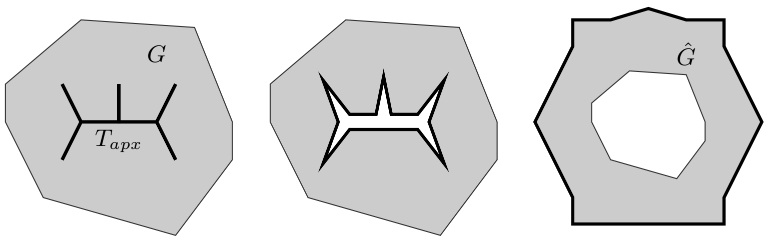


Figure 1 Construction of the cut-open graph $\hat{G}$.

We can now proceed with the basic notion of a *brick*. We call a vertex or edge of a plane graph *enclosed by a closed walk* of that graph if it is contained in a bounded Jordan region defined by the Jordan curve which the walk induces in the plane embedding.

▶ **Definition 1.** A *brick* H is the subgraph of $\hat{G}$ enclosed by a simple cycle of $\hat{G}$. This cycle is called the *perimeter* of the brick and denoted by ∂H.

We will also use the term perimeter to refer to the length of the perimeter of a brick H. By int$H$ we denote the set of edges enclosed by ∂H, i.e., int$H = E(H) \setminus E(\partial H)$. Observe that although $\pi(\partial H)$ may not always be a simple cycle in G, it is still a closed walk without self-crossings. Moreover, int$H$ is isomorphic to the part of G on one of the sides of $\pi(\partial H)$. Finally, observe that $\hat{G}$ itself is a brick with the Eulerian tour of T_{apx} as its perimeter.

The algorithm of this paper continuously decomposes bricks into smaller bricks. Here, a *decomposition* of a brick H is a collection of bricks inside H such that every face inside H belongs to exactly one of these bricks. Any brick produced by the algorithm will be immediately decomposed further into a particular type of bricks, called *strips*.

▶ **Definition 2.** A brick H is called a *strip* if ∂H can be partitioned into two paths R and B (called the *red* and *blue* paths below), such that

- B is the shortest path (in H) between its endpoints,
- every proper subpath of R is the shortest path (in H) between its endpoints.

The second condition is equivalent to saying that if $R = r_1, \ldots, r_{|R|}$, then $r_2, \ldots, r_{|R|}$ is a shortest path between r_2 and $r_{|R|}$ and $r_1, \ldots, r_{|R|-1}$ is a shortest path between r_1 and $r_{|R|-1}$. Observe that R and B share no edges, but do share two vertices of ∂H. All algorithms in the paper will construct and maintain R and B for each strip.

Klein [19] showed that, given a n-vertex brick H, one can decompose it into strips in time polynomial in n. We need a slightly different result for our unweighted, parameterized case.

▶ **Lemma 3.** *There exists an $\mathcal{O}(|\partial H|^2\, n)$-time algorithm that, given a brick H, decomposes it into at most $|\partial H|$ strips of perimeter at most $|\partial H|$ each.*

Proof. To prove the lemma it is sufficient to show an algorithm that in $\mathcal{O}(|\partial H|\, n)$ time decomposes the brick H into a strip H_0 and a number of bricks $H_1, H_2, \ldots, H_r$, such that the sum of the perimeters of all bricks $H_1, H_2, \ldots, H_r$ is strictly smaller than $|\partial H|$. We may then recursively decompose the bricks $H_1, H_2, \ldots, H_r$. The decrease in the total length of the perimeters yields the bound on the total number of strips in the obtained decomposition via a trivial induction, and also proves the claimed running time bound.

We say that an ordered pair of vertices $(x, y) \in V(\partial H) \times V(\partial H)$ is *cuttable* if $\partial H[x, y]$ is not a shortest path between x and y in H. A cuttable pair (x, y) is *minimal* if there does not

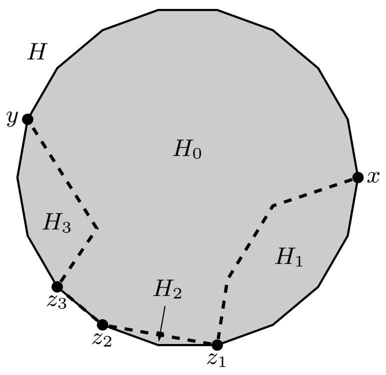


Figure 2 Step of the decomposition algorithm of Lemma 3. A brick H is decomposed into a strip H_0 and bricks H_1, H_2 and H_3. The dashed path is the path P.

exist another cuttable pair (x', y') with $\partial H[x', y']$ being a proper subpath of $\partial H[x, y]$. Note that a cuttable pair always exists in a brick H: since ∂H is a simple cycle, $|\partial H| \geq 3$ and a pair (x, y) is cuttable whenever $\partial H[y, x]$ consists of a single edge. Therefore a minimal cuttable pair always exists. Moreover, we can find such a pair in $\mathcal{O}(|\partial H|\, n)$ time using a breadth-first search for each vertex of ∂H.

Let (x, y) be a minimal cuttable pair in H and let P be a shortest path from x to y in H. We claim that P does not contain any vertex from the interior of the path $\partial H[x, y]$. Indeed, if z would be such a vertex, then a subpath of P from x to z or a subpath of P from z to y would witness that (x, z) or (z, y) is a cuttable pair, contradicting the minimality of (x, y).

We infer that $\partial H[x, y] \cup P$ is a simple cycle in H, and let H_0 be the part of H enclosed by $\partial H[x, y] \cup P$ (see Figure 2). Since P is a shortest path between x and y, $|P| \leq |\partial H[y, x]|$ and we infer that $|\partial H_0| \leq |\partial H|$. Moreover, the choice of P and the pair (x, y) implies that H_0 is a strip, with the blue path being the path P and the red path being $\partial H[x, y]$.

Let $x = z_0, z_1, z_2, \ldots, z_q = y$ be vertices of $V(P) \cap V(\partial H)$ in the order in which they appear on the path P (see Figure 2). We claim that they appear in the reverse order on the path $\partial H[y, x]$, that is, on the path $\partial H[y, x]$ the vertex z_i is closer than z_j is to x whenever $i < j$. Assume otherwise, and let i be the smallest index such that z_{i+1} is closer to x than z_i on the path $\partial H[y, x]$. Clearly $i \geq 1$. As P is a shortest path between x and y, P is a simple path, the vertices z_j are distinct, and the subpath of P from x to z_i separates z_{i+1} from y in the graph H. Therefore the subpath of P from x to z_i intersects the subpath from z_{i+1} to y, a contradiction to the fact that P is a simple path.

For $1 \leq i < q$, consider a closed walk P_i in H that consists of a subpath of P from z_{i-1} to z_i and of $\partial H[z_i, z_{i-1}]$. Note that P_i is a simple cycle unless it is of length 2. Let $H_1, H_2, \ldots, H_r$ be the set of all bricks enclosed by the paths P_i that are simple cycles (see Figure 2). From the previous claim we infer that $H_0, H_1, H_2, \ldots, H_r$ is a decomposition of H into $r + 1$ bricks. Moreover, $\sum_{i=1}^{r} |\partial H_i| \leq |P| + |\partial H[y, x]| < |\partial H[x, y]| + |\partial H[y, x]| = |\partial H|$. This finishes the proof of the lemma. ◀

If the algorithm of Lemma 3 is applied to a brick H, then a strip of the decomposition containing a vertex or edge of ∂H has that vertex or edge on its perimeter. In particular, if we apply the algorithm to the brick $\hat{G}$, then no terminal will lie in the interior of a strip.

4 A subexponential-time parameterized algorithm

We now show how to use strip decompositions to give a subexponential-time parameterized algorithm for Planar Steiner Tree.

4.1 Light strip decompositions

In this section, we formally define what we mean by the interaction of a Steiner tree with a decomposition of $\hat{G}$ into strips. Moreover, we show that if we know that this interaction is bounded sublinearly in k, then we can indeed find a Steiner tree of size at most k in subexponential time. Throughout, let (G, S, k) be an instance of Planar Steiner Tree and assume that it is a YES-instance. We will also assume that we have constructed the cut-open graph $\hat{G}$ as described before. Furthermore, we call a Steiner tree $T \subseteq E(G)$ *optimal* if the terminals in S are connected by T, $|T| \leq k$, and $|T|$ is minimum.

As a first step, we project trees in G onto $\hat{G}$. This can be done in a natural way using the mapping π. Somewhat abusing notation, given a tree T in G, we say that an edge e of $\hat{G}$ belongs to T if $\pi(e) \in T$. Note that this projection of T onto $\hat{G}$ may no longer be connected, and some edges of the tree may be duplicated if they coincide with the edges of T_{apx}. However, inside a brick the tree T should behave similarly in G and in $\hat{G}$, because the interiors of bricks are isomorphic in G and $\hat{G}$.

Using this projection, we can formally define the interaction between an optimal Steiner tree and a decomposition of $\hat{G}$ into strips.

▶ **Definition 4.** Let T be an optimal Steiner tree, let H be a strip in some strip decomposition of $\hat{G}$, and let $v \in V(\partial H)$. We say that v is a *portal for* T if v is incident to an edge e that belongs both to the interior of H and to T (formally, $e \in \text{int}H$ and $\pi(e) \in T$).

The number of portals of strips allows a distinction between *light* and *heavy* strips of a strip decomposition of $\hat{G}$. The threshold is based on a number $p = p(k) = k^{2/3}/\log^{1/3} k$. We choose to represent it symbolically in order to expose the nature of the trade-offs made in the design of the algorithm. We implicitly use that $p = o(k)$ and that $p, k/p = \omega(1)$.

▶ **Definition 5.** Let T be an optimal Steiner tree and let H be a strip in some strip decomposition of $\hat{G}$. A strip H is called *light with respect to* T if ∂H contains at most k/p portals for T. Otherwise, H is called *heavy with respect to* T.

We call a strip decomposition of $\hat{G}$ *light* if all its strips are light with respect to some optimal Steiner tree T. The next lemma shows that knowing a light strip decomposition of a particular type allows the instance to be solved.

▶ **Lemma 6.** *There exists an algorithm which takes as input an instance (G, S, k) of the* Planar Steiner Tree *problem together with the tree T_{apx}, the graph $\hat{G}$, and a strip decomposition $\mathcal{D}$ of $\hat{G}$ such that (i) $\mathcal{D}$ consists of at most $\mathcal{O}(k^2 p)$ strips of perimeter at most $\mathcal{O}(k^2)$ each, and (ii) no terminal is in the interior of a strip of $\mathcal{D}$, and which outputs either nothing or a solution of size at most k. Moreover, if (G, S, k) is a YES instance and $\mathcal{D}$ is light, then it returns a solution of size at most k. The algorithm runs in $\mathcal{O}(2^{\mathcal{O}(\sqrt{(k^2 \log k)/p})} n)$ time and polynomial space.*

Proof. The algorithm starts by marking the set of edges that it will be allowed to use. First, we mark all the perimeters of all the strips in the decomposition $\mathcal{D}$. As there are at most $\mathcal{O}(k^2 p)$ strips of perimeter at most $\mathcal{O}(k^2)$ each, this procedure marks at most $\mathcal{O}(k^4 p)$ edges in total. Second, we iteratively consider all strips $H \in \mathcal{D}$. For every strip H, consider each set $X \subseteq V(\partial H)$ of cardinality at most k/p. We then compute an optimal Steiner tree in H connecting X. This is an instance of the Planar Steiner Tree problem where all terminals lie on the outer face. Erickson et al. [12, p. 661] proved that such instances can be solved in $O(\ell^3 n + \ell^2 n \log n)$ time, where ℓ is the number of terminals. However, as noted by Borradaile et al. [5, p. 3], this running time can be trimmed to $\mathcal{O}(\ell^3 n)$. Hence in $\mathcal{O}(k^{\mathcal{O}(1)} |H|)$ time we can compute an optimal Steiner tree in H connecting X. If the cost of this tree does not exceed k, then we mark its edges. Note that the number of edges marked in this manner is at most $\mathcal{O}(k^2 p) \cdot k/p \cdot \binom{\mathcal{O}(k^2)}{k/p} \cdot k = \mathcal{O}(k^{\mathcal{O}(k/p)})$. Hence, the total number of edges marked is $\mathcal{O}(k^{\mathcal{O}(k/p)})$.

Observe that this marking of $\hat{G}$ immediately implies a marking of G using the mapping π. We now delete all the unmarked edges of G as well as any nonterminals that become isolated, and apply the algorithm of Tazari [22] to the remaining instance. The graph of this instance has $\mathcal{O}(k^{\mathcal{O}(k/p)})$ vertices. Hence Tazari's algorithm runs in $\mathcal{O}(2^{\mathcal{O}(\sqrt{k \log k^{\mathcal{O}(k/p)}})}) = \mathcal{O}(2^{\mathcal{O}(\sqrt{(k^2 \log k)/p})})$ time and polynomial space. If Tazari's algorithm finds a solution, then it is a solution of size at most k and we output it; otherwise, we output nothing. The marking procedure takes $\mathcal{O}(k^{\mathcal{O}(k/p)} n) \leq \mathcal{O}(2^{\mathcal{O}(\sqrt{(k^2 \log k)/p})} n)$ time and polynomial space as well.

It remains to prove that if (G, S, k) is a YES instance and every strip of $\mathcal{D}$ is light with respect to an optimal Steiner tree T, then there exists an optimal solution T' that uses only the marked edges. Consider a strip $H \in \mathcal{D}$ and let $C_1, C_2, \ldots, C_\ell$ be the components of the forest $T \cap \mathrm{int}H$. Each component C_i is a Steiner tree connecting incident portals. As the number of portals incident to C_i is bounded by k/p, $|C_i| \leq k$, and the interiors of the bricks are isomorphic in G and $\hat{G}$, we have marked some optimal Steiner tree C_i' connecting the same portals at non-greater cost. Replace each component C_i with C_i', and perform such replacements in all the strips of $\mathcal{D}$, thus obtaining a tree T'. Clearly, T' is still a solution, it is not more expensive than T, and it only uses marked edges. ◀

It remains to find a light strip decomposition that is required by the lemma.

4.2 A branching algorithm to obtain a light strip decomposition

In this section, we develop a branching algorithm that outputs many strip decompositions, one of which is light. It was observed before that $\hat{G}$ is a brick, which can thus be decomposed into strips using Lemma 3. This gives an initial strip decomposition where no terminal is in the interior of a strip. However, we cannot guarantee that this decomposition is light. The idea is therefore to guess a strip which is heavy with respect to some optimal Steiner tree, and then to partition it into two simpler bricks, which are subsequently decomposed again into strips using Lemma 3. Since we have no way of knowing which strips might be heavy or what a good partition is, we apply branching and try all possibilities. Our analysis shows that after branching a certain amount of times, we find a light strip decomposition in one of the branches. We now present a formal description of this algorithmic intuition.

The main analytical tool to measure the progress of the algorithm is the following potential function. For a strip decomposition $\mathcal{D}$ of $\hat{G}$ and an optimal solution T, let

$$\Phi(\mathcal{D}, T) = \left(4 \cdot \sum_{H \in \mathcal{H}(\mathcal{D},T)} \frac{|\mathrm{int}H \cap \pi^{-1}(T)|}{k/p}\right) - |\mathcal{H}(\mathcal{D}, T)|.$$

Here $\mathcal{H}(\mathcal{D}, T)$ is the set of strips from $\mathcal{D}$ that are heavy with respect to T. Note that the potential is always nonnegative, as heavy strips each contain more than k/p portals for T

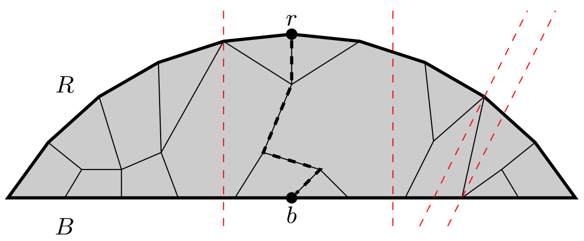


Figure 3 The split asserted by the statement of Lemma 7. The red dashed lines indicate the partition into portal components. The dashed path represents a path of length at most k between r and b, which must exist because r and b belong to the same portal component.

and thus their interiors each contain more than $k/(2p)$ edges of T. Moreover, $\Phi(\mathcal{D}, T) \leq 4p$ for any decomposition $\mathcal{D}$ and any solution T. Finally, if $\Phi(\mathcal{D}, T) = 0$ for some decomposition $\mathcal{D}$ and some solution T, then all the strips of $\mathcal{D}$ are light with respect to T. The potential function enables the definition of an *extremal* solution for a strip decomposition $\mathcal{D}$: let $T(\mathcal{D})$ denote an optimal solution that minimizes $\Phi(\mathcal{D}, T(\mathcal{D}))$.

We are now ready formalize the partition of a heavy strip (see Figure 3 for an illustration).

▶ **Lemma 7.** *Let (G, S, k) be an instance of* Planar Steiner Tree *and let H be a strip in some strip decomposition $\mathcal{D}$ of $\hat{G}$ not containing terminals in the interior, with R and B being the red and blue paths of ∂H, respectively. Assume that (G, S, k) is a YES-instance and assume furthermore that H is heavy with respect to $T(\mathcal{D})$. Let $\ell \geq k/p$ be the number of portals in H for solution $T(\mathcal{D})$. Then there exist vertices $r \in V(R)$ and $b \in V(B)$ such that*
(i) there exists a path in H between r and b that avoids ∂H apart from the endpoints and has length at most k,
(ii) for every such path P, the bricks with perimeters $\partial H[r, b] \cup P$ and $\partial H[b, r] \cup P$ both contain at least $\ell/2 - 2$ portals for $T(\mathcal{D})$.

Proof. Let F be the forest induced in $T(\mathcal{D})$ by the internal edges of the strip, i.e., $F = T(\mathcal{D}) \cap \operatorname{int} H$. We say that two edges $e_1, e_2 \in F$ are in the same *portal component* of F if some endpoint of e_1 can be connected to some endpoint of e_2 by a path in F traversing only vertices not from $V(\partial H)$. Observe that this path can have length 0, but then its only vertex cannot belong to $V(\partial H)$. We observe that this relation is an equivalence relation. Note that the partition of the edges of F into portal components can be more refined than a partition into connected components, as edges from F incident to the same portal are all in different portal components. We say that a portal v is incident to a portal component C if v is incident to an edge from C.

Claim 1. Every portal component C is incident to a portal belonging to $V(R)$ and to a portal belonging to $V(B)$.

Assume first that C is incident only to portals from $V(R)$. Let r_1 and r_2 be the first and the last portal on R that are incident to C, in counter-clockwise direction on ∂H. As C is not incident to portals from $V(B)$ and the endpoints of R belong to $V(B)$, we have that r_1 and r_2 lie in the interior of R. From the definition of a strip, we know that $\partial H[r_1, r_2]$ is a shortest path between r_1 and r_2. We infer that in T we can substitute the portal component C with the path $\partial H[r_1, r_2]$, thus creating a solution with non-greater cost (as C contains some path from r_1 to r_2) and with strictly smaller potential. This contradicts the properties of $T(\mathcal{D})$. The argument that C must be incident to a portal from $V(B)$ is analogous. This settles Claim 1.

Using Claim 1, we prove the claimed existence of the vertices r and b. Let $r_1, r_2, \ldots, r_t$ be the portals on R, in clockwise direction on ∂H, and let $b_1, b_2, \ldots, b_s$ be the portals on B, in counter-clockwise direction on ∂H. Note that possibly $r_1 = b_1$ or $r_t = b_s$. For indices $i \in \{1, 2, \ldots, t\}$ and $j \in \{1, 2, \ldots, s\}$, let $f(i, j)$ be the number of portals in the interior of the path $\partial H[r_i, b_j]$, i.e., $f(i, j) = |\{r_1, \ldots, r_{i-1}\} \cup \{b_1, \ldots, b_{j-1}\}|$. Let (i_0, j_0) be such a pair for which (a) r_{i_0} and b_{j_0} are incident to the same portal component C, and (b) $f(i_0, j_0) \leq \ell/2 - 1$ but is maximum. By Claim 1, we infer that r_1 and b_1 are always incident to the same portal component and that $f(1, 1) = 0$. Hence, such a pair (i_0, j_0) is well defined. Let $r = r_{i_0}$ and $b = b_{j_0}$. As r and b are incident to the same portal component, property (i) is satisfied. To prove property (ii), we need the following claim:

Claim 2. $f(i_0, j_0) \geq \ell/2 - 2$.

Assume otherwise, i.e. $f(i_0, j_0) \leq \ell/2 - 3$. If $i_0 = t$ and $j_0 = s$, then $f(i_0, j_0) \geq \ell - 2 > \ell/2 - 3$, a contradiction. If $i_0 < t$ and r_{i_0+1} is incident to the same portal component as r and b, then we have that $f(i_0 + 1, j_0) \leq \ell/2 - 1$, which is a contradiction with the choice of (i_0, j_0). An analogous contradiction occurs if $j_0 < s$ and b_{j_0+1} is incident to the same portal component as r and b. Now note that if $i_0 = t$ and $j_0 < s$, then by Claim 1 portal b_{j_0+1} is incident to the same portal component as r and b, which, as observed, gives a contradiction. Similarly we exclude the case when $j_0 = s$ and $i_0 < t$. Therefore $i_0 < t$, $j_0 < s$, and both r_{i_0+1} and b_{j_0+1} do not belong to the same portal component as r and b. By Claim 1, r_{i_0+1} and b_{j_0+1} are incident to the same portal component. As $f(i_0+1, j_0+1) \leq f(i_0, j_0)+2 \leq \ell/2-1$, this contradicts the choice of (i_0, j_0) and settles Claim 2.

We now prove that property (ii) is satisfied. Let P be any path in H between r and b that avoids ∂H apart from the endpoints. Consider the brick with perimeter $\partial H[r, b] \cup P$ and observe that all the portals in the interior of $\partial H[r, b]$ are still portals in this brick. As $f(i_0, j_0) \geq \ell/2 - 2$, this brick has at least $\ell/2 - 2$ portals. Similarly, all the portals in the interior of $\partial H[b, r]$ are still portals in the brick with perimeter $\partial H[b, r] \cup P$. Hence, this brick also has at least $\ell - f(i_0, j_0) - 2 \geq \ell - (\ell/2 - 1) - 2 \geq \ell/2 - 2$ portals. ◀

Note that $\ell/2 - 2 = \omega(1)$, so we can assume that vertices r, b given by Lemma 7 are distinct from each other. Hence, the obtained decomposition is non-degenerate.

Using Lemma 7, we can give the branching strategy.

▶ **Lemma 8.** *There exists an algorithm that, given an instance (G, S, k) of* Planar Steiner Tree, *the tree T_{apx}, and the graph $\hat{G}$, in time $\mathcal{O}(k^{\mathcal{O}(p)} n)$ outputs a sequence $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_t$ of strip decompositions of $\hat{G}$ such that the following properties hold:*

(i) $t = k^{\mathcal{O}(p)}$;

(ii) each decomposition consists of at most $\mathcal{O}(k^2 p)$ strips of perimeter at most $\mathcal{O}(k^2)$ each, and no terminal lies in the interior of any strip;

(iii) if (G, S, k) is a YES-instance, then there is a decomposition $\mathcal{D}_i$ such that all the strips of $\mathcal{D}_i$ are light with respect to $T(\mathcal{D}_i)$.

The working space of the algorithm, excluding the output decompositions, is polynomial.

Proof. We follow a recursive branching procedure. At depth d of the recursive procedure we maintain a strip decomposition $\mathcal{D}$ with following properties: $\mathcal{D}$ consists of at most $\mathcal{O}(dk(d + k))$ strips of perimeter at most $2k^2 + dk$ each, and no terminal lies in the interior of any strip. Moreover, at depth d we branch into at most $\mathcal{O}(dk^3(d + k)^3)$ branches. Finally, we show that it suffices to run the branching to a depth of at most $8p$. Hence, as $p = o(k)$, the total number of produced decompositions is bounded by $t = k^{\mathcal{O}(p)}$.

First, we need to provide a starting decomposition at the 0-th level of the recursion. We start with a single brick $\hat{G}$ — that is, the entire graph $\hat{G}$ with its outer face (the Euler tour of T_{apx}) as the perimeter. Recall that the size of T_{apx} is bounded by k^2, and thus the perimeter of $\hat{G}$ is at most $2k^2$. We apply Lemma 3 to this brick in order to obtain some strip decomposition $\mathcal{D}_0$ of $\hat{G}$. This decomposition consists of at most $2k^2$ strips, each having perimeter at most $2k^2$. As all the terminals lie on the perimeter of $\hat{G}$, no terminal lies inside any strip of $\mathcal{D}_0$. This decomposition is the initial one for the branching procedure.

We now describe the operations performed on level d of the recursion. We do the following:

- output the current decomposition $\mathcal{D}$ as the next $\mathcal{D}_i$;
- branch on each H, r, b, where H is a strip of D with R, B being the red and the blue path of ∂H, respectively, and $r \in V(R), b \in V(B)$ are vertices that can be connected via a path P (chosen arbitrarily) of length at most k that avoids ∂H apart from the endpoints. For each such H, r, b, divide H into bricks with perimeters $\partial H[r, b] \cup P$ and $\partial H[b, r] \cup P$, and apply Lemma 3 to decompose both bricks into strips. Proceed with the in total $\mathcal{O}(dk(d + k) \cdot (k(d + k))^2)$ obtained decompositions to the next level of the recursion.

Observe that if $|\partial H| \leq 2k^2 + dk$, then both bricks created in the initial division have perimeter at most $2k^2 + (d + 1)k$, as $|P| \leq k$. Applying Lemma 3 cannot create strips with longer perimeter, and increases the number of strips in the decomposition by at most $4k^2 + 2(d+1)k$. Hence, the $\mathcal{O}(dk(d + k))$ bound on the number of strips on level d of the recursion holds. As all the obtained decompositions only refine the initial one, in all the subcases no terminal will lie inside any strip.

We now analyse the running time of the algorithm. Note that finding the path P for a given strip H and vertices $r \in V(R), b \in V(B)$ boils down to finding a shortest path in an appropriate subgraph of the graph $\hat{G}$, which can be done in $O(n)$ time. As we output $\mathcal{O}(k^{\mathcal{O}(p)})$ decompositions, the bound on the running time follows.

It remains to prove that if we perform $8p$ levels of the branching procedure, then property (iii) will hold. We do this by examining the change of the potential $\Phi(\mathcal{D}, T(\mathcal{D}))$ during the branching procedure.

Claim 1. Let $\mathcal{D}$ be the decomposition at some step of the branching procedure, such that strip $H \in \mathcal{D}$, where R, B are the red and blue paths of ∂H, respectively, is heavy with respect to $T(\mathcal{D})$. Let $r \in V(R)$ and $b \in V(B)$ be the vertices whose existence is asserted by Lemma 7. Then in the branch of H, r, and b the potential decreases by at least 1/2.

Let $\mathcal{D}'$ be the strip decomposition after performing the aforementioned branching. We prove a slightly stronger claim, namely that $\Phi(\mathcal{D}', T(\mathcal{D})) \leq \Phi(\mathcal{D}, T(\mathcal{D})) - 1/2$. As $\Phi(\mathcal{D}', T(\mathcal{D}')) \leq \Phi(\mathcal{D}', T(\mathcal{D}))$ by the definition of $T(\mathcal{D}')$, this implies Claim 1.

We consider three cases. First assume that all the strips created when constructing $\mathcal{D}'$ from $\mathcal{D}$ are in fact light with respect to $T(\mathcal{D})$. Then the first term in the potential function decreases by at least 2, as every heavy brick has more than $k/(2p)$ edges from $T(\mathcal{D})$ in the interior, while the second term increases by 1. Hence the potential decreases by at least 1.

Second, assume that exactly one created strip H_0 is heavy. Without loss of generality assume that this strip was obtained while decomposing the brick with perimeter $\partial H[r, b] \cup P$. By Lemma 7, the brick with perimeter $\partial H[b, r] \cup P$ has at least $k/(2p) - 2 \geq k/(4p)$ portals; here we use the bound $k \geq 8p$ that holds for large enough k due to $p = o(k)$. Hence, it has at least $k/(8p)$ edges from $T(\mathcal{D})$ in the interior. Therefore, the brick with perimeter $\partial H[r, b] \cup P$ has at least $k/(8p)$ edges from $T(\mathcal{D})$ fewer in the interior than H. The strip H_0 can only have fewer edges from $T(\mathcal{D})$ in the interior, so the first term of the potential function decreases by at least 1/2. As the second term is unchanged, this settles this case.

Finally, assume that at least two created strips are heavy. Then the first term of the potential function cannot increase, as every edge of $T(\mathcal{D})$ in the interior of the created heavy strips was already in the interior of the heavy strip H, while the second term decreases by at least 1. Hence the potential decreases by at least 1 in this case. This settles Claim 1.

Since the potential initially is at most $4p$, and Claim 1 proves that the potential decreases by at least $1/2$ in one of the branches, it suffices to branch to at most $8p$ levels deep to ensure that we find a decomposition $\mathcal{D}_i$ for which $\Phi(\mathcal{D}_i, T(\mathcal{D}_i)) = 0$. As observed before, this implies that all strips of $\mathcal{D}_i$ are light with respect to $T(\mathcal{D}_i)$. This proves property (iii). ◀

We are ready to prove the main result of this paper.

▶ **Theorem 9.** *The* Planar Steiner Tree *problem can be solved in* $\mathcal{O}(2^{\mathcal{O}((k \log k)^{2/3})}\, n)$ *time and polynomial space.*

Proof. We start by constructing the approximate solution T_{apx} and the graph $\hat{G}$ in $\mathcal{O}(n)$ time. Then, we run the algorithm given by Lemma 8 and to each output decomposition we apply Lemma 6. The correctness of the algorithm follows from property (iii) in the statement of Lemma 8, while the claim on the running time follows from the fact that to at most $\mathcal{O}(k^{\mathcal{O}(p)}) = \mathcal{O}(2^{\mathcal{O}((k \log k)^{2/3})})$ instances we apply an algorithm running in $\mathcal{O}(2^{\mathcal{O}(\sqrt{(k^2 \log k)/p})}\, n) = \mathcal{O}(2^{\mathcal{O}((k \log k)^{2/3})}\, n)$ time. The polynomial space bound can be achieved by applying Lemma 6 to each output decomposition once it is fully constructed. ◀

5 Conclusions and open problems

Although our result positively answers the question whether a subexponential-time parameterized algorithm for Planar Steiner Tree exists, it raises many new open questions.

We first ask whether our algorithm can be improved. We can show that Planar Steiner Tree cannot have a $\mathcal{O}(2^{o(\sqrt{k})}\, n^{\mathcal{O}(1)})$-time algorithm unless the Exponential Time Hypothesis is false, using the standard NP-hardness reduction from Connected Vertex Cover. It seems reasonable to think that the right upper bound is $\mathcal{O}(2^{\mathcal{O}(\sqrt{k})}\, n^{\mathcal{O}(1)})$. Such an algorithm — up to a logarithmic factor in the exponent — follows immediately if Planar Steiner Tree would admit a polynomial kernel. We conjecture that such a kernel indeed exists.

Apart from the parameterization by the size of the tree investigated in this paper, there is a second natural parameterization of the Steiner Tree problem, namely by $|S|$, the number of terminals. Recall that on general graphs the $\mathcal{O}(2^{|S|} n^{\mathcal{O}(1)})$-time algorithm due to Nederlof [21] is probably optimal [7]. Our techniques seem to break down on this stronger parameterization. Does Planar Steiner Tree admit a subexponential-time algorithm with respect to the number of terminals in the instance?

Another interesting direction is to try to generalize our algorithm to the closely related Planar Steiner Forest problem. With a bit of simple preprocessing, we can obtain an analogue of the approximate tree T_{apx} and construct the cut-open graph $\hat{G}$. Using this graph, we may perform the same branching procedure as in Lemma 8, and obtain a subexponential number of subexponential kernels for this problem. However, the last step of the algorithm — Tazari's algorithm [22] based on Baker's approach [1] — breaks down, as Steiner Forest is NP-hard on graphs of treewidth 3 [2, 16]. Thus, one needs significantly new ideas to turn a subexponential kernel into a subexponential algorithm for this problem.

Last but not least, we mention that Borradaile et al. [4] did some work on lifting the brick and strip decomposition to graphs of bounded genus. It may therefore be interesting whether our algorithm can also be extended to such graphs.

Acknowledgments
The authors thank Fedor Fomin, M.S. Ramanujan, and Marek Cygan for helpful discussions.

References

1 B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994.

2 M. Bateni, M. T. Hajiaghayi, and D. Marx. Approximation schemes for Steiner forest on planar graphs and graphs of bounded treewidth. *J. ACM*, 58(5):21, 2011.

3 M. W. Bern and P. E. Plassmann. The Steiner problem with edge lengths 1 and 2. *Inf. Process. Lett.*, 32(4):171–176, 1989.

4 G. Borradaile, E. D. Demaine, and S. Tazari. Polynomial-time approximation schemes for subset-connectivity problems in bounded-genus graphs. In *STACS*, pages 171–182, 2009.

5 G. Borradaile, P. N. Klein, and C. Mathieu. An $O(n \log n)$ approximation scheme for Steiner tree in planar graphs. *ACM Transactions on Algorithms*, 5(3), 2009.

6 J. Byrka, F. Grandoni, T. Rothvoß, and L. Sanità. An improved LP-based approximation for Steiner tree. In L. J. Schulman, editor, *STOC*, pages 583–592. ACM, 2010.

7 M. Cygan, H. Dell, D. Lokshtanov, D. Marx, J. Nederlof, Y. Okamoto, R. Paturi, S. Saurabh, and M. Wahlström. On problems as hard as CNF-SAT. In *IEEE Conference on Computational Complexity*, pages 74–84. IEEE, 2012.

8 E. D. Demaine, F. V. Fomin, M. T. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. ACM*, 52(6):866–893, 2005.

9 R. Diestel. *Graph Theory.* Springer, 2005.

10 M. Dom, D. Lokshtanov, and S. Saurabh. Incompressibility through colors and IDs. In *ICALP (1)*, pages 378–389, 2009.

11 R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, 1999.

12 R. E. Erickson, C. L. Monma, and A. F. J. Veinott. Send-and-split method for minimum-concave-cost network flows. *Mathematics of Operations Research*, 12(4):pp. 634–664, 1987.

13 J. Flum and M. Grohe. *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. Springer, 2006.

14 F. V. Fomin, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Bidimensionality and kernels. In M. Charikar, editor, *SODA*, pages 503–510. SIAM, 2010.

15 M. R. Garey and D. S. Johnson. The Rectilinear Steiner Tree problem is NP complete. *SIAM Journal of Applied Mathematics*, 32:826–834, 1977.

16 E. Gassner. The Steiner forest problem revisited. *J. Discrete Algorithms*, 8(2):154–163, 2010.

17 R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

18 R. Karp. On the computational complexity of combinatorial problems. *Networks*, 5:45–68, 1975.

19 P. N. Klein. A subset spanner for planar graphs, with application to Subset TSP. In J. M. Kleinberg, editor, *STOC*, pages 749–756. ACM, 2006.

20 K. Mehlhorn. A faster approximation algorithm for the Steiner problem in graphs. *Inf. Process. Lett.*, 27(3):125–128, 1988.

21 J. Nederlof. Fast polynomial-space algorithms using Möbius inversion: Improving on Steiner Tree and related problems. In *ICALP (1)*, pages 713–725, 2009.

22 S. Tazari. Faster approximation schemes and parameterized algorithms on H-minor-free and odd-minor-free graphs. In *MFCS*, pages 641–652, 2010.

23 P. Widmayer. On approximation algorithms for Steiner's problem in graphs. In G. Tinhofer and G. Schmidt, editors, *WG*, volume 246 of *LNCS*, pages 17–28. Springer, 1986.

24 Y.-F. Wu, P. Widmayer, and C. K. Wong. A faster approximation algorithm for the Steiner problem in graphs. *Acta Inf.*, 23(2):223–229, 1986.

The arithmetic complexity of tensor contractions

Florent Capelli[1,2], Arnaud Durand[2], and Stefan Mengel*[3]

1 **ENS Lyon, France**
`florent.capelli@ens-lyon.fr`
2 **IMJ UMR 7586 - Logique**
Université Paris Diderot F-75205 Paris, France
`durand@logique.jussieu.fr`
3 **Institute of Mathematics**
University of Paderborn D-33098 Paderborn, Germany
`smengel@mail.uni-paderborn.de`

Abstract

We investigate the algebraic complexity of tensor calulus. We consider a generalization of iterated matrix product to tensors and show that the resulting formulas exactly capture VP, the class of polynomial families efficiently computable by arithmetic circuits. This gives a natural and robust characterization of this complexity class that despite its naturalness is not very well understood so far.

1998 ACM Subject Classification F.1.3 [Computation by abstract devices]: Complexity Measures and Classes — Relations among complexity measures

Keywords and phrases algebraic complexity, arithmetic circuits, tensor calculus

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.365

1 Introduction

The question of which polynomials can be computed by polynomial size arithmetic circuits is one of the central questions of algebraic complexity. It was first brought up explicitly by Valiant [11] who formulated a complexity theory in this setting with its own complexity classes and notions of completeness. Efficient computation in Valiant's model is formalized by the complexity class VP which consists of families of polynomials that can be computed by arithmetic circuits of polynomial size. Despite recent efforts relating VP to logically defined classes of polynomial families [9, 4], this class is not very well understood. This is reflected in the low number of helpful alternative characterizations and the conspicuous absence of any known natural complete problem.

Consequently, most progress in arithmetic circuit complexity has not been achieved by considering arithmetic circuits directly, but instead by considering the somewhat more restricted model of arithmetic branching programs (see e.g. [7, 11, 10, 5]). Arithmetic branching programs are widely conjectured to have expressivity strictly between that of arithmetic formulas and circuits, but have so far been better to handle with known proof techniques. One of the nice properties of branching programs that has often played a crucial role is that they can equivalently be seen as computing a specified entry of the iterated product of a polynomial number of matrices.

* Partially supported by DFG grants BU 1371/2-2 and BU 1371/3-1 and the [European Community's] Seventh Framework Programme [FP7/2007-2013] under grant agreement n° 238381.

We extend this view on branching programs by going from matrices to higher dimensional tensors. Consequently, we also go from matrix product to the generalized notion of contraction of tensors. It turns out that generalizing iterated matrix product to iterated tensor contractions does increase the expressive power of the model and that the resulting tensor formulas capture exactly VP. This characterization of VP turns out to be fairly robust in the sense that one can add different restrictions on the dimensions of the tensors without changing the expressive power of the model at all.

This is not the first time that the complexity of tensor calculus is studied. Damm, Holzer and McKenzie [3] and later Beaudry and Holzer [1] have characterized different boolean complexity classes by formulas having matrices as inputs and using addition, matrix product and tensor product as operations. Malod [6] adapted these formulas to the arithmetic circuit setting and showed characterizations for most arithmetic circuit classes. One difference between these results and those in our paper is that in [3, 6, 1] tensors are always encoded as matrices, i.e. the tensor product is expressed as the Kronecker product of two matrices. Another difference is that both the characterization of VP obtained in [6] and the similar characterization of LOGCFL (the Boolean analogon of VP) from [3] require an additional restriction, called *tameness* on the size of matrices computed at each gate of the formula. This restriction permits to control the growth of the intermediate objects in the computation but may seem not very natural. In this present work, working directly with tensors instead of a matrix representation makes such a restriction unnecessary and a more direct connection between VP and tensor calculus is established.

2 Preliminaries

Below, $\mathbb{K}$ is a field and bold letters denote tuples when there is no ambiguity on their length.

2.1 Arithmetic circuits

We will use the well known model of arithmetic circuits to measure the complexity of polynomials. In this section we give some definitions and well known properties of arithmetic circuits (see e.g. [2, 7] for more on the subject).

An *arithmetic circuit* is a directed acyclic graph with vertices of indegree 0 or 2 called *gates*. The gates of indegree 0 are called the *inputs* and are labeled with elements of $\mathbb{K}$ or variables. The gates of indegree 2, called *computation gates*, are labeled with operations of the field ($+$ and $\times$). The polynomial computed by a gate is defined inductively. The polynomial computed by an input gate is the one corresponding to its label. The polynomial computed by a computation gate is the sum or the product of the polynomials computed by its children. We assume that there exists a distinguished gate called the *output*. The polynomial computed by an arithmetic circuit is the one computed by its output gate. The *size* of a circuit C, denoted by $|C|$, is the number of vertices of its underlying DAG.

An arithmetic circuit C is said to be *skew* if for each $\times$-gate at least one of its children is an input of the circuit. A circuit is said to be *multiplicatively disjoint* if for each $\times$-gate, its two input subcircuits are disjoint.

A family $(f_n)_{n\in\mathbb{N}}$ of polynomials is in VP if there exists a family of multiplicatively disjoint circuits $(C_n)_{n\in\mathbb{N}}$ and a polynomial P such that for all $n \in \mathbb{N}$, C_n computes f_n and $|C_n| \le P(n)$. The family (f_n) is in $\mathsf{VP}_{\mathrm{ws}}$ if the C_n are skew.

▶ Remark. Originally, VP was defined as families of polynomials that can be computed by polynomial size circuits and have polynomially bounded degree. As shown in [7] the definition given here is equivalent to the original one. We prefer this one here because the semantic condition on the degree is harder to deal with than multiplicatively disjointness which is more syntactic.

In the following, we will simulate arithmetic circuits by formulas computing tensors. We use the notion of *parse trees* of a circuit (see [7] for more details and bibliographical references on this notion). For a multiplicatively disjoint circuit C, we define its parse trees inductively. A parse tree T of C is a subgraph of C constructed as follows:

- Add the output of C to T
- For every gate v added to T do the following:
  - If v is a $+$-gate, add exactly one of its children to T.
  - If v is a $\times$-gate, add both of its children to T.

As C is multiplicatively disjoint, a parse tree of C is a tree. The monomial $m(T)$ computed by a parse tree T is the product of the labels of its leaves. The polynomial computed by C is the sum of the monomials of all parse trees of C.

2.2 Tensors

In this paper, we interpret tensors as multidimensional arrays. Their algebraic nature is not studied here. A good introduction to multilinear algebra and tensors can be found in [8].

Let $n_1, \ldots, n_k$ be k positive integers. A *k-dimensional tensor* T of *order* $(n_1, \ldots, n_k)$ is a mapping $T : [n_1] \times \ldots \times [n_k] \to \mathbb{K}$. For $i_1 \in [n_1], \ldots, i_k \in [n_k]$, we denote by $T[i_1, \ldots, i_k]$ the value of the mapping on the point $(i_1, \ldots, i_k)$. We call these values *entries* of T. We denote by $D(T)$ the *domain* of T. Obviously $D(T) = [n_1] \times \ldots \times [n_k]$. We also denote by $\dim(T)$ (equal to k, here), the dimension of T. The *size* of a tensor T, denote by $\|T\|$, is the number of entries, i.e. $\|T\| = \prod_{i=1}^{k} n_i$, where T is of order $(n_1, \ldots, n_k)$. The *maximal order* of T, denote by $\mathsf{maxorder}(T)$ is $\max_{i\in[k]} n_i$. In the following, we also call matrices (resp. vectors), tensors of dimension 2 (resp. 1).

▶ **Definition 1** (Contraction)**.** Let T be a k-dimensional tensor of order $(n_1, \ldots, n_k)$ and G an l-dimensional tensor of order $(m_1, \ldots, m_l)$ with $k, l \geq 1$. If $n_k = m_1$, we denote by $T * G$ the *contraction* of T and G on the dimensions k and 1 which is a tensor of order $(n_1, \ldots, n_{k-1}, m_2, \ldots, m_l)$ defined as $(T * G)[\mathbf{e}_1, \mathbf{e}_2] = \sum_{i=1}^{n_k} T[\mathbf{e}_1, i]G[i, \mathbf{e}_2]$ for all $\mathbf{e}_1 \in [n_1] \times \ldots \times [n_{k-1}]$ and $\mathbf{e}_2 \in [m_2] \times \ldots \times [m_l]$.

▶ Remark. Obviously, contraction is a direct generalization of the matrix product. Indeed, if both T and G are matrices, then $T * G$ is the ordinary matrix product.

▶ **Proposition 2.** *Let T, G, H be tensors with $\dim(G) \geq 2$ such that $T * (G * H)$ and $(T * G) * H$ are both well defined. Then, $T * (G * H) = (T * G) * H$.*

Proof. By direct consequence of the associativity of the ordinary matrix product. For a tensor T of dimension $k \geq 2$ and order $(n_1, \ldots, n_k)$, and a tuple $\mathbf{e}$ of length $k-2$ we define the $n_1 \times n_k$ matrix $T_{\mathbf{e}} := (T[i, \mathbf{e}, j])_{i\in[n_1], j\in[n_k]}$. Then by associativity of matrix product:

$$\forall \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3 : T_{\mathbf{e}_1} * (G_{\mathbf{e}_2} * H_{\mathbf{e}_3}) = (T_{\mathbf{e}_1} * G_{\mathbf{e}_2}) * H_{\mathbf{e}_3}.$$

Hence, the claim follows when $\dim(T) \geq 2$ and $\dim(H) \geq 2$. For $\dim(T) = 1$ or $\dim(H) = 1$ the argument is similar. ◀

▶ Observation 3. If G is a vector, then the equality of Proposition 2 may not be true anymore. For example

$$\left(\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} * \begin{pmatrix} 0 \\ 1 \end{pmatrix}\right) * \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \neq \begin{pmatrix} 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} * \left(\begin{pmatrix} 0 \\ 1 \end{pmatrix} * \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}\right).$$

where the input $\begin{pmatrix}0\\1\end{pmatrix}$ is a 1-dimensional tensor T of order (2) such that $T[1] = 0$ and $T[2] = 1$.

▶ **Definition 4.** A $\{*\}$-*formula* F is a labeled, ordered, rooted binary tree whose the leaves, called the inputs, are labeled by tensors whose entries are elements of $\mathbb{K}$ or variables and the other nodes are labeled by $*$. The tensor T_v computed by a node v is defined inductively:

- If v is a leaf then $T_v := \mathsf{label}(v)$.
- If v is labeled by $*$ and has left child v_1 and right child v_2 then $T_v := F_{v_1} * F_{v_2}$.

A $\{*\}$-formula computes the tensor computed by its root.

▶ Remark. In general, there may be a dimension mismatch in some contractions in an $\{*\}$-formula and thus the tensor computed by it may not be well-defined. However, detecting such $\{*\}$-formulas is easy and thus we only consider formulas in which no such problems occur.

As the entries of the input tensors are constants of $\mathbb{K}$ or variables, each entry of a tensor computed by a gate is a polynomial of $\mathbb{K}[X_1, \dots, X_n]$. This is why it makes sense to compare the computational power of $\{*\}$-formulas and arithmetic circuits defined in the last section. Moreover, of all the polynomials computed in the output of a $\{*\}$-formula we will mostly only be interested in one single polynomial. Thus we assume that the output tensor has only one single entry, i.e. the tensor is indeed a scalar. Observe that this form can always be achieved by contracting with vectors. We say that the scalar polynomial computed by a $\{*\}$-formula is *the* polynomial computed by it.

▶ **Definition 5.** The *size* of a $\{*\}$-formula F, denoted by $|F|$, is the number of $*$-gates plus the size of the inputs, i.e. $|F| := |\{v \mid \mathsf{label}(v) = *\}| + \sum_{T: T \text{ input of } F} \|T\|$. The *dimension* of F, denoted by $\dim(F)$ is the dimension of the tensor computed by F. The *maximal dimension* of F, denoted by $\operatorname{maxdim}(F)$ is the maximal dimension of the tensors computed at the gates of F, i.e. $\operatorname{maxdim}(F) := \max_{v\text{: gate in } F}(\dim(T_v))$. The *input dimension* of F is $\max_{v: v \text{ input of } F} \dim(T_v)$.

We will often mix the notations for tensors and for tensor formulas. For example, if F is a tensor formula computing the tensor T, we will speak of the order of F instead of T and write $F[\mathbf{e}]$ instead of $T[\mathbf{e}]$. Moreover, given two different formulas F and F', we will write $F \simeq F'$ if they compute the same tensor.

3 From arithmetic circuits to $\{*\}$-formulas

We describe how a family of polynomials in VP can be simulated by a family of $\{*\}$-formulas of polynomial size and maximal dimension 3. Our proof is inspired by a proof from [9] where it is shown that polynomials in VP can be represented by bounded treewidth CSPs.

▶ **Theorem 6.** *Let $(f_n) \in \mathsf{VP}$. There exists a family of $\{*\}$-formulas (F_n) of maximal dimension 3 and polynomial size such that F_n computes f_n for all n.*

We use the following observation from [9] which can be proved by combining results from Malod and Portier [7] and Valiant et al. [12].

▶ **Proposition 7.** *Let f be computed by an arithmetic circuit C of size s. Then there is an arithmetic circuit C' of size $s^{O(1)}$ that also computes f such that all parse trees of C' are isomorphic to a common tree $\mathcal{T}$.*

Theorem 6 follows direcly from Proposition 7 and the following lemma:

▶ **Lemma 8.** *Let C be an arithmetic circuit computing the polynomial f whose parse trees are all isomorphic to a common parse tree $\mathcal{T}$. Then there exists a $\{*\}$-formula F of maximal dimension 3 and of size $9|C|^3|\mathcal{T}|$ that computes f.*

Proof. We construct a tensor formula along the tree $\mathcal{T}$ which contains the sum of all monomials of f_n in its entries. We denote by $V(\mathcal{T})$ (resp. $V(C)$) the vertices of $\mathcal{T}$ (resp. C). For $s \in V(\mathcal{T})$, we call $\mathcal{T}_s$ the subtree of $\mathcal{T}$ rooted in s. We define a *partial parse tree* rooted in s to be a function $p : V(\mathcal{T}_s) \to V(C)$ respecting the following conditions for all $t \in V(\mathcal{T}_s)$:

1. If t is a leaf, then $p(t)$ is an input of C.
2. If t has one child t_1, $p(t)$ is a $+$-gate and $p(t_1)$ is a child of $p(t)$ in C.
3. If t has two children t_1 and t_2, then
   a. $p(t)$ is a $\times$-gate,
   b. $p(t_1)$ is the left child of $p(t)$, and
   c. $p(t_2)$ is the right child of $p(t)$.

We call these conditions the parse tree conditions. It is easy to see that when s is the root of $\mathcal{T}$ (and thus $\mathcal{T}_s = \mathcal{T}$) and $p : V(\mathcal{T}) \to V(C)$, then $p(V(\mathcal{T}))$ is the vertex set of a parse tree of C if and only if p is a partial parse tree rooted in s.

If p is a partial parse tree rooted in $s \in V(\mathcal{T})$, we define the monomial $m(p)$ computed by p by $m(p) := \prod_{t\in \mathsf{leaf}(\mathcal{T}_s)} \mathsf{label}(p(t))$. Observe that this is well defined as p respects, in particular, the first parse tree condition and thus $p(t)$ for $t \in \mathsf{leaf}(\mathcal{T}_s)$ is always an input of C. If p does not respect the parse tree conditions, we set $m(p) = 0$. With this notation we have

$$f = \sum_{p:V(\mathcal{T})\to V(C)} m(p).$$

We index the vertices of C : $V(C) = \{v_1, \dots, v_r\}$ with $r = |C|$. We denote by E the tensor of dimension 1 and order (r) such that for all $i \le r$, $E[i] = 1$ and by $\delta_{i,j}$ the Kronecker function which equals 1 if $i = j$ and 0 otherwise. We construct by induction along the structure of $\mathcal{T}$ a $\{*\}$-formula F_s for each $s \in \mathcal{T}$. The formula F_s has dimension 2, order (r, r), size at most $9r^3|\mathcal{T}_s|$ and maximal dimension 3. Furthermore, for all $i, j \le r$:

$$F_s[i, j] = \delta_{i,j} \sum_{\substack{p:V(\mathcal{T}_s)\to V(C)\\ p(s)=v_i}} m(p).$$

Observing $f = E * F_s * E$ when s is the root of $\mathcal{T}$ completes the proof. We now describe the inductive construction of F_s. Several cases occur:

s is a leaf: In this case $\mathcal{T}_s$ consists only of the leaf s. The partial parse trees of $\mathcal{T}_s$ are functions $p : \{s\} \to V(C)$ and $m(p) = \mathsf{label}(p(s))$ if $p(s)$ is a input of C and $m(p) = 0$ otherwise. Then F_s consists of a $r \times r$ input matrix I such that for all $i, j \le r$,

$$I[i, j] = \begin{cases} \delta_{i,j}\mathsf{label}(v_j) \text{ if } v_j \text{ is an input} \\ 0 \text{ otherwise.} \end{cases}$$

Obviously, F_s is of size $r \le 9r^3$, of maximal dimension 2 and $I[i, j] = \delta_{i,j} \sum_{\substack{p:\{s\}\to V(C)\\ p(s)=v_i}} m(p)$.

s has one child s_1: We start with an observation on functions $p : V(\mathcal{T}_s) \to V(C)$. Let p_1 be the restriction of p on $V(\mathcal{T}_{s_1})$. If p is a partial parse tree, then p_1 is one, too, because it fulfills the parse tree conditions for all $t \in V(\mathcal{T}_{s_1}) \subseteq V(\mathcal{T}_s)$. Moreover, $m(p) = m(p_1)$ because the leaves in p and in p_1 are the same. In addition, if p is not a partial parse tree then

- either p violates a parse tree condition for $t \in \mathcal{T}_{s_1}$. In that case, p_1 is not a partial parse tree and then $m(p) = m(p_1) = 0$,
- or $p(s)$ is not a $+$-gate,
- or $p(s)$ is a $+$-gate but $p(s_1)$ is not a child of $p(s)$.

We encode these conditions in a tensor of dimension 3 and order (r, r, r) defined as

$$M[i,j,k] := \begin{cases} \delta_{j,k} \text{ if } v_j \text{ is } +\text{-gate and } v_i \text{ is a child of } v_j \\ 0 \text{ otherwise.} \end{cases}$$

Let $F_s := E * (F_{s_1} * M)$. Formula F_s is of maximal dimension 3, dimension 2 and order (r, r). We have $|F_s| \leq 9r^3(|\mathcal{T}_s| - 1) + r^3 + 2 + \|E\| \leq 9r^3|\mathcal{T}_s|$ and

$$\begin{aligned} F_s[j,k] &= \sum_{i=1}^{r}(\sum_{p=1}^{r} F_{s_1}[i,p]M[p,j,k]) \\ &= \delta_{j,k}\sum_{i=1}^{r}(\sum_{\substack{p_1:V(\mathcal{T}_{s_1})\to V(C)\\ p_1(s_1)=v_i}} m(p_1)M[i,j,j]). \end{aligned}$$

Let p be a function $p : V(\mathcal{T}_s) \to V(C)$ such that $p(s_1) = v_i$ and $p(s) = v_j$. Let p_1 be its restriction on $V(\mathcal{T}_{s_1})$. We have $m(p) = m(p_1)M[i,j,j]$, because

- if p is a partial parse tree then $M[i,j,j] = 1$ and thus $m(p_1)M[i,j,j] = m(p_1) = m(p)$,
- if v_j is not a $+$-gate then $m(p) = 0$ and also $m(p_1)M[i,j,j] = 0$ because $M[i,j,j] = 0$,
- if $p_1(s_1) = v_i$ is not a child of $v_j = p(s)$ then $m(p) = 0$. Since $M[i,j,j] = 0$, we have $m(p) = 0 = m(p_1)M[i,j,j]$.

This part of the proof is completed by remarking that, in each case

$$F_s[j,k] = \delta_{j,k}\sum_{i=1}^{r}\sum_{\substack{p_1:V(\mathcal{T}_{s_1})\to V(C)\\ p_1(s_1)=v_i}} m(p_1)M[i,j,k] = \delta_{j,k}\sum_{\substack{p:V(\mathcal{T}_s)\to V(C)\\ p(s)=v_j}} m(p).$$

s has two children, s_1 (left child) and s_2 (right child): As above, we encode the parse tree conditions in tensors of dimension 3 and contract them correctly to compute the desired result. This time there are two different tensors: one encoding the condition 3.b and one for 3.c. Let M_L and M_R be the two following (r, r, r) tensors:

$$M_L[i,j,k] = \begin{cases} \delta_{j,k} \text{ if } v_j \text{ is a } \times\text{-gate and } v_i \text{ is the left child of } v_j \\ 0 \text{ otherwise,} \end{cases}$$

$$M_R[i,j,k] = \begin{cases} \delta_{j,k} \text{ if } v_j \text{ is a } \times\text{-gate and } v_i \text{ is the right child of } v_j \\ 0 \text{ otherwise.} \end{cases}$$

Let F_s be the formula of maximal dimension 3, dimension 2 and order (r, r) defined as

$$F_s = (E * (F_{s_1} * M_L)) * (E * (F_{s_2} * M_R)).$$

We have $|F_s| = |F_{s_1}| + \|M_L\| + \|M_R\| + |F_{s_2}| + 5 + 2\|E\| \leq 9r^3|\mathcal{T}_s|$. In addition:

$$\begin{aligned} F_s[i,j] &= \sum_{k=1}^{r}((\sum_{a=1}^{r} F_{s_1}[a,a]M_L[a,i,k]) \times (\sum_{b=1}^{r} F_{s_2}[b,b]M_R[b,k,j])) \\ &= \delta_{i,j}\sum_{a,b=1}^{r} F_{s_1}[a,a]M_L[a,i,i]F_{s_2}[b,b]M_R[b,i,i] \\ &= \delta_{i,j}\sum_{a,b=1}^{r}\sum_{\substack{p_1:V(\mathcal{T}_{s_1})\to V(C)\\ p_1(s_1)=v_a}}\sum_{\substack{p_2:V(\mathcal{T}_{s_2})\to V(C)\\ p_2(s_2)=v_b}} m(p_1)m(p_2)M_L[a,i,i]M_R[b,i,i] \end{aligned}$$

.

Similarly as before, let $p : V(\mathcal{T}_s) \to V(C)$ such that $p(s) = v_i$, $p(s_1) = v_a$ and $p(s_2) = v_b$. We denote by p_1 (resp. p_2) the restriction of p on $V(\mathcal{T}_{s_1})$ (resp. $V(\mathcal{T}_{s_2})$) and show that

$$m(p) = M_L[a,i,i]M_R[b,i,i]m(p_1)m(p_2)$$

by studying the possible cases:

- If p is a partial parse tree then p_1 and p_2 are, too. Moreover, since s has two children, $p(s) = v_i$ is necessarily a $\times$-gate, v_a its left child and v_b its right child. It follows that $M_L[a,i,i] = M_R[b,i,i] = 1$ and

$$m(p) = \prod_{l \in \mathsf{leaf}(\mathcal{T}_s)} \mathsf{label}(l) = \prod_{l \in \mathsf{leaf}(\mathcal{T}_{s_1})} \mathsf{label}(l) \prod_{l \in \mathsf{leaf}(\mathcal{T}_{s_2})} \mathsf{label}(l) = m(p_1)m(p_2).$$

- If p is not a partial parse tree then three cases can occur: If p_1 (resp. p_2) is not a partial parse tree, then $m(p_1) = 0$ (resp. $m(p_2) = 0$). If v_i is not a $\times$-gate, then $M_L[a,i,i] = 0$. Finally, if v_a (resp. v_b) is not the left (resp. right) child of v_i, then $M_L[a,i,i] = 0$ (resp. $M_R[b,i,i] = 0$). In all those cases, $M_L[a,i,i]M_R[b,i,i]m(p_1)m(p_2) = 0 = m(p)$.

This completes the proof. ◀

4 From $\{*\}$-formulas to arithmetic circuits

In this section we will show that the polynomials computed by polynomial size $\{*\}$-formulas can also be computed by polynomial size arithmetic circuits. We start by first proving this for formulas with bounded maximal dimension. Then we extend this result by showing that any $\{*\}$-formula can be transformed into an equivalent one with bounded maximal dimension without increasing the size.

4.1 Formulas with bounded maximal dimension

▶ **Proposition 9.** *Let F be a $\{*\}$-formula of maximal dimension k, dimension $l \le k$ and order $(n_1, \ldots, n_l)$. Let $n := \max_{T:T \text{ input of } F}(\mathsf{maxorder}(T))$. Then there exists a multiplicatively disjoint circuit C of size at most $2n^{k+1}|F|$ such that for all $\mathbf{e} \in D(F)$ there exists a gate $v_{\mathbf{e}}$ in C computing $F[\mathbf{e}]$.*

Proof. If F is an input, let C be the circuit having $\prod_{i=1}^{l} n_i$ inputs, each one labeled with an entry of F. The size of C is $\prod_{i=1}^{l} n_i \le n^k$.

If $F = G * H$, by induction we have circuits C_G and C_H with the desired properties for G and H. The dimension of F is less than k and for $\mathbf{e} \in D(F)$, $F[\mathbf{e}] = \sum_{i=1}^{m} G[\mathbf{e}_1, i]H[i, \mathbf{e}_2]$ with $m \le n$.

Each $G[\mathbf{e}_1, i]$ and $H[i, \mathbf{e}_2]$ is computed by a gate of C_G and C_H,respectively, so we can compute $F[\mathbf{e}]$ by adding at most $2n$ gates (m $\times$-gates and $m-1$ $+$-gates). As there are at most n^k entries in F, we can compute all of them with a circuit C by adding at most $2n \times n^k$ gates to $C_H \cup C_G$.

The circuit C is multiplicatively disjoint since each $\times$-gate receives one of its input from C_G and the other one from C_H. Also $|C| = |C_G| + |C_H| + 2n^{k+1} \le 2n^{k+1}|F|$. ◀

▶ **Corollary 10.** *Let (F_n) be a family of $\{*\}$-formulas of polynomial size and of maximal dimension k computing a family (f_n) of polynomials. Then (f_n) is in* VP.

4.2 Unbounded maximal dimension

Since the size of the circuit constructed in the previous section is exponential in $k := \mathrm{maxdim}(F)$, we cannot apply the results from there directly if k is not bounded by a constant. Somewhat surprisingly we will see in this section that one does not gain any expressivity by letting intermediate dimensions of formulas grow arbitrarily. Thus bounding $\mathrm{maxdim}(F)$ is not a restriction of the computational power of $\{*\}$-formulas.

▶ **Definition 11.** A $\{*\}$-formula F of dimension k and input dimension p is said to be *tame* if $\mathrm{maxdim}(F) \leq \max(k, p)$.

▶ **Definition 12.** A $\{*\}$-formula F is said to be *totally tame* if each subformula of F is tame.

Let us remark again that also in [3] and [6] there is a notion of tameness that prevents intermediate results from growing too much during the computation. It turns out that in those papers tameness plays a crucial role: Tame formulas can be evaluated efficiently while general formulas are hard to evaluate in the respective models. We will see that in our setting tameness is not a restriction at all. Indeed, any $\{*\}$-formula can be turned into an equivalent totally tame formula without any increase of its size. Thus totally tame and general formulas have the same expressive power in our setting which is a striking difference to the setting from [3] and [6]. We start with the following lemma:

▶ **Lemma 13.** *Let F be a totally tame formula with $\mathrm{dim}(F) = k$ and input dimension p. For all totally tame formulas E of dimension 1 and input dimension at most p, there exist totally tame formulas G_r and G_l of size $|F * E| = |E * F| = |F| + |E| + 1$ such that $G_r \simeq F * E$ and $G_l \simeq E * F$.*

Proof. We only show the construction of G_r; the construction of G_l is completely analogous. We proceed by induction on F.

If F is an input, then $\mathrm{maxdim}(F) = \mathrm{dim}(F) = p$. Let E be any totally tame formula of dimension 1 and input dimension at most p. We set $G_r := F*E$. Clearly, $k = \mathrm{dim}(G_r) = p-1$. Furthermore, $\mathrm{maxdim}(G_r) = \max(p-1, \mathrm{maxdim}(F), \mathrm{maxdim}(E)) \leq p$ because E has input dimension at most p and is totally tame. Thus G_r is totally tame.

Let now $F = F_1 * F_2$. Let $k_1 := \mathrm{dim}(F_1)$ and $k_2 := \mathrm{dim}(F_2)$. Let E be a totally tame formula of dimension 1 and input dimension at most p.

- If $\mathrm{dim}(F_2) = 1$, we claim that $G_r = F * E$ is totally tame. Indeed, since $\mathrm{dim}(F_2) = 1$, we have $\mathrm{dim}(F) = k_1 - 1$. But F is by assumption tame, so $k_1 = \mathrm{dim}(F_1) \leq \max(k_1 - 1, p)$. Hence $k_1 \leq p$ and $\mathrm{dim}(F) \leq p$. Thus all intermediate results of G_r have dimension at most p, so G_r is tame. But then it is also totally tame, because its subformulas are totally tame by assumption.
- If $\mathrm{dim}(F_2) = 2$, we have $p \geq 2$ obviously and $\mathrm{dim}(F) = \mathrm{dim}(F_1)$. F_2 is a subformula of F, so it is totally tame, too. Furthermore, $F_2 * E$ is of dimension 1 and it is also totally tame since $2 \leq p$. Moreover, by Proposition 2 we have $F * E \simeq F_1 * (F_2 * E)$. Applying the induction hypothesis on F_1 and $(F_2 * E)$ gives the desired G_r.
- If $\mathrm{dim}(F_2) > 2$, by Proposition 2 we have $F * E \simeq F_1 * (F_2 * E)$. We first apply the induction hypothesis on F_2 and E to construct a totally tame formula G' computing $F_2 * E$. Finally $G_r := F_1 * G'$ is totally tame since F_1 and G' are totally tame and F is of dimension $k = k_1 + k_2 - 2 \geq \max(k_1, k_2 - 1)$.

◀

We now prove the main proposition of this section.

▶ **Proposition 14.** *For every $\{*\}$-formula F there exists a totally tame $\{*\}$-formula F' such that $F' \simeq F$ and $|F| = |F'|$.*

Proof. The proof is by induction on F. If F is an input then it is trivially totally tame as the dimension of F is equal to the input dimension of F. So we set $F' := F$.

If $F = F_1 * F_2$ then several cases can occur depending on the dimension of F_1 and F_2. We denote by k, k_1, k_2 the dimensions of F, F_1 and F_2 respectively. We recall that $k = k_1 + k_2 - 2$.

- If both k_1 and k_2 are different from 1. Then $F' = F_1' * F_2'$ is totally tame since $k \geq \max(k_1, k_2)$
- If $k_1 = 1$ or $k_2 = 1$, we use Lemma 13 on F_1' and F_2' to construct F' of size $|F|$, totally tame, computing $F_1 * F_2$.

◀

Combining Proposition 14 and Corollary 10 we get the following theorem:

▶ **Theorem 15.** *Let (F_n) be a family of $\{*\}$-formulas of polynomial size and input dimension p (independent of n) computing a family of polynomials (f_n). Then (f_n) is in* VP*.*

Proof. Applying Proposition 14 on (F_n) gives a family (F_n') computing (f_n) such that (F_n') is tame. Then the maximal dimension of F_n' is p (because F_n is scalar, thus of dimension 1) and applying Corollary 10 proves the claim. ◀

4.3 Unbounded input dimension

While we got rid of the restriction on the maximum dimension of $\{*\}$-formulas in the last section, we still have a bound on the dimension of the inputs in Theorem 15. In this section we will show that this bound is not necessary to have containment of the computed polynomials in VP. We will show that inputs having "big" dimension can be computed by polynomial size $\{*\}$-formulas of input dimension 3. We can then use this to eliminate inputs of dimension more than 3 in $\{*\}$-formulas. Applying Theorem 15 we conclude that the only restriction on $\{*\}$-formulas that we need to ensure containment in VP is the polynomial size bound.

▶ **Proposition 16.** *Let T be a r-dimensional tensor of order $(n_1, \ldots, n_r)$. Let $L := \|T\| = \prod_{i=1}^{r} n_i$ be the number of entries in T. Then there is a $\{*\}$-formula F of size $r + 1 + L^3 + 2L$ and input dimension 3 computing T.*

Proof (sketch). Choose an arbitrary bijection $B : [L] \to [n_1] \times \ldots \times [n_r]$. Let furthermore $B_i : [L] \to [n_i]$ for $i \leq r$ be the projection of B onto the i-th coordinate. We define the 3-dimensional tensors T_i of order (L, n_i, L) by

$$T_1[m,k,n] = \begin{cases} T[B(m)] \text{ if } m = n \text{ and } B_1(m) = k \\ 0 \text{ otherwise.} \end{cases}$$

and, for $2 \leq i \leq r$,

$$T_i[m,k,n] = \begin{cases} 1 \text{ if } m = n \text{ and } B_i(m) = k \\ 0 \text{ otherwise.} \end{cases}$$

By induction one can show that for the tensor $P = T_1 * \ldots * T_r$, $P[m, k_1, \ldots, k_r, n] = T[k_1, \ldots, k_r]$ holds if $m = n$ and $B(m) = (k_1, \ldots, k_r)$ and $P[m, k_1, \ldots, k_r, n] = 0$ hold otherwise. Hence $T = E * P * E$ where E is a vector of order L filled with 1. ◀

The following theorem is a direct consequence of Proposition 16 and Theorem 15.

▶ **Theorem 17.** *Let (F_n) be a family of $\{*\}$-formulas of polynomial size computing a family of polynomials (f_n). Then f_n is in* VP*.*

5 The power of contracting with vectors

In this section we will make a finer examination of where exactly the additional power originates when going from iterated matrix product of [7] to tensor contractions. We will see that this additional expressivity crucially depends on the possiblity of contracting tensors on more than two of their dimensions. We will show that when we prevent this possibility by disallowing contractions with vectors – which are used in the proof of Theorem 6 to "collapse" dimensions not needed anymore so that we can access other dimensions to contract on – the expressivity of $\{*\}$-formulas drops to that of iterated matrix product.

Observe that we cannot assume that $\{*\}$-formulas compute scalars in this setting, because we cannot decrease the dimension of the tensors computed by a formula. Also we cannot compute all entries of the output at the same time efficiently, because there might be exponentially many such entries. But we will see in the following Proposition that we can compute each individual entry of the output more efficiently than in the general setting where contraction with tensors is allowed.

▶ **Proposition 18.** *Let F be a $\{*\}$-formula of order $(n_1, \dots, n_k)$ whose inputs are all of dimension at least 2. Then for all $\mathbf{e} \in D(F)$ there exists a skew arithmetic circuit C of size at most $2n^3|F|$ where $n := \max_{T\colon T \text{ input of } F}(\mathsf{maxorder}(T))$ computing $F[\mathbf{e}]$.*

Proof. By Proposition 2 we can write F as $A_1 * (A_2 * (A_3 * \dots * A_n))$. The proof then follows easily by induction: We do the same construction as in Theorem 10 but this time we only have n^2 entries and at each $*$-gate, one side is an input, resulting in a skew circuit. ◀

The case of Proposition 18 exactly corresponds to the characterization of $\mathsf{VP}_{\mathrm{ws}}$ by Malod and Portier [7] by n products of matrices of size $n \times n$. Thus Proposition 18 naturally generalizes this result and the real new power seen in Theorem 6 must come from the use of vectors in the products. As we have seen in the proof of Proposition 18 it is crucial that vectors are the only case which breaks the associativity of Proposition 2. So what looked like a not very important edge case in Observation 3 plays a surprisingly important role for the expressivity of $\{*\}$-formulas.

6 The $*_{i,j}$ operators

The $*$-operator contracts tensors only in a very specific way: It always only contracts on the last dimension of one tensor and the first dimension of the other one. It is thus natural to ask if this is a restriction of the computational power of the formulas. In this section we will see that it is indeed not. If we allow free choice of the dimensions to contract on during a contraction this does not make the resulting polynomials harder to compute. To formalize this we give the following definition of a contraction $*_{i,j}$.

▶ **Definition 19.** Let T be a k-dimensional tensor of order $(n_1, \dots, n_k)$ and G a l-dimensional tensor of order $(m_1, \dots, m_l)$ with $k, l \geq 1$. When $n_i = m_j$ for $i \leq k$ and $j \leq l$, we denote by $T *_{i,j} G$ the contraction of T and G on the dimensions i and j the $(k + l - 2)$-dimensional tensor of order $(n_1, \dots, n_{i-1}, n_{i+1}, \dots, n_k, m_1, \dots, m_{j-1}, m_{j+1}, \dots, m_l)$ defined as

$$(T *_{i,j} G)[\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{e}_4] = \sum_{r=1}^{n_i} T[\mathbf{e}_1, r, \mathbf{e}_2] G[\mathbf{e}_3, r, \mathbf{e}_4]$$

for all $\mathbf{e}_1 \in [n_1] \times \dots \times [n_{i-1}]$, $\mathbf{e}_2 \in [n_{i+1}] \times \dots \times [n_k]$, $\mathbf{e}_3 \in [m_1] \times \dots \times [m_{j-1}]$ and $\mathbf{e}_4 \in [m_{j+1}] \times \dots \times [m_l]$.

$\{*_{i,j}\}$*-formulas* are defined in complete analogy to $\{*\}$-formulas.

It turns out that $\{*_{i,j}\}$-formulas cannot compute more than $\{*\}$-formulas, so the free choice of the dimensions to meld on does not change much.

▶ **Theorem 20.** *Let (F_n) be a family of $\{*_{i,j}\}$-formulas of polynomial size computing a family of polynomials (f_n). Then (f_n) is in* VP*.*

The proof of Theorem 20 follows a similar approach as that of Theorem 17. Let us sketch some key steps here: If we bound the maximal dimension of $\{*_{i,j}\}$-formulas by a constant k, it is easy to see that the proof of Theorem 10 can be adapted to $\{*_{i,j}\}$-formulas in a straightforward way. The main complication is then turning general $\{*_{i,j}\}$-formulas into totally tame ones. $*_{i,j}$ is not associative anymore, and this makes a straightforward translation of the proof of Proposition 14 tricky. These problems can be solved by the observation that the crucial steps in the process of making a formula tame are those where a $\{*_{i,j}\}$-formula is multiplied by a tensor of dimension 1. But for such contractions we can give explicit formulas for different cases that may occur, so again every $\{*_{i,j}\}$-formula has an equivalent tame $\{*_{i,j}\}$-formula. Combining this with Proposition 16 completes the proof.

7 Conclusion

We have shown that one can get a robust characterization of VP by formulas with tensors as input and tensor contraction as the only operation. This generalizes the known characterization of $\mathsf{VP}_{\mathrm{ws}}$ by iterated matrix product by Malod and Portier [7]. In some aspects the situation in our setting is more subtle, though. We remarked that vectors and in general breaking associativity plays a crucial role if we want to characterize VP. Also, unlike for iterated matrix product we have to make a choice if we take $*_{i,j}$ or $*$ as our basic operation. It is easy to check that using the equivalence to $*_{i,j}$ for matrix product would merely be transposing the matrix, so it clearly does not change the expressivity of the model. But fortunately also in our setting, the choice of $*_{i,j}$ or $*$ does not influence the complexity of the computed polynomials.

Unfortunately, unlike for iterated matrix product our characterization seemingly does not directly lead to a characterization of VP by something similar to branching programs. We still think that such a characterization is highly desirable, because the branching program characterization of $\mathsf{VP}_{\mathrm{ws}}$ has been the source of important insights in arithmetic circuit complexity. Thus we believe that a similar characterization of VP might lead to a better understanding of VP, a class that is arguably not very well understood, yet.

Let us quickly discuss several extensions to the results in this paper that we had to leave out for lack of space: First, analyzing the proofs of Section 4 a little more carefully one can see that our results remain true if one does not measure the size of a tensor as the number of its entries but as the number of its *nonzero* entries. This makes it possible to allow inputs of large dimension and large order.

Also, it seems plausible and straightforward to generalize our results to arbitrary semi-rings in the style of Damm, Holzer and McKenzie [3]. Choosing different semi-rings one would then probably get characterizations of classes like LOGCFL and its counting, mod-counting and gap-versions. The main new consideration would be the treatment of uniformity in these settings which appears to be possible with a more refined analysis of our proofs.

Finally, for tensors there are other natural operations to perform on them like addition or tensor product. It is natural to ask, if adding such operations does change the complexity of the resulting polynomials. While it is straightforward to see that adding only tensor product as an operation does not increase the expressivity of $\{*\}$-formulas, we could so far not answer the corresponding question for addition. Therefore, we leave this as an open question.

Acknowledgements

We thank Yann Strozecki for a detailed and helpful feedback on an early version of this paper. We also thank Hervé Fournier, Guillaume Malod and Sylvain Perifel for helpful discussions.

References

1 Martin Beaudry and Markus Holzer. The complexity of tensor circuit evaluation. *Computational Complexity*, 16(1):60–111, 2007.

2 P. Bürgisser. *Completeness and reduction in algebraic complexity theory*, volume 7. Springer Verlag, 2000.

3 C. Damm, M. Holzer, and P. McKenzie. The complexity of tensor calculus. *Computational Complexity*, 11(1-2):54–89, 2002.

4 A. Durand and S. Mengel. The Complexity of Weighted Counting for Acyclic Conjunctive Queries. *Arxiv preprint arXiv:1110.4201*, 2011.

5 P. Koiran. Arithmetic circuits: The chasm at depth four gets wider. *Theoretical Computer Science*, 448(0):56 – 65, 2012.

6 G. Malod. Circuits arithmétiques et calculs tensoriels. *Journal of the Institute of Mathematics of Jussieu*, 7:869–893, 2005.

7 G. Malod and N. Portier. Characterizing Valiant's algebraic complexity classes. *Journal of complexity*, 24(1):16–38, 2008.

8 M. Marcus. *Finite dimensional multilinear algebra*, volume 1. M. Dekker, 1973.

9 S. Mengel. Characterizing Arithmetic Circuit Classes by Constraint Satisfaction Problems - (Extended Abstract). In *ICALP (1)*, pages 700–711, 2011.

10 N. Nisan. Lower bounds for non-commutative computation. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 410–418. ACM, 1991.

11 L.G. Valiant. Completeness classes in algebra. In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 249–261. ACM, 1979.

12 L.G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. Fast parallel computation of polynomials using few processors. *SIAM Journal on Computing*, 12:641, 1983.

Search versus Decision for Election Manipulation Problems*

Edith Hemaspaandra[1], Lane A. Hemaspaandra[2], and Curtis Menton[2]

1 **Department of Computer Science, RIT, Rochester, NY 14623, USA**
2 **Dept. of Computer Science, Univ. of Rochester, Rochester, NY 14627, USA**

Abstract

Most theoretical definitions about the complexity of manipulating elections focus on the decision problem of recognizing which instances can be successfully manipulated, rather than the search problem of finding the successful manipulative actions. Since the latter is a far more natural goal for manipulators, that definitional focus may be misguided if these two complexities can differ. Our main result is that they probably do differ: If integer factoring is hard, then for election manipulation, election bribery, and some types of election control, there are election systems for which recognizing which instances can be successfully manipulated is in polynomial time but producing the successful manipulations cannot be done in polynomial time.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes; F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Search vs. decision; application of structural complexity theory

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.377

1 Introduction

Elections are such a ubiquitous model for human and electronic collective decision-making—and during the past few decades, with the rise of computers, multiagent systems, and the internet, elections have become important even in many "modern" challenges such as collaborative filtering/recommender systems, planning, and reducing web spam—that much work has been devoted to studying how to manipulate elections. However, the broad stream of theoretical work on the computational complexity of manipulative attacks on elections (see the surveys [15, 12]) is largely centered on the complexity of the decision versions: Given an instance, determining whether there exists a successful manipulation (typically, ensuring that a given candidate wins, or ensuring that a given candidate does not win) of the given sort.

As a running example that we will use in this introduction, consider unweighted noncoalition (i.e., a single manipulator) manipulation, which was central in one of the seminal papers on manipulation ([1], see also [2]). For this problem, relative to some fixed election system, the inputs are the candidate set, the voter set consisting of a collection of nonmanipulative voters (whose preferences are each typically expressed by each voter as a preference ballot, e.g., Gore > Nader > Bush), and a single manipulative voter who has not yet set her vote but who has a "preferred" candidate p. And the question is: Does there exist a preference

* Supported by grants NSF-CCF-{0915792,1101452,1101479} and ARC-DP110101792, and two Humboldt Foundation Bessel Research Awards. Work done in part while visiting the University of Düsseldorf.

(vote) the manipulative voter can cast that will make p win the election? This is typically viewed as a decision (language) problem, namely, as the set of all instances for which the answer to that question is "Yes."

Of course, what a manipulator might most want is not to know a successful manipulation *exists* (a decision problem), but rather to know what *specific action* (what vote, bribe, etc.) to take to achieve success (a search problem). For the case of our unweighted noncoalition manipulation example, the search version would be a function that takes the same input as the decision version but then either outputs that no successful strategic vote for the manipulative voter exists or, if a successful vote does exist, *outputs a successful vote—one that makes p win.*

This paper studies whether these two goals' achievability can differ: whether decision versions of election problems can be easy yet their search versions intractable.

Virtually all papers in this area, to prove polynomial-time results for deciding when manipulative actions can succeed, actually give polynomial-time algorithms to produce the successful action. So one might suspect that perhaps that is always the case. For manipulation, bribery, and some types of control, we prove otherwise, under a complexity-theoretic hypothesis that is widely believed true. Our main contributions are:

- If $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$, then for each of manipulation (including in particular the case of our running example, unweighted noncoalition manipulation), bribery, and certain types of partition-control, there exist election systems for which there are polynomial-time algorithms to determine whether each given instance has a successful manipulative action, but no polynomial-time algorithm can exist that given an instance that is manipulable provides the successful manipulation. (It is widely believed in cryptography that integer factoring is hard. It is well known that if integer factoring is hard then $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$.) Informally put, the situation is that the frustrated world of polynomial-time computation will have to say things such as, "I can totally guarantee you that there are strategic votes you can cast to make Barack Obama win in the given electoral setting, but I have no idea what those votes are." We show that this bizarre setting can even occur in extremely simple cases, such as unweighted noncoalition (i.e., where we have just a single manipulative voter) manipulation. It follows immediately from our results that if $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ then for each of the above-mentioned manipulative actions there exists an election system in which the search problem does not polynomial-time Turing reduce to the decision problem.

  To the best of our knowledge, this is the first result separating, even conditionally, search from decision in the setting of computational social choice (see [7, 26])—an area whose core definitions on election manipulative-actions, which date back twenty years, are framed in terms of decision problems.

  To the best of our knowledge, our proof is the first time the complexity-theoretic Borodin-Demers Theorem, from the 1970s, has found application in an applied domain.
- In contrast, we show that for all the standard types of election-control actions based on adding or deleting voters or candidates, and for some of the standard election-control actions based on partitioning, the search problem (finding how to succeed) polynomial-time Turing reduces to the decision problem (knowing when one can succeed). It follows that, for these manipulative actions and for every election system, the bizarre type of behavior mentioned earlier cannot occur: Easy recognition of instances where success is possible implies polynomial-time algorithms for how to achieve success.

  While proving this, we notice that two pairs of control attacks assumed to differ in fact are identical problems, namely, for every election system, destructive control by partition of candidates and destructive control by run-off partition of candidates are the same set in both the standard tie-breaking cases (ties-eliminate and ties-promote); this reduces by two the number of distinct, standard control types. This is the first collapse of standard control types that we are aware of.

- Regarding the results of the first bullet point above, which, when $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ make decision easy but search hard, one might worry that search may be only infrequently hard. We address this by, as Theorem 9, constructing manipulative-action problems whose search versions are *just as often hard as are those problems in* $\mathrm{NP} \cap \mathrm{coNP}$ *that have the highest density of hardness*, give or take a slight degree of flexibility. These results provide a transference of density-of-hardness from a class to a particular type of concrete problem.

Given that $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ suffices to for some systems make the natural problem to care about (the search version) hard even as the problem that has been the theoretical literature's central definition (the decision version) declares the problem easy, we suggest that in definition and problem framing it may now be good to more energetically stress the importance of the search versions of election manipulation problems.

2 Preliminaries

An election will consist of a set C of candidates and a (multi)set V of voters (who for us will be given just as their preferences). For all the cases discussed in this paper, each voter's preferences will be a tie-free linear ordering of the candidates. We assume each vote is input distinctly (i.e., the voters' preferences come in as separate ballots; but it would be cheating for us to use the order of that input list within our proofs). So a typical election might be $C = \{\mathrm{Alice}, \mathrm{Bob}, \mathrm{Carol}\}$ and the voter (multi)set might be $V = \{(\mathrm{Carol} > \mathrm{Bob} > \mathrm{Alice}), (\mathrm{Bob} > \mathrm{Alice} > \mathrm{Carol})\}$. Most familiar election systems, such as plurality-rule elections, don't care about voter names; our constructions never need to use voter names, and so like most papers we don't have voter names in V.

Election systems, or voting systems, map from an election instance (C, V) to a set of winners (i.e., to a set W, $\emptyset \subseteq W \subseteq C$). (Pure social choice papers often definitionally exclude the case $W = \emptyset$, but like most papers on computational social choice we allow it.)

For each fixed election system $\mathcal{E}$, one can define the election winner problem as follows (see [3]).

Name: $\mathcal{E}$-winner, or the winner problem for $\mathcal{E}$.
Given: Election (C, V) and candidate $p \in C$.
Question: Is p a winner of the election (C, V) under election system $\mathcal{E}$?

This is actually, in the way universally accepted in computer science, describing a set, i.e., a language. That set is the set of all triples $\langle C, V, p\rangle$ such that the answer to the question is "Yes."

We now briefly present the key definitions for the three most commonly studied types of manipulative actions: manipulation, bribery, and control. These three types were first studied, respectively, by Bartholdi, Orlin, Tovey, and Trick [2,1], Faliszewski, Hemaspaandra, and Hemaspaandra [11], and Bartholdi, Tovey, and Trick [4] for the "constructive" cases, i.e., where the goal is to make a particular candidate be a winner.[1] The "destructive" cases,

[1] We say "be a winner" as this entire paper will focus on that notion, known as the nonunique-winner

where the goal is to ensure that a particular candidate is not a winner, were introduced by Conitzer, Sandholm, and Lang [8] for manipulation, by Faliszewski, Hemaspaandra and Hemaspaandra for [11] for bribery, and by Hemaspaandra, Hemaspaandra and Rothe [20] for control.

The manipulation problem is defined as follows, and models whether a coalition of strategic voters can make a certain candidate win.

Name: $\mathcal{E}$-unweighted coalition manipulation, or the unweighted coalition manipulation problem for $\mathcal{E}$; for short, the manipulation problem for $\mathcal{E}$.

Given: Candidate set C, nonmanipulative voter set V_1 (as a collection of preference ballots each with preferences over C), manipulative voter set V_2 (since we don't have names, this will be input as a unary string, 1^k, to indicate the number, k, of manipulative voters), and a candidate $p \in C$.

Question: Is there some choice of preferences for the manipulative voters such that p is a winner in the election in system $\mathcal{E}$ with candidates C and with both the nonmanipulative and the manipulative voters voting?

This again is a decision problem consisting of the set of all inputs yielding the answer "Yes." However, there is a very natural search problem associated with this, which we will call manipulation search, i.e., finding the successful action. In particular, a function f solves the manipulation search problem (for a given election system) if on all inputs where the Question's answer is "No" (i.e., all inputs not in the set that is the decision version) the function indicates in some clear way (e.g., by outputting -1) that manipulation is not possible, and on each input that belongs to the decision version, f specifies settings to the preferences of the manipulative voters in such a way that those result in p being a winner in the election $(C, V_1 \cup V_2)$. If some solution for the manipulation search problem is a polynomial-time computable function, we will say that the manipulation search problem is polynomial-time computable.

One can also define "weighted" coalition manipulation, where each manipulative and nonmanipulative voter has a weight (how many times her vote counts). Our results on manipulation all will hold for that case too. But it is more interesting that the results hold even in the unweighted case—and indeed, our proofs establish that they hold even when the number of manipulative voters is limited to being at most one.

Unlike manipulation, in bribery all voters have initial preferences. In the simplest model of bribery, voters are unweighted and each has unit cost to bribe. (By varying these parameters, [11] obtained three other models: unweighted, priced; weighted, unpriced; and weighted, priced. Our results on bribery hold in all four models.) This problem models whether having the ability to reshape (bribe) the preferences of a number of voters allows one to make a given candidate win.

Name: $\mathcal{E}$-bribery, or the bribery problem for $\mathcal{E}$.

Given: Election (C, V), candidate $p \in C$, and integer $b \geq 0$.

Question: Does there exist some collection of at most b voters, and a way of setting their votes, so that in the election under $\mathcal{E}$ in which those votes are thus set and the other voters cast the votes the input specified for them, p is a winner?

model (i.e., allowing ties). That model has broadly been the one previous papers favored (except the earliest work on control, but we feel that for control too this model is the more natural one).

Again, this is and should be viewed as a decision problem—as a set. It has the natural search version, which we will call bribery search.

Finally, we come to election control, the most varied, the most difficult to describe, but in our opinion the most interesting of the three most studied types of manipulative attacks on elections. Control asks whether by various adjustments to the participation and structure of an election, a given candidate can be made a winner. A natural set of control actions was specified in [4], the seminal paper on control, and we adopt that set, very slightly modified—as is now done in most papers—to treat adding of candidates symmetrically with the other add/delete types (as suggested by [14]) and to be clear in the "partition" cases about how first-round ties are handled (following [20]). Those control types are adding candidates, deleting candidates, adding voters, deleting voters, partition of voters, run-off partition of candidates, and partition of candidates. These loosely model many real-life settings, ranging from get-out-the-vote drives, to voter suppression, to having a culling "primary" round, to encouraging (or discouraging) "spoiler" candidates (see [14] for discussion of how these model various real-life scenarios). Each of the three partition control types is two control types—one (denoted by a TP—"ties promote"—modifier) for the model in which if a first-round election has multiple winners they all move forward to the second round, and one (denoted by a TE—"ties eliminate"—modifier) for the model in which one moves forward from a first-round election only if one is the unique winner of that contest.

Due to space limitations, we define here only the control type for which we include a proof sketch of our main result. The other control types are each defined in the intuitively natural way, and their full definitions can be found in the TR version [19], *which also contains proofs of all our results.*

▶ **Definition 1.** Let $\mathcal{E}$ be an election system. In the control by run-off partition of candidates problem for $\mathcal{E}$, in the TP or TE tie-handling rule model, we are given an election (C, V) and a candidate $p \in C$. Is there a partition of C into C_1 and C_2 such that p is a winner of the two-stage election where the winners of subelection (C_1, V) that survive the tie-handling rule compete against the winners of subelection (C_2, V) that survive the tie-handling rule? Each subelection (in both stages) is conducted using election system $\mathcal{E}$.[2]

Control problems are decision problems, i.e., sets. And they have the obvious search versions, which we will refer to in ways analogous to those we mentioned earlier regarding manipulation.

All the manipulation, bribery, and control problems defined so far are about trying to make a certain candidate be a winner. We will henceforward when mentioning these problems always add the word "constructive," to indicate that the problem is about making the specified candidate be a winner. As alluded to earlier, for every problem we have defined there is a "destructive" version, where the question is whether one can ensure that the specified candidate is not a winner. Both the constructive and destructive problems have both decision and search versions, in the obvious way.

A (decision or search) problem A is said to polynomial-time Turing reduce ($\leq_T^p$-reduce) to a decision problem B if there is a machine M such that (a) M^B runs in polynomial time (relative to the length of its input), and (b) if A is a decision problem then the language

[2] When speaking of an election, (C', V'), we always implicitly mean that each vote in V' is passed to the election system only as the version of itself restricted to the candidates in C'. This is the normal approach in defining control types, but we stress it because if we did not follow this approach, we might cheat in some of our constructions and use parts of a vote regarding candidates not in the election to pass/control information.

Table 1 Results summary. Key: "S $\leq$ D" is shorthand for: For each election system $\mathcal{E}$, the named constructive or destructive manipulative action has the property that its search version polynomial-time Turing reduces to its decision version. (Note that this implies that it is impossible for its decision version to be polynomial-time computable but its search version not to be polynomial-time computable.) "S $\not\leq$ D" is shorthand for: If $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$, then there exists an election system $\mathcal{E}$, having a polynomial-time winner problem, such that the named constructive or destructive manipulative action's decision problem is in polynomial time but its search problem is not in polynomial time. (Note that this implies that if $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$, then there is an election system $\mathcal{E}$ such that for the named constructive or destructive manipulative action, search does not polynomial-time Turing reduce to decision.)

Manipulative Action	Constructive	Destructive
bribery	S $\not\leq$ D	S $\not\leq$ D
control by adding voters	S $\leq$ D	S $\leq$ D
control by deleting voters	S $\leq$ D	S $\leq$ D
control by partition of voters, ties promote	S $\not\leq$ D	S $\not\leq$ D
control by partition of voters, ties eliminate	S $\not\leq$ D	S $\not\leq$ D
control by adding candidates	S $\leq$ D	S $\leq$ D
control by deleting candidates	S $\leq$ D	S $\leq$ D
control by partition of candidates, ties promote	S $\not\leq$ D	S $\leq$ D
control by partition of candidates, ties eliminate	S $\not\leq$ D	S $\leq$ D
control by run-off partition of candidates, ties promote	S $\not\leq$ D	S $\leq$ D
control by run-off partition of candidates, ties eliminate	S $\not\leq$ D	S $\leq$ D
control by unlimited adding of candidates	S $\leq$ D	S $\leq$ D
manipulation	S $\not\leq$ D	S $\not\leq$ D

accepted by M^B is A, and if A is a search problem then M^B computes a function that is a solution of the search problem (M^B means machine M given a unit-cost subroutine testing membership in B); this is the standard definition of polynomial-time Turing reductions, which along with polynomial-time many-one reductions are the central ways computer science links and compares the complexity of problems. For example, if we say that $\mathcal{E}$-manipulation search polynomial-time Turing reduces to $\mathcal{E}$-manipulation, that means that given an instance of the $\mathcal{E}$-manipulation problem (but being interested in getting an action, i.e., we are doing the search version), we can in polynomial time, given access to an oracle for the set $\mathcal{E}$-manipulation, correctly either state that successful manipulation is impossible or output a successful manipulation. We move directly on to the presentation of our results, and then provide a discussion of related work.

3 Results

The tightly related goals of this paper are to determine for which manipulative actions

(a) for all election systems, search (polynomial-time Turing) reduces to decision,

and to determine for which manipulative actions

(b) there exists some election system $\mathcal{E}$, whose winner problem is in P, for which the decision version of the manipulative action is in P yet the search version of the decision problem is not polynomial-time computable (i.e., no polynomial-time function solves the search version).

These are related, as "(a)" implies "NOT (b)."

For manipulation, bribery, and every standard type of control, we in effect strongly resolve this. That is, for some, we prove (a)—which of course implies NOT (b) (in fact, it implies even that "NOT (b′)," where (b′) is (b) with the "winner problem in P" requirement removed). And for all the others we prove, under the complexity-theoretic assumption $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$, that (b) holds—which of course implies NOT (a). The more striking group of cases is the latter collection—manipulative actions for which for some election system with an easy (i.e., polynomial-time) winner problem we can easily (i.e., in polynomial time) for a given setting determine whether a successful attack exists, and yet there can exist no polynomial-time algorithm to always tell us what the successful attack action (that we know exists!) is.

In the process of proving the latter group of cases we will do even more than promised above. We will not only show that $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ implies (b), but we also will characterize (b), for each of those manipulative actions, as being equivalent to the right-hand side condition of the so-called Borodin-Demers Theorem from computational complexity theory (i.e., the "then" part of Theorem 4 below). So, although we need a rich variety of complex election schemes and tricky coding schemes to prove our results, from those results and that work we establish that twelve different instances of whether (b) holds are, deep down, the same issue.

In the process of proving the other group of cases we will note that two pairs of control types that have always been viewed as distinct in fact pairwise collapse: viewed as sets, they are the exact same set. So all previous papers that gave separate proofs for the two elements of a collapsing pair were proving the same result twice. To be fair to earlier papers it is important to mention that of the two pairs that we show to collapse (in the nonunique winner model), only one of those pairs collapses in the unique winner model; that itself is also a new result.

3.1 Cases When the Manipulative-Action Decision Problem Is Easy but Its Search Problem Is Hard

Our main result, showing that if $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ then there are easy election systems (i.e., having a polynomial-time winner problem) whose manipulative-action decision problem is easy but whose manipulative-action search problem is hard, is the following.

▶ **Theorem 2.** *If $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$, then for each manipulative action $\mathcal{A}$ marked "$S \not\leq D$" in Table 1, there exists an election system $\mathcal{E}$ (which may differ based on $\mathcal{A}$), whose winner problem is in polynomial time, such that the $\mathcal{A}$-decision problem for $\mathcal{E}$ is in* P *but the $\mathcal{A}$-search problem for $\mathcal{E}$ is not polynomial-time computable.*

▶ **Corollary 3.** *If $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$, then for each of the manipulative actions $\mathcal{A}$ covered by Theorem 2, $\mathcal{A}$-search for $\mathcal{E}$ does not polynomial-time Turing reduce to $\mathcal{A}$-decision for $\mathcal{E}$.*

Let us present the idea behind the proof of Theorem 2, focusing in particular as an example on constructive control by run-off partition of candidates in the ties-promote model. So, let us use the statement's hypothesis, and assume that $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ holds. We invoke a complexity-theoretic result known as the Borodin-Demers Theorem. To the best of our knowledge, the Borodin-Demers Theorem has never before been applied in the study of elections, computational social choice, multiagent systems, or for that matter anywhere outside of computational complexity theory.

▶ **Theorem 4** ([6], see [18, 25] for the form used here). *If $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ then there is a set B so (1) $B \in \mathrm{P}$, (2) $B \subseteq \mathrm{SAT}$, and (3) no* P *machine can find solutions for all formulas in*

B (that is, for no *polynomial-time computable function g do we have $(\forall f)[f \in B \Rightarrow g(f)$ is a satisfying assignment of $f])$.*

So we have something quite striking: A set of boolean formulas that are easily recognized as being satisfiable but for which it is not in general easy to find how they can be satisfied, i.e., every polynomial-time machine fails on some of them (indeed, on infinitely many, as otherwise one could finitely patch). (The Borodin-Demers Theorem certainly does *not* say that if $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ then search does not reduce to decision for SAT; it is well-known that for SAT—and indeed for any NP-complete problem—search $\leq_T^p$-reduces to decision. However, we will use Borodin-Demers as a tool to show that in certain election settings search does not reduce to decision if $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$.) Our goal, of course, is to shoehorn the set B into the world of election manipulation for a variety of manipulative actions. Of course, each manipulative action comes with its own form and definition, and so for many such shoehorning is essentially impossible—as we show in Section 3.2. But for others, we can do this, sometimes smoothly and sometimes through extreme, difficult contortions. The difficulty is that the structure of many electoral manipulations, and our goal to realize a separation with respect even to some election system with a polynomial-time winner problem, very much ties our hands. And in fact, even for our results here, the different manipulative actions have enormously differing proofs, as each proof must be tailored to the manipulative action.

Nonetheless, the general approach is clear and shared, although the implementations and constructions differ wildly. The general approach is given a set B from the Borodin-Demers result, we must build an election system $\mathcal{E}$, whose winner problem is in P, such that for our manipulative action the decision problem is in P but the search problem is not polynomial-time computable. To do this, our election system $\mathcal{E}$ will clearly need to be very much attuned to B. It typically will be interpreting voters, candidates, collections of voters, and collections of candidates as variously trying to specify a Borodin-Demers "puzzle"—i.e., an obviously satisfiable formula (a string $x \in B$), and it also will interpret some similar things about its input as trying to propose solutions to that puzzle.

To really explain how this works in practice would require going through the actual proofs (which we provide in [19]). But to give an idea of the flavor, let us speak here in a high-level, handwaving way about a specific example (that is neither our hardest nor our easiest case), namely constructive control by run-off partition of candidates in the ties-promote model.

Proof (sketch, for the just-mentioned case): Our scheme here is to hope—although other inputs won't trip us up—that our input consists of two almost-copies of a Borodin-Demers puzzle x, namely that part of our input is $x0$ and $x1$, $x \in B$. In particular, we'll hope that the lexicographically two smallest candidates have those strings as their names. Suppose that the obviously satisfiable formula x (for concreteness of this sketch) is 1000 bits long and has 27 variables. Then we will hope to have exactly $2 \cdot 27 = 54$ other candidates, who will all form a lexicographically contiguous segment starting at, say $0^{5 \cdot 1000}$, i.e., the first of the 54 candidates is named 0^{5000}, the second is named $0^{4999}1$, the third is named $0^{4998}10$, and so on. Now, we'll interpret these strings as 27 pairs—the first two, the next two, and so on. And we'll set up our election system so that it will try to ensure that exactly one of each pair goes on one side of the partition in any partition that will lead to victory of $x0$. The election system if it sees in its candidate set $x0$, $x \in B$, will compute the size and number of variables of x, will see if it has the right collection of other candidates to indicate it has precisely one from each of the 27 pairs, will then interpret the low-order bit of each of those pair-choices as the ith bit of a guessed satisfying assignment for x, and if that assignment does satisfy x, will make $x0$ the one and only winner. Also, the election system when its input contains $x1$, $x \in B$, will check that it also has precisely one candidate from each of the 27 pairs (and no candidates other than those and $x1$), and if so $x1$ and only $x1$ will win—it does not in this case do any satisfiability check. A third and final case in which we will have a winner is if the candidate set is $\{x0, x1\}$, $x \in B$, in which case $x0$ and only $x0$ will win. And these three cases are the only ways to win.

Now, recall that run-off partition splits the candidates into two groups for primary elections and then runs the winners of those against each other. If the input set is of just the dream-case form we have described, and we ask whether $x0$ can by run-off partition, ties-promote, be made a winner of the overall election, the answer is obviously "Yes," as $x \in B$ is satisfiable and so the partition that puts into one side of the partition $x0$ and precisely a set of one-per-pair candidates encoding a satisfying assignment and puts the rest on the other side will have $x0$ win its first-round contest, will have $x1$ win its first-round contest, and will have $x0$ win the second-round contest between $x0$ and $x1$.

But it is possible to see that if we have a polynomial-time algorithm for the search problem of how to make $x0$ win, that on the special input we just described, any search-problem output, i.e., any successful partition, will immediately make clear a satisfying assignment of x, as the election system in fact will force that. So if we had a search-problem polynomial-time algorithm, the third property (the one about no FP function always yielding solutions) of the Borodin-Demers set B would be violated. So search for our election system is not polynomial-time computable.

But our election system clearly does have a P winner problem—it is just three simple cases to check. So all that remains is to show that the decision problem for this control type is in P. Note that we need a P algorithm that works for *all* inputs—not just inputs so nice as to have our dream-case format. However, when one carefully checks everything, with the system very clearly specified, one can see that this holds also (see our full version of this paper [19]). This is the part that causes a large part of the complexity of the election system; for example, the simpler system without the $x1$ requirement will fail this requirement. ❑

Now, Theorem 2 gives twelve cases where $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ implies the existence of a P-winner problem election system where for a particular manipulative action decision is easy but search is hard. It is natural to wonder whether the converses of some or all of these twelve results hold. We note that either all of the converses hold or none do, and which of those cases holds is identical to a long-open issue in complexity theory, namely, whether the converse of the Borodin-Demers Theorem holds. Let us call the three-part right-hand side of the Borodin-Demers Theorem the "Borodin-Demers Condition." We claim the following result.

▶ **Theorem 5.** *For each of the twelve manipulative actions $\mathcal{A}$ referred to in Theorem 2, the following two conditions are equivalent:*

- *The Borodin-Demers Condition holds.*
- *There exists an election system $\mathcal{E}$, with a polynomial-time winner problem, such that the $\mathcal{A}$ decision problem for $\mathcal{E}$ is in* P *but the $\mathcal{A}$ search problem for $\mathcal{E}$ is not polynomial-time computable.*

Finally, one might worry, given the broad interest recently in how often NP-hard election manipulation problems are hard (e.g., [17]), that although Theorem 2's conclusion says that decision is easy while search is hard, the hardness for search that it speaks of is a worst-case notion of hardness, and so perhaps the hard instances form a very sparse set. This is a natural worry, but to partially address it we will as Theorem 9 prove that if even one set A in $\mathrm{NP} \cap \mathrm{coNP}$ is frequently hard, then all of our search cases are (in a certain sense) almost as frequently hard as that set A.

3.2 Cases Where Search Reduces to Decision

This section's main result states that for many manipulative actions search polynomial-time Turing reduces to decision.

▶ **Theorem 6.** *For each manipulative action $\mathcal{A}$ marked "$S \leq D$" in Table 1, and for each election system $\mathcal{E}$, the $\mathcal{A}$ search problem for $\mathcal{E}$ polynomial-time Turing reduces to the $\mathcal{A}$ decision problem for $\mathcal{E}$.*

Thus the behavior displayed in Theorem 2 is impossible for all of the above manipulative actions, even if Theorem 2's "winner problem in P" requirement is dropped.

▶ **Corollary 7.** *For each of the manipulative actions $\mathcal{A}$ referred to in Theorem 6, for no election system $\mathcal{E}$ can it be the case that the $\mathcal{A}$ decision problem for $\mathcal{E}$ is in* P *but the $\mathcal{A}$ search problem for $\mathcal{E}$ is not polynomial-time computable.*

The proofs can be found in [19]. But we mention that important to establishing the four cases with the most interesting proofs, and an interesting result in its own right, is the following. We show that two pairs of control types which in previous papers have been assumed to be distinct, are in fact identical.

▶ **Theorem 8.** *1.* DC-RPC-TP = DC-PC-TP *(i.e., viewed as decision problems, destructive control by run-off partition of candidates in the ties-promote model is* exactly *the same problem—the same set—as is destructive control by partition of candidates in the ties-promote model).*
2. DC-RPC-TE = DC-PC-TE*.*

4 Related Work, Frequency of Hardness, and Open Directions

Although to the best of our knowledge search versus decision has not previously been a focus area in the long line of work on the complexity of manipulative attacks, the detailed analysis of the complexity of attacks on particular systems has been a focus area. For example, detailed classifications of the complexities of constructive and destructive control actions on specific systems can be found in such work as [14, 10, 9, 5, 24]. These papers are about specific systems. In contrast, our "search reduces to decision" results hold for all systems. Our "if P $\neq$ NP $\cap$ coNP" results on the other hand use that complexity-theoretic hypothesis to build specific systems that make decision easy while making search hard.

Existing papers that give polynomial-time attack algorithms against specific systems typically do so by (at least implicitly) finding a polynomial-time solution to the search problem. Probably the definition most related to the interests of this paper is the definition of "certifiably vulnerable" of Hemaspaandra, Hemaspaandra, and Rothe [20], which captures the notion of demanding that an attack provide a successful action when one exists. That paper actually adds an "optimality" twist to that notion, but subsequent papers (e.g., [13, 16]) when using the notion of certifiability take it to mean providing some successful action when one exists, rather than the "smallest in size/effort/cost" such action.

Our search reduces to decision results of course hold on all inputs. But our "P $\neq$ NP $\cap$ coNP"-induced results put decision versions in P while ensuring that their search versions are not polynomial-time computable. That latter part is a worst-case claim. However, by a detailed look at the properties of (and length-stretching in, and injectivity of reductions related to) the proofs of both the Borodin-Demers Theorem and Theorem 2, we can prove (see [19]) the following result, that says that *we can construct manipulative-action problems* *within Theorem 2 whose search versions are just as often hard as are those problems in* $\mathrm{NP} \cap \mathrm{coNP}$ *(such as, potentially, problems related to factoring) that have the highest density of hardness*, give or take an ϵ of flexibility. Speaking more broadly, although our paper speaks in terms of keeping search algorithms out of polynomial time, its proof infrastructure is enough to strongly address the issue of how often failure occurs—or at least to strongly link that to the open issue of how densely hard sets in $\mathrm{NP} \cap \mathrm{coNP}$ can be.

▶ **Theorem 9.** *If f is any nondecreasing function, and for some set $A \in \mathrm{NP} \cap \mathrm{coNP}$ it holds that every polynomial-time membership-in-A-testing algorithm errs, at infinitely many lengths n (respectively, at almost every length n), on at least $f(n)$ of the strings up to that length, then for each manipulative action (that appears in our $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ theorems) there will exist an $\epsilon > 0$ and an election system having a polynomial-time winner problem such that each search algorithm for that manipulative action with respect to that election system will err, at infinitely many lengths n (respectively, at almost every length n), on at least $f(n^{\epsilon})$ of the strings up to that length, but the decision problem will be in* P.

As a concrete example, if some set in $\mathrm{NP} \cap \mathrm{coNP}$ causes, for some $\epsilon > 0$, $2^{n^{\epsilon}}$ errors up to length n at infinitely many lengths, by each P algorithm, then in our theorems we can, for some $\tilde{\epsilon} > 0$, have our search problems for infinitely many lengths make each polynomial-time solver err $2^{n^{\tilde{\epsilon}}}$ times up to that length.

There is far too large a literature exploring the many aspects of search versus decision to cite it all here, but as an indication of how broad the literature is we mention a paper related to search versus decision as it interacts with parallelism [22] and a paper related to P-selectivity and self-reducibility [21]. Of course, the present paper is looking at concrete cases of search versus decision, in the context of manipulative actions on elections.

Does $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ hold? There are a number of problems that are known to belong to $\mathrm{NP} \cap \mathrm{coNP}$ yet that despite intense effort have not been shown to belong to P. Such problems include important questions about lattice problems, stochastic games, parity games, and factoring. Regarding factoring, it is well known that if $\mathrm{P} = \mathrm{NP} \cap \mathrm{coNP}$ then integer factoring is in polynomial time; this is to many people very strong evidence that $\mathrm{P} \neq \mathrm{NP} \cap \mathrm{coNP}$ (see [23]). *Note that, thus, if one believes factoring is hard, then by our results one must also believe that search and decision differ in complexity for many types of manipulative attack. The natural lesson to draw is that in framing definitions and questions, heightened attention should in the future be given to search versions.*

The problems mentioned in the previous paragraph are relevant to the most pressing open direction: Can one find existing—or build new but still highly *natural*—election systems for which, for some of the attacks we've discussed, the decision problem is in P but the search problem seems not to be in P? Note that since decision reduces to search, system-attack pairs known to have poly-time search algorithms or NP-hard decision problems are not reasonable possibilities here. Rather, the most attractive approach would be to find or more likely build natural election systems whose definitions involve seemingly NP-intermediate problems, such as factoring, lattice problems, and graph isomorphism.

Acknowledgments We are grateful to the STACS 2013 referees for helpful comments.

References

1 J. Bartholdi, III and J. Orlin. Single transferable vote resists strategic voting. *Social Choice and Welfare*, 8(4):341–354, 1991.

2 J. Bartholdi, III, C. Tovey, and M. Trick. The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6(3):227–241, 1989.

3 J. Bartholdi, III, C. Tovey, and M. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(2):157–165, 1989.

4 J. Bartholdi, III, C. Tovey, and M. Trick. How hard is it to control an election? *Mathematical and Computer Modeling*, 16(8/9):27–40, 1992.

5 D. Baumeister, G. Erdélyi, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Computational aspects of approval voting. In *Handbook of Approval Voting*. Springer, 2010.

6 A. Borodin and A. Demers. Some comments on functional self-reducibility and the NP hierarchy. Technical Report TR 76-284, Dept. of Comp. Sci., Cornell Univ., July 1976.

7 Y. Chevaleyre, U. Endriss, J. Lang, and N. Maudet. A short introduction to computational social choice. In *Proc. of SOFSOM-07*, pages 51–69. Springer-Verlag *LNCS #4362*.

8 V. Conitzer, T. Sandholm, and J. Lang. When are elections with few candidates hard to manipulate? *JACM*, 54(3):1–33, 2007.

9 G. Erdélyi, L. Piras, and J. Rothe. The complexity of voter partition in Bucklin and fallback voting: Solving three open problems. In *Proc. of AAMAS-11*, pages 837–844.

10 G. Erdélyi and J. Rothe. Control complexity in fallback voting. In *CATS-10*, pages 39–48.

11 P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. How hard is bribery in elections? *JAIR*, 35:485–532, 2009.

12 P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. Using complexity to protect elections. *Communications of the ACM*, 53(11):74–82, 2010.

13 P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. Multimode control attacks on elections. *JAIR*, 40:305–351, 2011.

14 P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Llull and Copeland voting computationally resist bribery and constructive control. *JAIR*, 35:275–341, 2009.

15 P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. A richer understanding of the complexity of election systems. In S. Ravi and S. Shukla, editors, *Fundamental Problems in Computing*, pages 375–406. Springer, 2009.

16 P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. The shield that never was: Societies with single-peaked preferences are more open to manipulation and control. *Inf. and Comp.*, 209(2):89–107, 2011.

17 E. Friedgut, G. Kalai, and N. Nisan. Elections can be manipulated often. In *Proc. of FOCS-08*, pages 243–249.

18 L. Hemachandra. *Counting in Structural Complexity Theory*. PhD thesis, Cornell University, Ithaca, NY, 1987. Available as Cornell CSD Technical Report TR87-840.

19 E. Hemaspaandra, L. Hemaspaandra, and C. Menton. Search versus decision for election manipulation problems. Technical Report arXiv:1202.6641 [cs.GT], arXiv.org, February 2012. Revised, March 2012.

20 E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Anyone but him: The complexity of precluding an alternative. *Artificial Intelligence*, 171(5–6):255–285, 2007.

21 E. Hemaspaandra, A. Naik, M. Ogihara, and A. Selman. P-selective sets and reducing search to decision vs. self-reducibility. *JCSS*, 53(2):194–209, 1996.

22 R. Karp, E. Upfal, and A. Wigderson. The complexity of parallel search. *JCSS*, 36(1):225–253, 1988.

23 S. Kintali. NP int. coNP. `kintali.wordpress.com/2010/06/06/np-intersect-conp/`.

24 C. Menton. Normalized range voting broadly resists control. *ToCS*. To appear.

25 J. Rothe. Complexity of certificates, heuristics, and counting types, with applications to cryptography and circuit theory. Habilitation, Friedrich-Schiller-Universität Jena, 1999.

26 J. Rothe, D. Baumeister, C. Lindner, and I. Rothe. *Einführung in Computational Social Choice*. Spektrum Akademischer Verlag, 2011.

Improved Bounds for Online Preemptive Matching

Leah Epstein[1], Asaf Levin[2], Danny Segev[3], and Oren Weimann[4]

1 **Department of Mathematics, University of Haifa**
Haifa 31905, Israel. Email: lea@math.haifa.ac.il
2 **Faculty of Industrial Engineering and Management, The Technion**
Haifa 32000, Israel. Email: levinas@ie.technion.ac.il
3 **Department of Statistics, University of Haifa**
Haifa 31905, Israel. Email: segevd@stat.haifa.ac.il
4 **Department of Computer Science, University of Haifa**
Haifa 31905, Israel. Email: oren@cs.haifa.ac.il

Abstract

When designing a preemptive online algorithm for the *maximum matching* problem, we wish to maintain a valid matching M while edges of the underlying graph are presented one after the other. When presented with an edge e, the algorithm should decide whether to augment the matching M by adding e (in which case e may be removed later on) or to keep M in its current form without adding e (in which case e is lost for good). The objective is to eventually hold a matching M with maximum weight.

The main contribution of this paper is to establish new lower and upper bounds on the competitive ratio achievable by preemptive online algorithms:

- We provide a lower bound of $1 + \ln 2 \approx 1.693$ on the competitive ratio of any randomized algorithm for the maximum cardinality matching problem, thus improving on the currently best known bound of $e/(e-1) \approx 1.581$ due to Karp, Vazirani, and Vazirani [STOC'90].
- We devise a randomized algorithm that achieves an expected competitive ratio of 5.356 for maximum weight matching. This finding demonstrates the power of randomization in this context, showing how to beat the tight bound of $3+2\sqrt{2} \approx 5.828$ for deterministic algorithms, obtained by combining the 5.828 upper bound of McGregor [APPROX'05] and the recent 5.828 lower bound of Varadaraja [ICALP'11].

1998 ACM Subject Classification F.2.2. Nonnumerical algorithms and problems

Keywords and phrases Online algorithms, matching, lower bound

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.389

1 Introduction

In the *maximum matching* problem, we are given an undirected graph $G = (V, E)$ whose edges have non-negative weights associated with them. A set of edges $M \subseteq E$ is called a *matching* when no two of them share a common vertex. The objective is to compute a matching of maximum total weight. Due to its wide real-life applicability, as well as to its appealing theoretical nature, this computational setting has received a great deal of attention from various communities such as computer science, mathematics, operations research, and economics (see Schrijver's book [13] and references therein for a comprehensive overview of classic work).

As can only be expected, the algorithmic research revolving around maximum matching has deviated from studying the traditional (offline) setting to other models. In particular, the online setting has been extensively studied over the last few decades [4, 6, 7, 8, 9, 10, 11].

Here, the edges (along with their weights) are presented one by one to the algorithm, which is required to keep a valid matching at all times. In other words, once an edge e is presented, the algorithm must decide whether to add it to M or not. However, e may be added only if the resulting set of edges $M \cup \{e\}$ remains a valid matching. The online setting has two fundamental models, depending on whether the acceptance of an edge is permanent or not.

In the non-preemptive model, the decision of whether or not to add any given edge to M is irrevocable, i.e., once an edge is added to the set M of previously accepted edges it can never be removed. The final matching thus consists of all edges that were ever accepted. Alas, in this model, simple examples demonstrate that the competitive ratio of any (deterministic or randomized) algorithm exceeds any function of the number of vertices, meaning that no competitive algorithm exists (see for example [6]). That being said, in the unweighted case (where all edge weights are equal, which is also called the *maximum cardinality matching* problem), a greedy approach that accepts an edge whenever possible has a competitive ratio of 2. For deterministic algorithms, this ratio is actually best possible, as shown by Karp, Vazirani, and Vazirani [9].

In the preemptive model, the algorithm is given more freedom by being able to regret on (retrospectively) bad decisions, in the sense that we are now allowed to remove previously accepted edges from the current matching at any point in time; this event is called *preemption*. Nevertheless, an edge that was either rejected or preempted cannot be re-inserted to the matching later on. As opposed to the non-preemptive model, with this extra freedom competitive algorithms do exist. Specifically, a deterministic algorithm that was proposed by Feigenbaum et al. [7] attains a competitive ratio of 6. Later on, McGregor [11] improved on this finding, by tweaking it into achieving a ratio of $3 + 2\sqrt{2} \approx 5.828$. On the other hand, Epstein et al. [6] established a lower bound of 4.967 for any deterministic algorithm, which has recently been improved by Varadaraja [3] to $3 + 2\sqrt{2} \approx 5.828$.

The upper bound of McGregor and the lower bound of Varadaraja establish a tight bound of 5.828 on the competitive ratio of any *deterministic* algorithm in the preemptive model. For *randomized* algorithms, the currently best lower bound of $e/(e-1) \approx 1.581$ can be inferred from the work of Karp et al. [9] on a similar model. Their worst-case example, which will be discussed later, actually works for the unweighted case.

1.1 Our results

The main contribution of this paper is to establish new lower and upper bounds on the competitive ratio of *randomized* algorithms in the *preemptive model*. Our findings, along with some technical comments, can be briefly summarized as follows.

A lower bound for unweighted graphs. We provide a lower bound of $1 + \ln 2 \approx 1.693$ on the competitive ratio of any randomized algorithm for the maximum cardinality matching problem, thus improving on the currently best known bound of $e/(e-1) \approx 1.581$ due to Karp et al. [9] in a similar model. It is worth pointing out that the latter has originally been proven to be best possible for bipartite graphs[1], and quite surprisingly, it turns out that our construction results in a bipartite graph as well. At first glance, this seems to be a contradiction. However, in the model studied by Karp et al., the edges adjacent to each vertex of one (fixed) part of the partition are all revealed simultaneously. On the other hand, in our construction, the sequence of arriving edges no longer follows this restriction. This bound is given in Section 2.

[1] Specifically, the authors also proposed the well-known ranking algorithm, whose competitive ratio exactly matches the lower bound of $e/(e-1)$.

An upper bound for weighted graphs. As previously mentioned, when arriving edges are accompanied by non-negative weights, there is a tight bound of $3 + 2\sqrt{2} \approx 5.828$ on the competitive ratio of any deterministic algorithm [11, 3]. An interesting open question is whether randomization offers any advantage in this context. We answer this question in the affirmative, by devising a *randomized* preemptive algorithm that beats the aforementioned bound, and achieves a competitive ratio of $\theta \approx 5.356$, where θ is the unique solution to $2(\ln\theta + 1) = \theta$ over $(2, \infty)$. This bound is given in Section 3.

1.2 Related work

What seems to have ignited renewed interest in the preemptive online model is the investigation of maximum matching in the *semi-streaming* model, which was introduced by Muthukrishnan [12]. In this model, the algorithm is allowed to use only $O(n \cdot \text{polylog}(n))$ space at all times but is not required to hold a valid matching. The possibility to keep in memory *any* set of edges (not only a matching) is what gives this model its added strength. In particular, an edge may be re-inserted to M, even if it has previously been removed, as long as this edge was kept in memory.

Epstein et al. [6] observed that the semi-streaming algorithms of Feigenbaum et al. [7] and McGregor [11] can actually be viewed as preemptive online algorithms in disguise. Nevertheless, the semi-streaming model is not as strict as the preemptive online model. In particular, the currently best semi-streaming algorithm for maximum weighted matching is that of Epstein et al. whose competitive ratio is 4.91. This improved on a ratio of 5.585 due to Zelke [16]. Both of these algorithms are *not* preemptive online, as the former may simultaneously hold $\Omega(\log n)$ matchings in memory (arguing that their union contains a good matching), while the latter keeps several additional edges for each edge in the current matching. When the semi-streaming algorithm is allowed to make a constant number of passes over the input stream, several algorithms of smaller approximation ratios were designed by McGregor [11] and Ahn and Guha [1, 2].

2 A Lower Bound for Randomized Algorithms

In this section, we establish a lower bound of $1 + \ln 2 \approx 1.693$ on the competitive ratio of any randomized algorithm in the preemptive online setting.

2.1 The general idea

Prior to delving into technical details, we provide a high-level description of how our construction works, along with some intuitive explanations. For ease of presentation, we use Yao's principle for profit maximization problems [14, 5]. That is, to prove a lower bound of C on the achievable (randomized) competitive ratio, we define a probability distribution on a class of inputs such that any deterministic algorithm (evaluated on this probability distribution) must have a competitive ratio of at least C.

Consider a fixed deterministic online algorithm ALG. In what follows, the underlying graph will be comprised of L layers (or vertical columns), each consisting of $2n$ vertices (see Figure 1). It is instructive to think of L and n as very large integers. With this structure in place, the input sequence starts with a random set of edges connecting vertices in layer 1 to vertices in layer 2. As soon as this sequence terminates, a new one begins, with a random set of edges between layers 2 and 3, then between layers 3 and 4, so on and so forth. The edges between adjacent layers ℓ and $\ell + 1$ are revealed in $2n$ rounds: In each round, all the edges connecting a vertex u in layer ℓ to its neighbors in layer $\ell + 1$ are revealed one after the other. The following properties are satisfied:

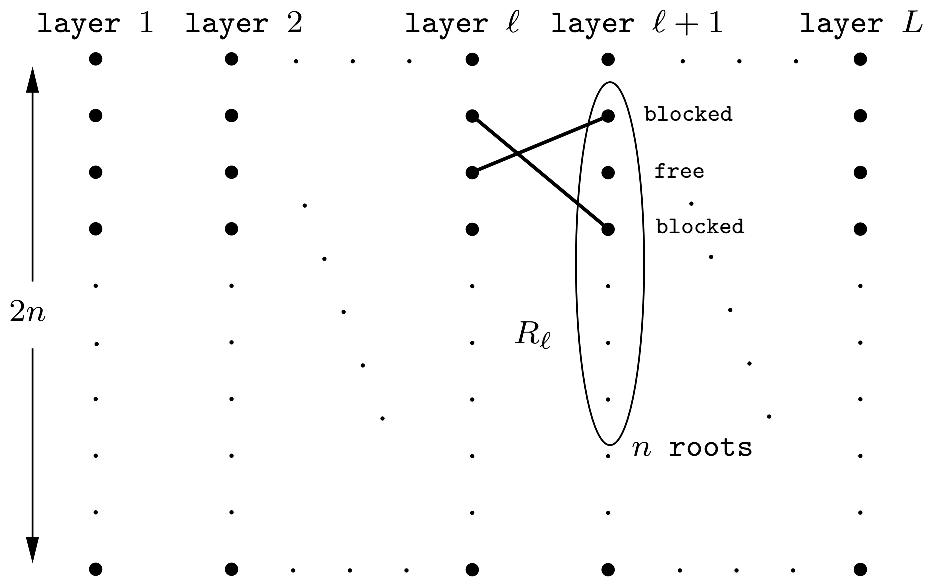


Figure 1 The layered graph used for the lower bound

Property 1: No preemption from previous layers. Once ALG picks a subset of edges between layers ℓ and $\ell + 1$, there is no motivation to preempt any of them when we proceed to subsequent layers. This property is already achieved by the above description, regardless of any particular randomization method we suggest, simply due to the fact that edges are revealed layer by layer. To see this, note that if ALG preempts such edge, say (u, v), then either the vertex v is eventually left unmatched, meaning that we have just lost (u, v) and gained nothing. Or, the vertex v is matched to a vertex in layer $\ell + 2$, meaning that the number of edges in the current matching remains unchanged, and we have just made things worse further down the road by (tentatively) matching a vertex in layer $\ell + 2$.

Property 2: No preemption from previous rounds. Recall that the edges between adjacent layers ℓ and $\ell + 1$ are revealed in $2n$ rounds. Consider some particular round, where all the edges connecting a vertex u in layer ℓ to its neighbors in layer $\ell + 1$ are revealed one after the other. Once this round terminates, ALG can hold a single edge (u, v) for some v in layer $\ell + 1$. From this point on, there is no motivation to ever preempt (u, v). To verify this, note that if ALG preempts (u, v), it has two options:

- Leave v unmatched to other vertices in layer ℓ, meaning that we have just lost (u, v) and gained nothing, since the only possible reason to preempt this edge is to match between v and vertices in layer $\ell + 2$, but there is no motivation to do so (by Property 1).
- Match v to some other vertex in layer ℓ, which leaves us at exactly the same situation when proceeding to the next layer (i.e., v is still matched).

From Properties 1 and 2 it may seem that preemptions do not occur at all. However, this is certainly not the case, as preemptions can occur within a round: While the edges connecting a vertex u in layer ℓ to all its neighbors in layer $\ell + 1$ are revealed, ALG may preempt a previously chosen edge (u, v) in favor of a different one (u, v').

The bottom line. The preceding discussion will allow us to argue that ALG picks edges in a very particular way. In fact, we are able to explicitly write the expected cardinality of the computed matching up to lower order terms, which evaluates to $\frac{1}{1+\ln 2} \cdot (L-1)n + o(Ln)$. On the other hand, in any realization of the randomized construction, we will show that there exists a feasible matching of cardinality $(L-1)n$. These observations will lead to the next theorem.

▶ **Theorem 2.1.** *The competitive ratio of any randomized preemptive online algorithm for maximum cardinality matching is at least* $1 + \ln 2$.

2.2 The randomized construction

In each layer $1 \leq \ell \leq L-1$, we will designate n out of its $2n$ vertices as being *roots*, collectively forming the set R_ℓ. These roots are a-priori unknown, and will be determined as soon as the random sequence of edges between layers $\ell - 1$ and ℓ terminates. This holds true for any layer other than the first, in which R_1 is defined by picking n arbitrary vertices. Once the set of roots R_ℓ is determined, we linearly order R_ℓ by picking a random permutation of these roots, such that each of the $n!$ possible permutation is equally likely. For any $r \in R_\ell$, the (random) location of r in this permutation is denoted by $L(r)$.

Revealing the edges. With these definitions in place, we can now explain how the edges between layers ℓ and $\ell+1$ are revealed. We say that a root in R_ℓ is *free* when it has not been matched to some vertex in the preceding layer, $\ell - 1$; otherwise, this root is said to be *blocked*.

Based on the order determined earlier, we pick the first root r and introduce edges connecting it to all $2n$ vertices in layer $\ell + 1$, which are initially colored white. If r is blocked, none of these edges is picked by ALG (Property 1). Otherwise, r is free and one of the newly presented edges will be picked, and will not be preempted in the future (Properties 1 and 2). We now choose a (random) vertex in layer $\ell + 1$, which is picked uniformly at random, and color it black. The second root in the linear ordering is then connected to all $2n - 1$ white vertices in layer $\ell + 1$, the same logic as to when this root is matched also applies here, and out of these $2n - 1$ white vertices one is randomly chosen to be colored black. This process continues for n iterations, until all roots in R_ℓ have been considered[2]. We still have to define a set of roots $R_{\ell+1}$ for the next layer, and these will be the n white vertices remaining in layer $\ell + 1$.

2.3 Analysis

Having explained how our construction works, we proceed by noting that the specific random process according to which edges are revealed can be used to derive two basic observations, comparing what can be achieved with and without knowing the input sequence in advance. These observation are summarized in the next lemma.

▶ **Lemma 2.2.** *For* $1 \leq \ell \leq L-1$, *let* F_ℓ *be a random variable that stands for the number of free roots in* R_ℓ. *Then,*

1. *ALG computes a matching of expected cardinality* $\sum_{\ell=1}^{L-1} \mathrm{E}[F_\ell]$.
2. *The maximum cardinality of a matching is* $(L-1)n$.

[2] In particular, a free root must be connected to at least two unmatched vertices in layer $\ell + 1$. This follows by observing that in iteration i, the current root is connected to $2n - (i-1)$ vertices, out of which at most $i - 1$ have previously been matched, so the number of unmatched vertices is at least $2n - 2(i-1) \geq 2$.

Proof. During the random process in which edges between layers ℓ and $\ell+1$ are revealed, we have argued in Section 2.2 that ALG expands the matching it holds just before this process begins by adding a new edge for every free root. In other words, if F_ℓ out of n roots in R_ℓ are free, ALG will add exactly F_ℓ edges to the matching, and these will not be preempted later on (Property 1). Therefore, evaluating the expected cardinality of the computed matching translates into evaluating the expectation of $\sum_{\ell=1}^{L-1} F_\ell$. On the other hand, the maximum cardinality of a matching in the resulting instance is $(L-1)n$, obtained by picking, for every layer ℓ, a matching of size n between the roots R_ℓ and the black vertices in layer $\ell+1$. A matching of this nature necessarily exists since the first root can be matched to the first vertex being colored black, the second root to the second vertex being colored black, and so on. ◀

Recursively computing E[F_ℓ]. In what follows, we derive a recursive formula that ties between the expected number of free roots in successive layers, showing that $\mathrm{E}[F_{\ell+1}]$ is close to being a linear function of $\mathrm{E}[F_\ell]$. For this purpose, we look into the question of how free roots (or equivalently, blocked roots) are created, and argue that the random permutation by which roots are processed leads to a large fraction of the next-layer roots being blocked (in expectation).

▶ **Lemma 2.3.** *There exists a layer-independent constant $c > 0$, such that for every $1 \le \ell \le L-1$,*

$$|\mathrm{E}[F_{\ell+1}] - n + \ln 2 \cdot \mathrm{E}[F_\ell]| \le c \ .$$

Proof. Let $B_{\ell+1}$ be the number of blocked roots in $R_{\ell+1}$. Since $\mathrm{E}[F_{\ell+1}] = n - \mathrm{E}[B_{\ell+1}]$, and $\mathrm{E}[B_{\ell+1}] = \mathrm{E}[\mathrm{E}[B_{\ell+1}|F_\ell]]$, it is sufficient to prove that $|\mathrm{E}[\mathrm{E}[B_{\ell+1}|F_\ell]] - \ln 2 \cdot \mathrm{E}[F_\ell]| \le c$, for a fixed value $c > 0$, independent of ℓ. To this end, we will show that $|\mathrm{E}[B_{\ell+1}|F_\ell = K] - \ln 2 \cdot K| \le c$, i.e., given that there are K free roots in layer ℓ, the expected number of blocked roots in layer $\ell+1$ is $\ln 2 \cdot K$, up to some additive constant.

Let $r_1, \dots, r_K$ be the collection of free roots in R_ℓ, indexed in some arbitrary order. The important observation is that blocked roots in layer $\ell+1$ are created only when they are matched to one of $r_1, \dots, r_K$, and once this happens, survive the random recoloring step in all subsequent iterations as white vertices. For example, if some free root r in layer ℓ is matched to v in layer $\ell+1$, and there are still p remaining iterations (excluding the current one), then v becomes a blocked root (at some time) with probability $\frac{n}{n+p+1}$, since out of the $n+p+1$ current white vertices, every subset of n vertices has equal probability to be the set of vertices that remain white.

Note that, for every $1 \le k \le K$, the location $L(r_k)$ in the random permutation is one of $1, \dots, n$, with equal probabilities to all values. Therefore, letting I_k be an indicator variable for the event where the vertex in layer $\ell+1$ to which r_k is matched survives all subsequent recoloring steps as a white vertex, we have

$$\mathrm{E}[I_k|F_\ell = K] = \Pr[I_k = 1|F_\ell = K] = \sum_{p=1}^{n} \frac{1}{n} \cdot \frac{n}{n+p} = H_{2n} - H_n = \ln 2 + \lambda_n \ ,$$

such that $|\lambda_n| = \Theta\left(\frac{1}{n}\right)$. The last equation holds since $|H_t - \ln t| = \gamma + O(\frac{1}{t})$, where γ is the Euler-Mascheroni constant [15]. The lemma follows by observing that

$$\mathrm{E}[B_{\ell+1}|F_\ell = K] = \mathrm{E}\left[\sum_{k=1}^{K} I_k \middle| F_\ell = k\right] = \sum_{k=1}^{K} \mathrm{E}[I_k|\, F_\ell = k] = \ln 2 \cdot K + K\lambda_n$$

and $|K\lambda_n| = \Theta(1)$. ◀

Concluding the proof of Theorem 2.1. Recall that by Lemma 2.3, we have $\mathrm{E}[F_{\ell+1}] + \ln 2 \cdot \mathrm{E}[F_\ell] \leq n + c$ for $1 \leq \ell \leq L-1$. Summing these inequalities over all values of ℓ gives

$$\sum_{\ell=1}^{L-1} \mathrm{E}\left[F_{\ell+1}\right] + \ln 2 \cdot \sum_{\ell=1}^{L-1} \mathrm{E}\left[F_\ell\right] \leq (L-1)n + (L-1)c \ .$$

Since $F_L \geq 0$ and $F_1 = n$, we have

$$\sum_{\ell=1}^{L-1} \mathrm{E}\left[F_{\ell+1}\right] + \ln 2 \cdot \sum_{\ell=1}^{L-1} \mathrm{E}\left[F_\ell\right] \geq (1+\ln 2) \cdot \sum_{\ell=1}^{L-1} \mathrm{E}\left[F_\ell\right] + F_L - F_1 \geq (1+\ln 2) \cdot \sum_{\ell=1}^{L-1} \mathrm{E}\left[F_\ell\right] - n \ .$$

Consequently, by Lemma 2.2 the expected cardinality of the matching computed by ALG is

$$\sum_{\ell=1}^{L-1} \mathrm{E}\left[F_\ell\right] \leq \frac{1}{1+\ln 2} \cdot (Ln + (L-1)c) \ .$$

On the other hand, the maximum cardinality of a matching in the resulting instance is $(L-1)n$, implying that the asymptotic competitive ratio of ALG (when L and n tend to infinity) is $1 + \ln 2$. Theorem 2.1 then follows.

3 A Randomized Algorithm

In this section, we show that by employing randomization, the lower bound of $3+2\sqrt{2} \approx 5.828$ on the performance of any deterministic algorithm can be beaten. In particular, by making use of randomized geometric rounding (see, for instance, [6]), our algorithm achieves an expected competitive ratio of roughly 5.356.

3.1 The algorithm

Parameters. In what follows, we utilize two real-valued parameters: A *base* $\theta > 1$, and a *shifting value* $\phi > 0$. The base θ will be optimized later on, so that the resulting competitive ratio of our algorithm is made as small as possible. In contrast, the shifting value ϕ will not be fixed; instead, it will be a random variable whose value is chosen to be θ^τ, where τ is uniformly distributed over the interval $(0, 1]$.

Weight classes and rounded weights. We define weight classes of edges in the following way. Let $w(e)$ denote the (non-negative) weight of an edge e. For every $i \in \mathbb{Z}$, we let the weight class W_i be the collection of edges whose weight is in the interval $[\phi\theta^i, \phi\theta^{i+1})$. Once an edge e is presented, we round down its original weight $w(e)$ to the lower endpoint of the weight class it belongs to, thereby obtaining its *rounded* weight $\tilde{w}(e)$. In other words, letting i be the unique integer for which $w(e) \in [\phi\theta^i, \phi\theta^{i+1})$, we set $\tilde{w}(e) = \phi\theta^i$. In the remainder of this section, the latter notation will be extended to matchings, so that $w(M)$ and $\tilde{w}(M)$ will stand for the original weight and rounded weight, respectively, of a matching M. In addition, for a maximum weight matching M^*, we denote $\mathrm{OPT} = w(M^*)$ and $\widetilde{\mathrm{OPT}} = \tilde{w}(M^*)$. Moreover, let $\widetilde{\mathrm{OPT}}_\tau$ denote the profit of a maximum weight matching for the weight function $\tilde{w}$ (resulting from a particular choice of τ).

Maintaining a matching online. The algorithm keeps a tentative matching M, which is initialized prior to reading the input sequence as $M = \emptyset$. Upon the arrival of a newly-presented edge $e = (u, v)$, we proceed as follows. Let $X(M, e)$ denote the set of edges in the matching M that have a common endpoint with e, that is, edges that have u or v as an endpoint. Clearly, there could be at most two such edges. If every edge $e' \in X(M, e)$ satisfies $\tilde{w}(e') < \tilde{w}(e)$ then e is inserted into M while the edges in $X(M, e)$ are preempted, i.e., we set $M \leftarrow (M \setminus X(M, e)) \cup \{e\}$. Otherwise, M remains unchanged.

3.2 Analysis

We begin by accounting for the extent to which the weight of each edge is rounded. More specifically, the next lemma shows how to evaluate the ratio between the expected rounded weight of any edge[3] to its original weight in terms of the base θ. Subsequently, this allows us to bound the ratio between the OPT and $\widetilde{\mathrm{OPT}}$.

▶ **Lemma 3.1.** *For every edge e, we have $\mathrm{E}_\tau[\tilde{w}(e)/w(e)] = \frac{\theta-1}{\theta \ln \theta}$.*

Proof. We denote by $\tilde{w}^\tau(e)$ the value $\tilde{w}(e)$ for a given choice of τ. Let p be an integer and let $0 < \alpha \leq 1$ be such that $w(e) = \theta^{p+\alpha}$ is satisfied. By our definition of the weight classes $\{W_i\}_{i \in \mathbb{Z}}$, it follows that the rounded weight $\tilde{w}^\tau(e)$ is determined by:

$$\tilde{w}^\tau(e) = \begin{cases} \theta^{p+\tau} & \text{if } \tau \leq \alpha \\ \theta^{p+\tau-1} & \text{if } \tau > \alpha \end{cases}$$

Therefore, the ratio between the expected rounded weight of e to its original weight is

$$\mathrm{E}_\tau\left[\frac{\tilde{w}(e)}{w(e)}\right] = \int_0^\alpha \frac{\theta^{p+\tau}}{\theta^{p+\alpha}} d\tau + \int_\alpha^1 \frac{\theta^{p+\tau-1}}{\theta^{p+\alpha}} d\tau = \frac{1}{\ln \theta} \cdot \left(\frac{1}{\theta^\alpha}(\theta^\alpha - 1) + \frac{1}{\theta^{\alpha+1}}(\theta - \theta^\alpha) \right) = \frac{\theta - 1}{\theta \ln \theta} .$$

◀

▶ **Lemma 3.2.** *The expected rounded weight of the optimal matching M^* satisfies $\mathrm{E}_\tau[\widetilde{\mathrm{OPT}}] = \frac{\theta-1}{\theta \ln \theta}\mathrm{OPT}$. Also, for any realization of the random variable τ, we have $\mathrm{OPT} < \theta \cdot \widetilde{\mathrm{OPT}}_\tau$.*

Proof. It is easy to verify that, due to linearity of expectation, the first claim follows from separately applying Lemma 3.1 on every edge of the optimal matching M^*, and summing over all edges. On the other hand, the second claim holds since, regardless of the choice of τ, for any edge e (in particular, those in M^*) we have $\tilde{w}(e)/w(e) > 1/\theta$, as the lower and upper endpoints of each weight class differ by a factor of at most θ. ◀

Up until now we have merely discussed how the expected weight of the optimal matching M^* depends on the geometric rounding procedure. We now move on to describe the technical crux in our analysis, where the online matching computed by the algorithm is compared against the rounded weight of M^*.

▶ **Lemma 3.3.** *For any given value of τ, the profit of the algorithm is at least $\frac{\theta-2}{2\theta-2}\widetilde{\mathrm{OPT}}$.*

Proof. We consider how the algorithm operates for an arbitrary choice of the variable τ. For this purpose, consider an optimal solution $\tilde{M}$ for the "rounded" instance, i.e., a maximum weight matching with respect to the rounded weights $\tilde{w}(\cdot)$. Clearly, $\tilde{w}(\tilde{M}) \geq \tilde{w}(M^*) = \widetilde{\mathrm{OPT}}$, as M^* is an optimal matching for the original weights, but not necessarily for the rounded weights.

We associate every edge of $\tilde{M}$ with an edge in the matching M that is being held by the algorithm upon termination, possibly using a single edge in M as a target for several edges

[3] Note that the expectation $\mathrm{E}_\tau[\cdot]$ is taken over the random choice of the variable τ.

in $\tilde{M}$. The way this association will be defined later on enables us to argue that the ratio between the total rounded weight of edges associated with any edge $e \in M$ and between $\tilde{w}(e)$ is at most $\frac{2\theta-2}{\theta-2}$. This claim immediately leads to a ratio of $\frac{2\theta-2}{\theta-2}$ between $\widetilde{\mathrm{OPT}}$ and the total profit of the algorithm according to the rounded weights $\tilde{w}(\cdot)$. Since $w(e) \geq \tilde{w}(e)$ for every edge e, the actual profit of the algorithm can only be higher.

For every edge $e \in M$, we create a *preemption tree*. The root of this tree is e, and the children of an edge e' are all edges that were preempted from the matching kept by the algorithm when e' was inserted. Note that the number of children is either 2, 1, or 0, since any matching cannot contain more than a single edge for every endpoint of e' (prior to the arrival of e'). The set of edges that do not belong to any preemption tree are edges that the algorithm never accepted, while the union of all edge sets over all trees is exactly the set of edges that belonged to the matching at some point in time.

By definition of the algorithm, for every edge $\tilde{e}$ that does not belong to a tree, there exists an edge e' that does belong to a tree, sharing an endpoint with $\tilde{e}$, such that $\tilde{w}(e') \geq \tilde{w}(\tilde{e})$. If for an edge $\tilde{e} \in \tilde{M}$ such that $\tilde{e}$ does not belong to a tree, there exists a unique such edge e', then $\tilde{e}$ is associated with the root of the tree where e' appears. Otherwise, one such edge e' is chosen arbitrarily, and $\tilde{e}$ is associated with the root of the tree of e' in this case as well. For an edge $\tilde{e} \in \tilde{M}$ that belongs to some tree, $\tilde{w}$ is associated with the root of the tree in which it appears.

Next, consider a specific tree with the root $e \in M$. For an edge $\hat{e}$ of distance $d[\hat{e}]$ from the root, measured in number of edges, let $\tilde{w}_d(\hat{e}) = \tilde{w}(e)/\theta^{d[\hat{e}]}$.

▶ Claim 3.4. For every edge $\hat{e}$ in the tree, $\tilde{w}(\hat{e}) \leq \tilde{w}_d(\hat{e})$.

Proof. Consider the path $e_0, e_1, e_2, \ldots, e_{d[\hat{e}]}$, where $e_0 = \hat{e}$, $e_{d[\hat{e}]} = e$, and for every $1 \leq i \leq d[\hat{e}]$, e_i is the edge whose arrival caused e_{i-1} to be preempted. By definition of the algorithm, $\tilde{w}(e_i) > \tilde{w}(e_{i-1})$. Since our geometric rounding method guarantees that, when rounded weights are not identical, they differ by a multiplicative factor of at least θ, it follows that $\tilde{w}(e_i) \geq \theta\tilde{w}(e_{i-1})$. Therefore, $\tilde{w}(e) = \tilde{w}(e_{d[\hat{e}]}) \geq \theta^{d[\hat{e}]}\tilde{w}(e_0) = \theta^{d[\hat{e}]}\tilde{w}(\hat{e})$, or alternatively $\tilde{w}(\hat{e}) \leq \tilde{w}_d(\hat{e})$. ◀

For a vertex v in the graph, and for every possible distance $d \geq 0$, we say that v is *new* for d if: (1) there is an edge of depth d in some preemption tree for which v is an endpoint; and (2) v is not an endpoint of any edge of smaller depth. Also, The two endpoints of an edge $e \in M$ are new for $d = 0$ and are not new for any other value of d.

▶ Claim 3.5. For every $d > 0$, there are at most 2^d new vertices d. Moreover, every edge in a tree in level $d > 0$ has at most one new vertex.

Proof. Since every edge has at most two children, the number of edges of depth d is at most 2^d. Every such edge has a common endpoint with an edge of depth $d-1$, thus there is at most one new vertex per edge, which results in a total of at most 2^d new vertices. ◀

▶ Claim 3.6. For every edge $e \in M$, the total rounded weight of the edges associated with e is at most $\frac{2\theta-2}{\theta-2}\tilde{w}(e)$.

Proof. For an edge $\tilde{e} \in \tilde{M}$, let $d_{\min}[\tilde{e}]$ be the minimum depth such that an edge of this depth has a common vertex with $\tilde{e}$. By definition, this common vertex must be new for level $d_{\min}[\tilde{e}]$. Let $\bar{e}$ be the edge of level $d_{\min}[\tilde{e}]$ with the common vertex. Since there exists an edge in the tree having a common endpoint with $\tilde{e}$ of rounded weight at least $\tilde{w}(\tilde{e})$, we have $\tilde{w}(\bar{e}) \geq \tilde{w}(\tilde{e})$. In addition, since $\tilde{M}$ is a matching, for every edge in the tree and every new vertex v that edge, at most one edge of $\tilde{M}$ has v as an endpoint. By Claim 3.5, every edge of the tree other than e has at most one new vertex, so the total rounded weight is at most the total rounded weight of all edges in the tree plus $\tilde{w}(e)$. Letting $\mathcal{T}(e)$ denote the set of edges in the tree rooted at e, by Claim 3.4 it follows that the total rounded weight is at most

$$\sum_{\bar{e}\in\mathcal{T}(e)} \tilde{w}(\bar{e}) + \tilde{w}(e) \leq \sum_{\bar{e}\in\mathcal{T}(e)} \tilde{w}_d(\bar{e}) + \tilde{w}(e) \leq \sum_{d=0}^{\infty} 2^d \frac{\tilde{w}(e)}{\theta^d} + \tilde{w}(e) \leq \left(\frac{1}{1-2/\theta} + 1\right) \tilde{w}(e) \ .$$

◀

This completes the proof of Lemma 3.3. ◀

▶ **Theorem 3.7.** *The algorithm achieves an expected competitive ratio of $\frac{2\theta \ln \theta}{\theta - 2}$. This ratio is minimized for $\theta^* \approx 5.356$, where θ^* is the unique solution to $2(\ln\theta + 1) = \theta$ over $(2, \infty)$, in which case its value is θ^*.*

Proof. By Lemmas 3.2 and 3.3, we have $\mathrm{E}_\tau[w(M)] \geq \frac{\theta-2}{2\theta-2}\mathrm{E}_\tau[\widetilde{\mathrm{OPT}}] \geq \frac{\theta-2}{2\theta-2} \cdot \frac{\theta-1}{\theta\ln\theta}\mathrm{OPT} = \frac{\theta-2}{2\theta\ln\theta}\mathrm{OPT}$. ◀

References

1 K. J. Ahn and S. Guha. Laminar families and metric embeddings: Non-bipartite maximum matching problem in the semi-streaming model. Available online at: `http://arxiv.org/abs/1104.4058`.

2 K. J. Ahn and S. Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 526–538, 2011.

3 A. Badanidiyuru Varadaraja. Buyback problem – approximate matroid intersection with cancellation costs. In *Proceedings of the 38th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 379–390, 2011.

4 N. Bansal, N. Buchbinder, A. Gupta, and J. Naor. An $O(\log^2 k)$-competitive algorithm for metric bipartite matching. In *Proceedings of the 15th annual European Symposium on Algorithms (ESA)*, pages 522–533, 2007.

5 A. Borodin and R. El-Yaniv. On randomization in on-line computation. *Information and Computation*, 150(2):244–267, 1999.

6 L. Epstein, A. Levin, J. Mestre, and D. Segev. Improved approximation guarantees for weighted matching in the semi-streaming model. *SIAM Journal on Discrete Mathematics*, 25(3):1251–1265, 2011.

7 J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3):207–216, 2005.

8 B. Kalyanasundaram and K. Pruhs. Online weighted matching. *Journal of Algorithms*, 14(3):478–488, 1993.

9 R. Karp, U. Vazirani, and V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 352–358, 1990.

10 S. Khuller, S. Mitchell, and V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theoretical Computer Science*, 127(2):255–267, 1994.

11 A. McGregor. Finding graph matchings in data streams. In *Proceedings of the 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 170–181, 2005.

12 S. Muthukrishnan. *Data Streams: Algorithms and Applications.* Foundations and Trends in Theoretical Computer Science. Now Publishers Inc, 2005.

13 A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, 2003.

14 A. C.-C. Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 222–227, 1977.

15 R. M. Young. Euler's constant. *The Mathematical gazette*, 75:187–190, 1991.

16 M. Zelke. Weighted matching in the semi-streaming model. *Algorithmica*, 62(1-2):1–20, 2012.

Parameterized Matching in the Streaming Model

Markus Jalsenius[1], Benny Porat[2], and Benjamin Sach[3]

1 Department of Computer Science, University of Bristol, U.K.
2 Department of Computer Science, Bar-Ilan University, Israel.
3 Department of Computer Science, University of Warwick, U.K.

Abstract

We study the problem of parameterized matching in a stream where we want to output matches between a pattern of length m and the last m symbols of the stream before the next symbol arrives. Parameterized matching is a natural generalisation of exact matching where an arbitrary one-to-one relabelling of pattern symbols is allowed. We show how this problem can be solved in constant time per arriving stream symbol and sublinear, near optimal space with high probability. Our results are surprising and important: it has been shown that almost no streaming pattern matching problems can be solved (not even randomised) in less than $\Theta(m)$ space, with exact matching as the only known problem to have a sublinear, near optimal space solution. Here we demonstrate that a similar sublinear, near optimal space solution is achievable for an even more challenging problem.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Pattern matching, streaming algorithms, randomized algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.400

1 Introduction

We consider the problem of pattern matching in a stream where we want to output matches between a pattern of length m and the last m symbols of the stream. Each answer must be reported before the next symbol arrives. The problem we consider in this paper is known as *parameterized matching* and is a natural generalisation of exact matching where an arbitrary one-to-one relabelling of the pattern symbols is allowed (one per alignment). For example, if the pattern is `abbca` then there there is a parameterized match with `bddcb` as we can apply the relabelling `a`$\rightarrow$`b`, `b`$\rightarrow$`d`, `c`$\rightarrow$`c`. There is however no parameterized match with `bddbb`. We show how this streaming pattern matching problem can be solved in near constant time per arriving stream symbol and sublinear, near optimal, space with high probability. The space used is reduced even further when only a small subset of the symbols are allowed to be relabelled. As discussed in the next section, our results demonstrate a serious push forward in understanding what pattern matching algorithms can be solved in sublinear space.

1.1 Background

Streaming algorithms is a well studied area and specifically finding patterns in a stream is a fundamental problem that has received increasing attention over the past few years. It was shown in [8] that many offline algorithms can be made online (streaming) and deamortised with a $\log m$ factor overhead in the time complexity per arriving symbol in the stream, where m is the length of the pattern. There have also been improvements for specific pattern matching problems but they all have one property in common: space usage is $\Theta(m)$ words. It is not difficult to show that we in fact *need* as much as $\Theta(m)$ space to do pattern matching, unless errors are allowed. The field of pattern matching in a stream took a significant step forwards in 2009 when it was shown to be possible to solve exact matching using only $O(\log m)$ words of space and $O(\log m)$ time per new stream symbol [15]. This method, which is based on fingerprints, correctly finds all matches with high probability. The initial approach was subsequently somewhat simplified [10] and then finally improved to run in constant time [7] within the same space requirements.

Being able to do exact matching in sublinear space raised the question of what other streaming pattern matching problems can be solved in small space. In 2011 this question was answered for a large set of such problems [9]. The result was rather gloomy: almost no streaming pattern matching problems can be solved in sublinear space, not even using randomised algorithms. An $\Omega(m)$ space lower bound was given for L_1, L_2, L_∞, Hamming, edit distance and pattern matching with wildcards as well as for any algorithm that computes the cross-correlation/convolution. So what other pattern matching problems could possibly be solved in small space? It seems that the only hope to find any is by imposing various restrictions on the problem definition. This was indeed done in [15] where a solution to k-mismatch (exact matching where up to k mismatches are allowed) was given which uses $O(k^2\text{poly}(\log m))$ time per arriving stream symbol and $O(k^3\text{poly}(\log m))$ words of space. The solution involves multiple instances of the exact matching algorithm run in parallel. Note that the space bound approaches $\Theta(m)$ as k increases, so the algorithm is only interesting for sufficiently small k. Further, the space bound is very far from the known $\Omega(k)$ lower bound. We also note that it is straightforward to show that exact matching with k wildcards in the pattern can be solved with the k-mismatch algorithm. To our knowledge, no other streaming pattern matching have been solved in sublinear space so far.

In this paper we present the first push forward since exact matching by giving a sublinear, near optimal space and near constant time algorithm for parameterized matching in a stream. This natural problem turns out to be significantly more complicated to solve than exact matching and our results provide the first demonstration that small space and time bounds are achievable for a more challenging problem. Note that our space bound, as opposed to k-mismatch, is essentially optimal like for exact matching. One could easily argue that our results are surprising, and yet again the question of what other problems are solvable in sublinear space calls for an answer. In particular, given that restrictions to the problem have to be made, what restrictions should one make to break the $\Omega(m)$ space barrier.

1.2 Problem definition and related work

A pattern P of length m is said to *parameterized match*, or *p-match* for short, an m length string S if there is an injective (one-to-one) function f such that $S[j] = f(P[j])$ for all $j \in \{0, \dots, m-1\}$. In our streaming setting, the pattern is known in advance and the symbols of the stream T arrive one at a time. We use the letter i to denote the index of the latest symbol in the stream. Our task is to output whether there is a p-match between P and $T[(i-m+1),\ i]$ before $T[i+1]$ arrives. The mapping f may be distinct for each i.

One may view this matching problem as that of finding matches in a stream encrypted using a substitution cipher. In offline settings, p-matching has its origin in finding duplication and plagiarism in software code although has since found numerous other applications. Since the first introduction of the problem, a great deal of work has gone into its study in both theoretical and practical settings (see e.g. [3, 1, 4, 5, 6, 12]). Notably, in an offline setting, the exact p-matching problem can be solved in near linear time using a variant [1] of the classic linear time exact matching algorithm KMP [14].

When the sublinear space algorithm for exact matching was given in [15], properties of the periods of strings formed a crucial part of their analysis. However, when considering p-matching the period of a string is a much less straightforward concept than it is for exact matching. For example, it is no longer true that consecutive matches must either be separated by the period of the pattern or be at least $m/2$ symbols apart. This property, which holds for exact but not p-matching, allows for an efficient encoding of the positions of the matches. This was crucial to reducing the space requirements of the previous streaming algorithms. Unfortunately, p-matches can occur at arbitrary positions in the stream, requiring new insights. This is not the only challenge that we face.

A natural way to match two strings under parameterization is to consider their *predecessor strings*. For a string S, the predecessor string, denoted $\text{pred}(S)$, is a string of length $|S|$ such that $\text{pred}(S)[j]$ is the distance, counted in numbers of symbols, to the previous occurrence of the symbol $S[j]$ in S. In other words, $\text{pred}(S)[j] = d$, where d is the smallest positive value for which $S[j] = S[j-d]$. Whenever no such d exists, we set $\text{pred}(S)[j] = 0$. As an example, if $S =$ `aababcca` then $\text{pred}(S) =$ `01022014`. We can perform p-matching offline by only considering predecessor strings using the fundamental fact [3] that two equal length strings S and S' p-match iff $\text{pred}(S) = \text{pred}(S')$. A plausible approach for our streaming problem would now be to translate the problem of p-matching in a stream to that of exact matching. This could be achieved by converting both pattern and stream into their corresponding predecessor strings and maintaining fingerprints of a sliding window of the translated input. However, consider the effect on the predecessor string, and hence its fingerprint, of sliding a window in the stream along by one. The leftmost symbol x, say, will move out of the window and so the predecessor value of the new leftmost occurrence of x in the new window will need to be set to `0` and the corresponding fingerprint updated. We cannot afford to store the positions of all characters in a $\Theta(m)$ length window.

We will show a matching algorithm that solves these problems and others we encounter en route using minimal space and in near constant time per arriving symbol. A number of technical innovations are required, including new uses of fingerprinting, a new compressed encoding of the positions of potential matches, a separate deterministic algorithm designed for prefixes of the pattern with small parameterized period as well as the deamortisation of the entire matching process. Section 2 gives a more detailed overview of these main hurdles.

1.3 Our new results

Our main result is a fast and space efficient algorithm for the streaming p-matching problem. It applies to *dense* alphabets where both the pattern and streaming text alphabets are $\Sigma = \{0, \ldots, |\Sigma| - 1\}$. Theorem 1 below is proved over the subsequent sections of this paper. Some supporting proofs are left for the full paper due to space constraints, including the proof of Theorem 2, which is based on communication complexity arguments.

▶ **Theorem 1.** *Let the pattern have length m, the text have length n and both have alphabets $\Sigma = \{0, \ldots, |\Sigma| - 1\}$. There is a randomised algorithm for streaming p-matching that takes $O(1)$ worst-case time per character and uses $O(|\Sigma| \log m)$ words of space. The probability that the output is correct at all text alignments is at least $1 - 1/n^c$ for any constant c.*

▶ **Theorem 2.** *There is a randomised space lower bound of $\Omega(|\Sigma|)$ bits for the streaming p-matching problem, where Σ is the pattern alphabet.*

Parameterized matching is often specified under the assumption that only some symbols are variable (allowed to be relabelled). The mapping f we used in Section 1.2 has to reflect this constraint. More precisely, let the pattern alphabet be partitioned into fixed symbols

Σ_{fixed} and variable symbols Π. For $\sigma \in \Sigma_{\text{fixed}}$, we require that $f(\sigma) = \sigma$. The result from Theorem 1 can be extended to handle *general* alphabets with arbitrary fixed symbols. The idea is to apply a suitable reduction that was given in [1] (Lemma 2.2) together with the streaming exact matching algorithm of Breslauer and Galil [7], as well as applying a "filter" on the text stream using a dynamic dictionary (for instance that of [2]). The dictionary is used to map text symbols to the variable pattern symbols in Π. The proof is omitted.

▶ **Theorem 3.** *Suppose Π is the set of pattern symbols that can be relabelled under p-matching. All other pattern symbols are fixed. Without any constraints on the text alphabet, there is a randomised algorithm for streaming p-matching that takes $O(\sqrt{\log|\Pi| / \log\log|\Pi|})$ worst-case time per character and uses $O(|\Pi|\log m)$ words of space, where m is the length of the pattern. The probability that the algorithm outputs correctly at all alignments of an n length text is at least $1 - 1/n^c$, where c is any constant.*

As part of the proof of Theorem 1 we had to develop an algorithm that efficiently solves streaming p-matching for patterns with small *parameterized period*. The parameterized period (*p-period*) of the pattern P, denoted ρ, is the smallest positive integer such that $P[0,\ (m-1-\rho)]$ p-matches $P[\rho,\ m-1]$. That is, ρ is the shortest distance that P must be slid by to p-match itself. Our algorithm is deterministic and is interesting in its own right (see Section 4). We also provide an almost matching space lower bound (proof omitted).

▶ **Theorem 4.** *Suppose the pattern and text alphabets are both $\Sigma = \{0, \dots, |\Sigma| - 1\}$ and the pattern has p-period ρ. There is a deterministic algorithm for streaming p-matching that takes $O(1)$ worst-case time per character and uses $O(|\Sigma| + \rho)$ words of space. Further, there is a deterministic space lower bound of $\Omega(|\Sigma| + \rho)$ bits.*

1.4 Fingerprints

We will make extensive use of Rabin-Karp style fingerprints of strings which are defined as follows. Let S be a string over the alphabet Σ. Let $p > |\Sigma|$ be a prime and choose $r \in \mathbb{Z}_p$ uniformly at random. The fingerprint $\phi(S)$ is given by $\phi(S) \overset{\text{def}}{=} \sum_{k=0}^{|S|-1} S[k]r^k \bmod p$. A critical property of the fingerprint function ϕ is that the probability of achieving a false positive, $\Pr(\phi(S) = \phi(S') \wedge S \neq S')$, is at most $|S|/(p-1)$ (see [13, 15] for proofs). Let n denote the total length of the stream. Our randomised algorithm will make $o(n^2)$ (in fact near linear) fingerprint comparisons in total. Therefore, by the applying the union bound, for any constant c, we can choose p to be of size $\Theta(n^{c+3})$ so that with probability at least $1 - 1/n^c$ there will be no false positive matches.

As we assume the RAM model with word size $\Theta(\log n)$, a fingerprint fits in a constant number of words. We assume that all fingerprint arithmetic is performed within $\mathbb{Z}_p$. In particular we will take advantage of two fingerprint operations.

- $\ominus$ *Splitting:* Given $\phi\big(S[0,\ a]\big)$, $\phi\big(S[0,\ b]\big)$ (where $b > a$) and the value of $r^{-a} \bmod p$, we can compute $\phi\big(S[a+1,\ b]\big) = \phi\big(S[0,\ b]\big) \ominus \phi\big(S[0,\ a]\big)$ in $O(1)$ time.
- $\circledcirc$ *Zeroing:* Let S, S' be two equal length strings such that S' is identical to S except for in positions $z \in Z \subseteq [0, s-1]$ at which $S'[z] = 0$. We write $\phi\big(S\big) \circledcirc Z$ to denote $\phi\big(S'\big)$. Given $\phi\big(S\big)$ and $(S[z],\ r^z \bmod p)$ for all $z \in Z$, computing $\phi\big(S\big) \circledcirc Z$ takes $O(|Z|)$ time.

2 Overview, key properties and notation

The overall idea of our algorithm in Theorem 1 follows that of previous work on streaming exact matching in small space, however for p-matching the situation is much more complex

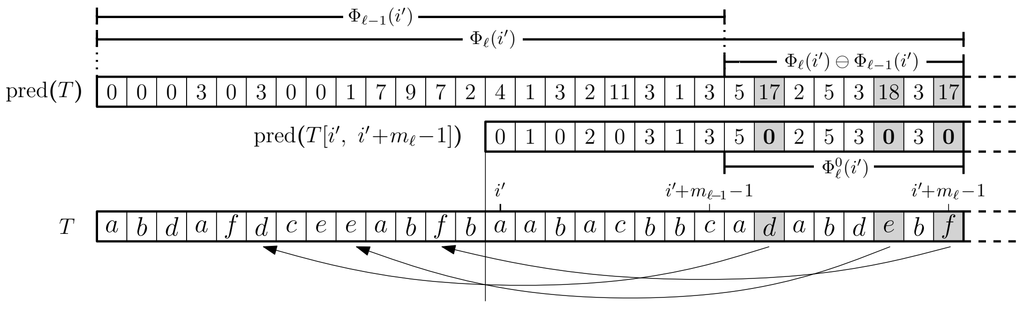


Figure 1 The key fingerprints used by the randomised algorithm. Characters contributing differently to $\Phi^0_\ell(i')$ and $\Phi_\ell(i') \ominus \Phi_{\ell-1}(i')$ are highlighted.

and calls for not only more involved details and methods but also a deep fundamental understanding of the nature of p-matching. We will now describe the overall idea, introduce some important notation and at the end of this section we will highlight key facts about p-matching that are crucial for our solution.

The main algorithm will try to match the streaming text with various prefixes of the pattern P. Let Σ_{P} denote the pattern alphabet. We define $\delta = |\Sigma_{\mathrm{P}}| \log m$ and let P_0 denote the shortest prefix of P that has p-period greater than 3δ (recall the definition of p-period given above Theorem 4). We define s prefixes P_ℓ of increasing length so that $|P_\ell| = 2^\ell |P_0|$ for $\ell \in \{1, \dots, s-1\}$, where $s \leqslant \lceil \log m \rceil$ is the largest value such that $|P_{s-1}| \leqslant m/2$. The final prefix P_s has length $m - 4\delta$. For all ℓ, we define $m_\ell = |P_\ell|$, hence $m_\ell = 2m_{\ell-1}$.

In order to determine if there is a p-match between the text and a pattern prefix, we will compare the fingerprints of their predecessor strings (recall that two strings p-match iff their predecessor strings are the same). We will need two related (but typically distinct) fingerprint definitions to achieve this. Figure 1 will be helpful when reading the following definitions which are discussed in an example below. For any index i' and $\ell \in \{0, \dots, s\}$,

$$\Phi_\ell(i') \overset{\text{def}}{=} \phi\big(\mathrm{pred}(T[0,\ (i'+m_\ell-1)])\big)\,,$$
$$\Phi^0_\ell(i') \overset{\text{def}}{=} \phi\big(\mathrm{pred}(T[i',\ (i'+m_\ell-1)])[m_{\ell-1},\ m_\ell-1]\big)\,.$$

For each $\ell \in \{1, \dots, s\}$ the main algorithm runs a process whose responsibility for finding p-matches between the text and P_ℓ (P_0 is handled separately as will be discussed later). The process responsible for P_ℓ will ask the process responsible for $P_{\ell-1}$ if it has found any p-matches, and if so it will try to extend the matches to P_ℓ. As an example, suppose that the process for $P_{\ell-1}$ finds a match at position i' of the text (refer to Figure 1). The process will then store this match along with the fingerprint $\Phi_{\ell-1}(i')$ which has been built up as new symbols arrive. The process for P_ℓ will be handed this information when the symbol at position $i' + m_\ell - 1$ arrives. The task is now to work out if i' is also a matching position with P_ℓ. With the fingerprint $\Phi_\ell(i')$ available (built up as new symbols arrive), the process for P_ℓ can use fingerprint arithmetics to determine if i' is a matching position. This is one instance where the situation becomes more tricky than one might first think.

As position i' is a p-match with $P_{\ell-1}$ it suffices to compare the second half of the predecessor string of P_ℓ with the second half of the predecessor string of $T[i',\ i'+m_\ell-1]$. Fingerprints are used for this comparison. It is crucial to understand that $\Phi_\ell(i') \ominus \Phi_{\ell-1}(i')$ cannot be used directly here; some predecessor values of the text might point very far back, namely to some position *before* index i'. In Figure 1 we have shaded the three symbols for which this is true and have drawn arrows indicating their predecessors. Thus, in order to

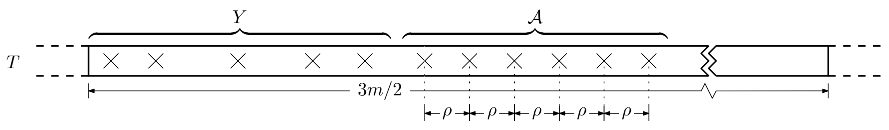


Figure 2 Partitioning of positions ($\times$) at which P p-matches in a $3m/2$ length substring of T.

correctly do the fingerprint comparison we need to set those positions to zero (we want the fingerprint of the predecessor string of the text substring starting at position i', not the beginning of T). The fingerprint we defined as $\Phi^0_\ell(i')$ above is the fingerprint we want to compare to the fingerprint of the second half of the predecessor string of P_ℓ. Using fingerprint operations, we have from the definitions that $\Phi^0_\ell(i') = \big(\Phi_\ell(i') \ominus \Phi_{\ell-1}(i')\big) \circledcirc \Delta_\ell(i')$, where $\Delta_\ell(i')$ is the set of positions that have to be set to zero. For a substring of T of length $\Theta(m_{\ell-1})$ consider the subset of positions which occur in $\Delta_\ell(i')$ for at least one value of i'. Any such position has a predecessor value greater than $m_{\ell-1}$. By summing over all distinct symbols we have that the size of this subset is only $O(|\Sigma_\text{P}|)$. Thus, we can maintain in small space every position in a suitable length window that will *ever* have to be set to zero.

Let us go back to the example where the process for $P_{\ell-1}$ had found a p-match at position i'. The process stores i' along with the fingerprint $\Phi_{\ell-1}(i')$. This information is not needed by the process for P_ℓ until $m_{\ell-1}$ text symbols later. During the arrival of these symbols, the process for $P_{\ell-1}$ might detect more p-matches, in fact many more matches. Their positions and corresponding fingerprints have to be stored until needed by the process for P_ℓ. We now have a space issue: how do we store this information in small space? To appreciate this question, first consider exact matching. Here matches are known to be either an exact period length apart or very far apart. The matching positions can therefore be represented by an arithmetic progression. Further, the fingerprints associated with the matches in an arithmetic progression can easily be stored succinctly as one can work out each one of the fingerprints from the first one. For p-matching the situation is much more complex: matches can occur more chaotically and, as we have seen above, fingerprints must be updated dynamically to reflect that symbols could be mapped differently in two distinct alignments. Handling these difficulties in small space (and small time complexity) is a main hurdle and is one point at which our work differ significantly from all previous work on streaming matching in small space. We cope with this space issue in the next section.

2.1 The structure of parameterized matches

First recall that an *arithmetic progression* is a sequence of numbers such that the (common) difference between any two successive numbers is constant. We can specify an arithmetic progression by its start number, the common difference and the length of the sequence. In the next lemma we will see that the positions at which a string P of length m p-matches a longer string of length $3m/2$ can be stored in small memory: either a matching position belongs to an arithmetic progression or it is one of relatively few positions that can be listed explicitly in $O(|\Sigma_\text{P}|)$ space. The proof (consult Figure 2) is deferred to Section 5.

▶ **Lemma 5.** *Let X be the set of positions at which P p-matches within an $3m/2$ length substring of T. The set X can be partitioned into two sets Y and $\mathcal{A}$ such that $|Y| \leqslant 6|\Sigma_\text{P}|$, $\max(Y) < \min(\mathcal{A})$ and $\mathcal{A}$ is an arithmetic progression with common difference ρ, where ρ is the p-period of P.*

The lemma is incredibly important for the algorithm as it allows us to store all partial matches (that need to be kept in memory before being discarded) in a total of $O(|\Sigma_\text{P}| \log m)$

space across all processes. The question of how to store their associated fingerprints remains, but is nicely resolved with the corollary below that follows immediately from the proof of Lemma 5. We can afford to store fingerprints explicitly for the positions that are identified to belong to the set Y from Lemma 5, and for the matching positions in the arithmetic progression $\mathcal{A}$ we can, as for exact matching, work out every fingerprint given the first one.

▶ **Corollary 6.** *For pattern P, text T and arithmetic progression $\mathcal{A}$ as specified in Lemma 5,* $\text{pred}(T)[(i+m-\rho),\ (i+m-1)]$ *is the same for all $i \in \mathcal{A}$.*

2.2 Deamortisation

So far we have described the overall approach but it is of course a major concern how to carry out computations in constant time per arriving symbol. In order to *deamortise* the algorithm, we run a separate process responsible for the pattern prefix P_0 that uses the deterministic algorithm of Section 4 (i.e. Theorem 4). As P_0 has p-period greater than 3δ, the p-matches it outputs are at least this far apart. This enables the other processes to operate with a small delay: process P_ℓ expects process $P_{\ell-1}$ to hand over matches and fingerprints with a small delay, and it will itself hand over matches and fingerprints to $P_{\ell+1}$ with a small delay. One of the reasons for the delays is that processes operate in a round-robin scheme – one process per arriving symbol. The process that is responsible for P_s (which has length $m-4\delta$) returns matches with a delay of up to 3δ arriving symbols. Hence there is a gap of length δ in which we can work out if the whole of P matches. To do this we have another process that runs in parallel with all other processes and explicitly checks if any match with P_s can be extended with the remaining 4δ symbols by directly comparing their predecessor values with the last 4δ predecessor values of the pattern. This job is spread out over δ arriving symbols, hence matches with P are outputted in constant time.

3 The main algorithm

We are now in a position to describe the full algorithm of Theorem 1. Recall that the algorithm will find p-matches with each of the pattern prefixes $P_0, \ldots, P_s$ defined in the previous section. If a shorter prefix fails to match at a given position then there is no need to check matches for longer prefixes. Our algorithm runs three main processes concurrently which we label A, B and C. The term process had a slightly different meaning in the previous section, but hopefully this will cause no confusion. Each process takes $O(1)$ time per arriving symbol. Recall that both the pattern and text alphabets are $\Sigma_\text{P} = \{0, \ldots, |\Sigma_\text{P}|-1\}$. **Process A** finds p-matches with prefix P_0 which are inserted as they occur into a *match queue* M_0. **Process B** finds p-matches for prefixes $P_1, \ldots, P_s$ which are inserted into the match queues $M_1, \ldots, M_s$, respectively. The p-matches are inserted with a delay of up to 3δ symbol arrivals after they occur. **Process C** finds p-matches with the whole pattern P which are outputted in constant time as they occur as described in Section 2.2.

It is crucial for the space usage that the match queues $M_0, M_1, \ldots, M_s$ will be stored in a compressed fashion. The delay in detecting p-matches with P_ℓ in Process B is a consequence of deamortising the work required to find a prefix match, which we spread out over $\Theta(\delta)$ arriving symbols. We can afford to spread out the work in this way because the p-period of $P_{\ell-1}$ is at least δ so any p-matches are at least this far apart.

Throughout this section we assume that $m > 14\delta$ so that $m_\ell - m_{\ell-1} \geqslant 3\delta$ for $\ell \in \{1, \ldots, s\}$. If $m \leqslant 14\delta$, or the p-period of P is 3δ or less, we use the deterministic algorithm presented in Section 4 to solve the problem within the required bounds.

3.1 Process A (finding matches with P_0)

From the definition of P_0 we have that if we remove the final character (giving the string $P[0,\ m_0 - 2]$) then its p-period is at most 3δ. The p-period of P_0 itself could be much larger. As part of process A we run the deterministic pattern matching algorithm from Section 4 (see Theorem 4) on $P[0,\ m_0 - 2]$. It returns p-matches in constant time and uses $O(|\Sigma_\text{P}| + 3\delta) = O(|\Sigma_\text{P}| \log m)$ space.

In order to establish matches with the whole of P_0 we handle the final character separately. If the deterministic subroutine reports a match that ends in $T[i-1]$, when $T[i]$ arrives we have a p-match with P_0 if and only if $\text{pred}(T)[i] = \text{pred}(P_0)[m_0 - 1]$ (or $\text{pred}(T)[i] \geqslant m_0$ if $\text{pred}(P_0)[m_0 - 1] = 0$). As the alphabet is of the form $\Sigma_\text{P} = \{0, \dots |\Sigma_\text{P}| - 1\}$, we can compute the value of $\text{pred}(T)[i]$ in $O(1)$ time by maintaining an array A of length $|\Sigma_\text{P}|$ such that for all $\sigma \in \Sigma_\text{P}$, $A[\sigma]$ gives the index of the most recent occurrence of symbol σ.

Whenever Process A finds a match with P_0 at position i' of the text, the pair $(i', \Phi_0(i'))$ is added to a (FIFO) queue M_0, which is queried by Process B when handling prefix P_1.

3.2 Process B (finding matches with $P_1, \dots, P_s$)

We split the discussion of the execution of Process B into s *levels*, $1, \dots, s$. For each level ℓ the fingerprint $\Phi^0_\ell(i')$ is computed for each position i' at which $P_{\ell-1}$ p-matches. Then, as discussed in Section 2, if $\Phi^0_\ell(i') = \phi(\text{pred}(P_\ell)[m_{\ell-1},\ (m_\ell - 1)])$, there is also a match with P_ℓ at i'. The algorithm will in this case add the pair $(i', \Phi_\ell(i'))$ to the queue M_ℓ which is subject to queries by level $\ell + 1$. To this end we compute $\Phi_\ell(i') \ominus \Phi_{\ell-1}(i')$ and $\Delta_\ell(i')$, where $\Delta_\ell(i')$ contains all the positions which should be zeroed in order to obtain $\Phi^0_\ell(i')$. In the example of Figure 1, $\Delta_\ell(i') = \{1, 5, 7\}$ (the `d`, `e` and `f`, respectively).

In order for process B to spend only constant time per arriving symbol, all its work must be scheduled carefully. The preparation of the $\Delta_\ell(i')$ values takes place as a subprocess we name B1. Computing $\Phi_\ell(i') \ominus \Phi_{\ell-1}(i')$ and establishing matches takes place in another subprocess named B2. The two subprocesses are run in sequence for each arriving symbol.

Subprocess B1 (prepare zeroing) We use a queue D_ℓ associated with each level l which contains the most recent $O(|\Sigma_\text{P}|)$ positions with predecessor the values greater than $m_{\ell-1}$. We will see below that $\Delta_\ell(i')$ is a subset of the positions in D_ℓ (adjusted to the offset i').

Unfortunately, in the worst case, for an arriving symbol $T[i]$, i could belong to all of the D_ℓ queues. Since we can only afford constant time per arriving symbol, we cannot insert i into more than a constant number of queues. The solution is to buffer arriving symbols. When some $T[i]$ arrives we first check whether $\text{pred}(T)[i] > m_0$. If so, the pair $(i, \text{pred}(T)[i])$ is added to a buffer $\mathcal{B}$ to be dealt with later. Together with the pair we also store the value $r^i \bmod p$ which will be needed to perform the required zeroing operations.

In addition to adding a new element to the buffer $\mathcal{B}$, the Subprocess B1 will also process elements from $\mathcal{B}$. If is is currently not in the state of processing an element, it will now start doing so by removing an element from $\mathcal{B}$ (unless $\mathcal{B}$ is empty). Call this element $(j, \text{pred}(T)[j])$. Over the next s arriving symbols the Subprocess B1 will do the following. For each of the s levels ℓ, if $\text{pred}(T)[j] > m_{\ell-1}$, add $(j, \text{pred}(T)[j])$ to the queue D_ℓ. If D_ℓ contains more than $12|\Sigma_\text{P}|$ elements, discard the oldest.

Subprocess B2 (establish matches) This subprocess schedules the work across the levels in a round-robin fashion by only considering level $\ell = 1 + (i \bmod s)$ when the symbol $T[i]$ arrives. Potential matches may not be reported by this subprocess until up to 3δ arriving symbols after they occur. As $P_{\ell-1}$ has p-period at least 3δ, the processing of potential matches does not overlap.

The Subprocess B2 for level ℓ is always in one of two states: either it is *checking* whether a matching position i' for $P_{\ell-1}$ is also a match with P_ℓ, or it is *idle*. If idle, level ℓ looks into queue $M_{\ell-1}$ which holds matches with $P_{\ell-1}$. If $M_{\ell-1}$ is non-empty, level ℓ removes an element from $M_{\ell-1}$, call this element $(i', \Phi_{\ell-1}(i'))$, and enters the checking state. Whenever $i > i' + m_\ell + \delta$, level ℓ will start checking if i' is also a matching position with P_ℓ. It does so by first computing the fingerprint $\Phi_\ell(i') \ominus \Phi_{\ell-1}(i')$, which by definition equals $\big(\Phi_\ell(i') - \Phi_{\ell-1}(i')\big)r^{-i'-m_{\ell-1}} \bmod p$. We can ensure the fingerprint $\Phi_\ell(i')$ is always available when needed by maintaining a circular buffer of the most recent $\Theta(\delta)$ fingerprints of the text. Similarly we can obtain $r^{-i'-m_{\ell-1}} \bmod p$ in $O(1)$ time by keeping a buffer of the most recent $\Theta(\delta)$ values of $r^{-i} \bmod p$ along with $r^{-m_\ell} \bmod p$ for all ℓ.

Over the next at most $|\Sigma_{\mathrm{P}}|$ arriving symbols for which Subprocess B2 is considering level ℓ (i.e. those with $\ell = 1 + (i \bmod s)$), $\Phi^0_\ell(i')$ will be computed from $\Phi_\ell(i') \ominus \Phi_{\ell-1}(i')$ by stepping through the elements of the queue D_ℓ. For any element $(j, \mathrm{pred}(T)[j]) \in D_\ell$, we have that $(j - i' - m_{\ell-1}) \in \Delta_\ell(i')$ if and only if $\mathrm{pred}(T)[j] > j - i'$. Further, as Subprocess B1 stored $r^j \bmod p$ with the element in D_ℓ and $r^{i'} \bmod p$ is obtained through the circular buffer as above, we can perform the zeroing in $O(1)$ time.

Having computed $\Phi^0_\ell(i')$, we then compare it to $\phi(\mathrm{pred}(P_\ell)[m_{\ell-1},\ (m_\ell - 1)])$. If they are equal, we have a p-match with P_ℓ at position i' of the text, and the pair $(i', \Phi_\ell(i'))$ is added to the queue M_ℓ. This occurs before $T[i' + m_\ell + 3\delta]$ arrives.

3.3 Correctness, time and space analysis

The time and space complexity almost follow immediately from the description of our algorithm, but correctness requires further attention. In particular one has to show that buffers do not overflow, elements in queues are dealt with before being discarded and every possible match will be found (disregarding the probabilistic error in the fingerprint comparisons).

► **Lemma 7.** *The algorithm described above proves Theorem 1.*

Proof. Coupled with the discussion in Section 2, the time and space complexity almost follow immediately from the description. It only remains to show that, at any time, $|\mathcal{B}| \leqslant |\Sigma_{\mathrm{P}}|$. First observe that any symbol $\sigma \in \Sigma_T$ is only inserted into $\mathcal{B}$ when $\mathrm{pred}(T)[i] > m_0 > \delta$ which can only happen at most once in every $\delta = |\Sigma_{\mathrm{P}}| \log m$ arriving symbols. Further we remove one element every $s \leqslant \lceil \log m \rceil$ arrivals and in particular remove the σ occurrence after at most $|\mathcal{B}|\lceil \log m \rceil$ arrivals. As $\mathcal{B}$ is initially empty, by induction it follows that no symbol occurs more than once in $\mathcal{B}$.

For correctness, it remains to show that we correctly obtain the positions of $\Phi^0_\ell(i')$ from D_ℓ. It follows from the description that all positions of $\Phi^0_\ell(i')$ correspond to elements inserted into D_ℓ at some point. However we need to prove that these elements are present in D_ℓ while $\Phi^0_\ell(i')$ is calculated. Any element inserted into $\mathcal{B}$ during $T[i',\ (i' + m_\ell - 1)]$ has cleared the buffer by the end of interval B (which has length δ) by the argument above. Therefore any relevant element has been inserted into D_ℓ by the start of interval C, during which we calculate $\Phi^0_\ell(i')$. Any element inserted into D_ℓ is at least $m_{\ell-1}$ characters from its predecessor. Therefore, summing over all symbols in the alphabet, there are at most $4|\Sigma_P|$ positions in $T[i',\ (i' + 2m_\ell - 1)]$ which are inserted into D_ℓ. As D_ℓ is a FIFO queue of size $12|\Sigma|$, the relevant elements are still present after interval C. As commented above, potential matches in M_ℓ are separated by more than 3δ arrivals because $P_{\ell-1}$ has p-period more than 3δ. They are processed within 3δ arrivals so M_ℓ does not overflow. ◄

4 The deterministic matching algorithm

We now describe the deterministic algorithm that solves Theorem 4. Its running time is $O(1)$ time per character and it uses $O(|\Sigma_P| + \rho)$ words of space, where ρ is the parameterized period of P. We require that both the pattern and text alphabets are $\Sigma_P = \{0, \ldots, |\Sigma_P|-1\}$.

We first briefly summarise the overall approach of [1] which our algorithm follows. It resembles the classic KMP algorithm. When $T[i]$ arrives, the overall goal is to calculate the largest r such that $P[0,\ r-1]$ p-matches $T[(i-r+1),\ i]$. A p-match occurs iff $r = m$. When a new text character $T[i+1]$ arrives the algorithm compares $\text{pred}(P)[r]$ to $\text{pred}(T)[i+1]$ in $O(1)$ time to determine whether $P[0,\ r]$ p-matches $T[(i-r+1),\ i+1]$. More precisely, the algorithm checks whether either $\text{pred}(P)[r] = \text{pred}(T)[i+1]$, or $\text{pred}(P)[r] = 0 \wedge \text{pred}(T)[i+1] > r$. The second case covers the possibility that the previous occurrence in the text was outside the window. If there is a match, we set $r \leftarrow r+1$ and $i \leftarrow i+1$ and continue with the next text character. If not, we shift the pattern prefix $P[0,\ r-1]$ along by its p-period, denoted ρ_{r-1}, so that it is aligned with $T[(i-r+\rho_{r-1}+1),\ i]$. This is the next candidate for a p-match. In the original algorithm, the p-periods of all prefixes are stored in an array of length m called a prefix table.

Our main hurdle here is to store both a prefix table suitable for p-matching as well as an encoding of the pattern in only $O(|\Sigma_P| + \rho)$ space, while still allowing efficient access to both. It is well-known that any string P can be stored in space proportional to its exact period. In Lemma 9, which follows from Lemma 8, we show an analogous result for $\text{pred}(P)$.

▶ **Lemma 8.** *For any $j \in [\rho]$ there is a constant k_j such that $\text{pred}(P)[j+k\rho]$ is 0 for $k < k_j$, and c_j for $k \geqslant k_j$, where $c_j > 0$ is a constant that depends on j.*

▶ **Lemma 9.** *The predecessor string $\text{pred}(P)$ can be stored in $O(\rho)$ space, where ρ is the p-period of P. Further, for any $j \in [m]$ we can recover $\text{pred}(P)[j]$ in $O(1)$ time.*

We now explain how to store the parameterized prefix table in only $O(\rho)$ space, in contrast to $\Theta(m)$ space which a standard prefix table would require. The p-period ρ_r of $P[0,\ r]$ is, as a function of r, non-decreasing in r. This property enables us to run-length encode the prefix table and store it as a doubly linked list with at most ρ elements, hence using only $O(\rho)$ space. Each element corresponds to an interval of prefix lengths with the same p-period, and the elements are linked together in increasing order (of the common p-period). This does not allow $O(1)$ time random access to the p-period of any prefix, but for our purposes it will suffice to perform sequential access. To accelerate computation we also store a second linked list of the indices of the first occurrences of each symbol in P in ascending order, i.e. every j such that $\text{pred}(P)[j] = 0$. This uses $O(|\Sigma_P|)$ space.

There is a crucial second advantage to compressing the prefix table which is that it allows us to upper bound the number of prefixes of P we need to inspect when a mismatch occurs. When a mismatch occurs in our algorithm, we repeatedly shift the pattern until a p-match between a text suffix and pattern prefix occurs. Naively it seems that we might have to check many prefixes within the same run. However, as a consequence of Lemma 8 we are assured that if some prefix does not p-match, every prefix in the same run with $\text{pred}(P)[j] \neq 0$ will also mismatch (except possibly the longest). Therefore we can skip inspecting these prefixes. This can be seen by observing (using Lemma 8) that for j such that $\rho_j = \rho_{j+1}$, we have $\text{pred}(P)[j-\rho_j] \in \{0,\ \text{pred}(P)[j]\}$. By keeping pointers into both linked lists, it is straightforward to find the next prefix to check in $O(1)$ time. Whenever we perform a pattern shift we move at least one of the pointers to the left. Therefore the total number of pattern shifts inspected while processing $T[i]$ is at most $O(|\Sigma_P| + \rho)$. As each pointer only moves to the right by at most one when each $T[i]$ arrives, an amortised time complexity of $O(1)$ per character follows. The space usage is $O(|\Sigma_{\mathrm{P}}| + \rho)$, dominated by the linked lists.

We now briefly discuss how to deamortise our solution by applying Galil's KMP deamortisation argument [11]. The main idea is to restrict the algorithm to shift the pattern at most twice when each text character arrives, giving a constant time algorithm. If we have not finished processing $T[i]$ by this point we accept $T[i+1]$ but place it on the end of a buffer, output 'no match' and continue processing $T[i]$. The key property is that the number of text arrivals until the next p-match occurs is at least the length of the buffer. As we shift the pattern up to twice during each arrival we always clear the buffer before (or as) the next p-match occurs. Further, the size of the buffer is always $O(|\Sigma_{\mathrm{P}}| + \rho)$. This follows from the observation above that the number of pattern shifts required to process a single text character is $O(|\Sigma_{\mathrm{P}}| + \rho)$. This concludes the algorithm of Theorem 4.

5 The proof of Lemma 5

In this section we prove the important Lemma 5. Let i_{left} denote an arbitrary position in T where P p-matches. Let X be the set of positions at which P p-matches within $T[i_{\text{left}},\ (i_{\text{left}} + 3m/2 - 1)]$. We now prove that there exist disjoint sets Y and $\mathcal{A}$ with the properties set out in the statement of the lemma.

Let α be the smallest integer such that all distinct symbols in P occur in the prefix $P[0,\ \alpha]$. We first show that ρ, the p-period of P is at least $\alpha/|\Sigma|$. From the minimality of α, we have that $P[\alpha]$ is the leftmost occurrence of some symbol. By the definition of the p-period, we have that $P[0,\ (m-1-\rho)]$ p-matches $P[\rho,\ m-1]$. Under this shift, $P[\alpha]$ (in $P[\rho,\ m-1]$) is aligned with $P[\alpha-\rho]$ (in $P[0,\ (m-1-\rho)]$) . Assume that $P[\alpha-\rho]$ is not a leftmost occurrence and let j be the position of the previous occurrence of $P[j] = P[\alpha-\rho]$. As a p-match occurs, we have that $P[j] = P[j+\alpha] \neq P[\alpha]$, contradiction. By repeating this argument we find distinct symbols at positions $\alpha - k\rho$ for all $k > 0$, implying that $\rho \geqslant \alpha/|\Sigma|$.

We first deal with two simple cases: $\rho > m/8$ or $\alpha \geqslant m/4$ (which implies that $\rho \geqslant m/(4|\Sigma|)$). In these two cases the number of p-matches is easily upper bounded by $6|\Sigma|$, so all positions can be stored in the set Y. We therefore continue under the assumption that $\alpha < m/4$ and $\rho < m/8$. As $\rho \geqslant \alpha/|\Sigma|$, there are at most $(\alpha+1)/(\alpha/|\Sigma|) \leqslant 2|\Sigma|$ positions from the range $[i_{\text{left}},\ i_{\text{left}}+\alpha]$ at which P can p-match T. We can store these positions in the set Y. Next we will show that the positions from the range $[(i_{\text{left}}+\alpha+1),\ (i_{\text{left}}+3m/2-1)]$ at which P p-matches T can be represented by the arithmetic progression $\mathcal{A}$.

First we show that ρ is an *exact period* (not p-period) of $\mathrm{pred}(P)[\alpha+1,\ m-1]$ (but not necessarily the shortest period). Consider arbitrary positions $P[j]$ and $P[j-\rho]$ where $\alpha < j < m-\rho$. By the definition of the p-period, we have that $P[\rho,\ m-1]$ p-matches $P[0,\ (m-1-\rho)]$ and hence that $\mathrm{pred}(P[\rho,\ m-1]) = \mathrm{pred}(P[0,\ (m-1-\rho)])$. In particular, $\mathrm{pred}(P[\rho,\ m-1])[j] = \mathrm{pred}(P[0,\ (m-1-\rho)])[j] = \mathrm{pred}(P)[j]$, where the second equality follows because we take the predecessor string of a prefix of P. Also observe that $\mathrm{pred}(P[\rho,\ m-1])[j]$ either equals 0 or $\mathrm{pred}(P)[j-\rho]$ by definition. Further, $\mathrm{pred}(P[0,\ (m-1-\rho)])[j] = \mathrm{pred}(P)[j] \neq 0$ as $j > \alpha$ and all leftmost occurrences are before α. This implies that $\mathrm{pred}(P[\rho,\ m-1])[j] \neq 0$, hence, $\mathrm{pred}(P)[j-\rho] = \mathrm{pred}(P[\rho,\ m-1])[j] = \mathrm{pred}(P[0,\ (m-1-\rho)])[j] = \mathrm{pred}(P)[j]$.

Recall that P p-matches $T[i_{\text{left}},\ i_{\text{left}}+m-1]$ so $\mathrm{pred}(P) = \mathrm{pred}(T[i_{\text{left}},\ i_{\text{left}}+m-1])]$ and hence ρ is an exact period of $\mathrm{pred}(T[i_{\text{left}},\ i_{\text{left}}+m-1])[\alpha+1,\ m-1]$. Let $j \in \{\alpha+1, \ldots, m-2\}$ and observe that by definition, $\mathrm{pred}(T[i_{\text{left}},\ i_{\text{left}}+m-1])[j] \in \{0, \mathrm{pred}(T)[i_{\text{left}}+j]\}$. However, $\mathrm{pred}(T[i_{\text{left}},\ (i_{\text{left}}+m-1)])[j] = \mathrm{pred}(P)[j] > 0$ because $j > \alpha$ and all leftmost occurrences are in $P[0,\ \alpha]$. This implies that $\mathrm{pred}(T[i_{\text{left}},\ (i_{\text{left}}+m-$

$1)])[j] = \mathrm{pred}(T)[i_{\mathrm{left}} + j]$. As j was arbitrary, we have that $\mathrm{pred}(T)[(i_{\mathrm{left}} + \alpha + 1),\ (i_{\mathrm{left}} + m - 1)] = \mathrm{pred}(T[i_{\mathrm{left}},\ (i_{\mathrm{left}} + m - 1)])[\alpha + 1,\ m - 1]$ and hence ρ is an exact period of $\mathrm{pred}(T)[(i_{\mathrm{left}} + \alpha + 1),\ (i_{\mathrm{left}} + m - 1)]$.

Let i_{right} be the rightmost position in $T[i_{\mathrm{left}},\ i_{\mathrm{left}} + 3m/2 - 1]$ where P p-matches. By the same argument as for i_{left}, we have that ρ is an exact period of $\mathrm{pred}(T)[(i_{\mathrm{right}} + \alpha + 1),\ (i_{\mathrm{right}} + m - 1)]$. Thus, both $\mathrm{pred}(T)[(i_{\mathrm{left}} + \alpha + 1),\ (i_{\mathrm{left}} + m - 1)]$ and $\mathrm{pred}(T)[(i_{\mathrm{right}} + \alpha + 1),\ (i_{\mathrm{right}} + m - 1)]$ has an exact period of ρ. As these two strings overlap by at least ρ characters, we have that ρ is also an exact period of $\mathrm{pred}(T)[i_{\mathrm{left}} + \alpha + 1,\ i_{\mathrm{right}} + m - 1]$.

Let $i \in \{(i_{\mathrm{left}} + \alpha + 1), \ldots, i_{\mathrm{right}} - 1\}$ be arbitrary such that P p-matches $T[i,\ (i + m - 1)]$. We now prove that if $i + \rho < i_{\mathrm{right}}$ then P p-matches $T[i + \rho,\ (i + \rho + m - 1)]$. As p-matches must be at least ρ characters apart this is sufficient to conclude that all remaining matches form an arithmetic progression with common difference ρ. As ρ is an exact period of $\mathrm{pred}(T)[(i_{\mathrm{left}} + \alpha + 1),\ (i_{\mathrm{right}} + m - 1)]$, we have that $\mathrm{pred}(T)[i,\ (i + m - 1)] = \mathrm{pred}(T)[i + \rho,\ (i + \rho + m - 1)]$. By definition, this implies that $\mathrm{pred}(T[i,\ (i + m - 1)]) = \mathrm{pred}(T[i + \rho,\ (i + \rho + m - 1)])$ and hence a p-match also occurs at $i + \rho$.

References

1 Amihood Amir, Martin Farach, and S. Muthukrishnan. Alphabet dependence in parameterized matching. *IPL*, 49(3):111 – 115, 1994.

2 Arne A. Andersson and Mikkel Thorup. Tight(er) worst-case bounds on dynamic searching and priority queues. In *STOC '00*, pages 335–342, 2000.

3 B. S. Baker. A theory of parameterized pattern matching: algorithms and applications. In *STOC '93*, pages 71–80, 1993.

4 Brenda S. Baker. Parameterized pattern matching by boyer-moore-type algorithms. In *SODA '95*, pages 541–550, 1995.

5 Brenda S. Baker. Parameterized pattern matching: Algorithms and applications. *Journal of Computer and System Sciences*, 52(1):28 – 42, 1996.

6 Brenda S. Baker. Parameterized duplication in strings: Algorithms and an application to software maintenance. *SIAM J. on Comp.*, 26(5):1343–1362, 1997.

7 Dany Breslauer and Zvi Galil. Real-time streaming string-matching. In *CPM '11*, pages 162–172, 2011.

8 Raphaël Clifford, Klim Efremenko, Benny Porat, and Ely Porat. A black box for online approximate pattern matching. In *CPM '08*, pages 143–151, 2008.

9 Raphaël Clifford, Markus Jalsenius, Ely Porat, and Benjamin Sach. Space lower bounds for online pattern matching. In *CPM '11*, pages 184–196, 2011.

10 F. Ergun, H. Jowhari, and M. Sağlam. Periodicity in streams. In *RANDOM '10*, pages 545–559, 2010.

11 Zvi Galil. String matching in real time. *Journal of the ACM*, 28(1):134–149, 1981.

12 Carmit Hazay, Moshe Lewenstein, and Dina Sokol. Approximate parameterized matching. *ACM Trans. Algorithms*, 3(3), 2007.

13 Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res Dev*, 31(2):249 –260, 1987.

14 D. E. Knuth, J. H. Morris, and V. B. Pratt. Fast pattern matching in strings. *SIAM J. on Comp.*, 6:323–350, 1977.

15 Benny Porat and Ely Porat. Exact and approximate pattern matching in the streaming model. In *FOCS '09*, pages 315–323, 2009.

Popular Matchings: Structure and Cheating Strategies*

Meghana Nasre[1]

1 University of Texas at Austin, USA.

Abstract

We consider the cheating strategies for the popular matchings problem. Let $G = (\mathcal{A} \cup \mathcal{P}, E)$ be a bipartite graph where $\mathcal{A}$ denotes a set of agents, $\mathcal{P}$ denotes a set of posts and the edges in E are ranked. Each agent ranks a subset of posts in an order of preference, possibly involving ties. A matching M is popular if there exists no matching M' such that the number of agents that prefer M' to M exceeds the number of agents that prefer M to M'. Consider a centralized market where agents submit their preferences and a central authority matches agents to posts according to the notion of popularity. Since a popular matching need not be unique, we assume that the central authority chooses an arbitrary popular matching. Let a_1 be the sole manipulative agent who is aware of the true preference lists of all other agents. The goal of a_1 is to falsify her preference list to get *better always*, that is, to improve the set of posts she gets matched to in the falsified instance. We show that the optimal cheating strategy for a single agent to get *better always* can be computed in $O(m + n)$ time when preference lists are all strict and in $O(\sqrt{n}m)$ time when preference lists are allowed to contain ties. Here $n = |\mathcal{A}| + |\mathcal{P}|$ and $m = |E|$.

To compute the cheating strategies, we develop a *switching graph* characterization of the popular matchings problem involving ties. The switching graph characterization was studied for the case of strict lists by McDermid and Irving (J. Comb. Optim. 2011) and was open for the case of ties. We show an $O(\sqrt{n}m)$ time algorithm to compute the set of *popular pairs* using the switching graph. These results are of independent interest and answer a part of the open questions posed by McDermid and Irving.

1998 ACM Subject Classification G.2.1 Combinatorial algorithms G.2.2 Graph algorithms

Keywords and phrases bipartite matchings, preferences, cheating strategies

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.412

1 Introduction

We consider the cheating strategies for the popular matchings problem. Let $G = (\mathcal{A} \cup \mathcal{P}, E)$ be a bipartite graph where $\mathcal{A}$ denotes a set of agents, $\mathcal{P}$ denotes a set of posts, and the edges in E are ranked. Each agent ranks a subset of posts in an order of preference, possibly involving ties. This ranking of posts by an agent is called the preference list of the agent. An agent a prefers post p_i to post p_j if the rank of post p_i is smaller than the rank of post p_j in a's preference list. An agent a is indifferent between posts p_i and p_j if they have the same rank on a's preference list. When agents can be indifferent between posts, the preference lists are said to contain ties, otherwise the preference lists are strict. A matching M of G is a subset of edges, no two of which share an end point. For a matched vertex u, let $M(u)$ denote its partner in the matching M. An agent a prefers a matching M to another matching M' if (i) a is matched in M but unmatched in M', or (ii) a prefers $M(a)$ to $M'(a)$.

* This work was supported in part by NSF grant CCF-0830737.

▶ **Definition 1.** A matching M is *more popular than* M' if the number of agents that prefer M is greater than the number of agents that prefer M'. A matching M is *popular* if there is no matching M' that is more popular than M.

There exist simple instances that do not admit any popular matching – however, when an instance admits a popular matching, there may be more than one popular matching. Abraham et al. [1] characterized the instances that admit popular matchings and gave efficient algorithms to compute a popular matching if one exists.

Our problem. Consider a centralized matching market where each agent $a \in \mathcal{A}$ submits a preference over a subset of posts and a central authority matches agents to posts using the criteria of popularity. Let a_1 be the sole manipulative agent who is aware of the true preference lists of all other agents and the preference lists of $a \in \mathcal{A} \setminus \{a_1\}$ remain fixed throughout. The goal of a_1 is clear: she wishes to falsify her preference list so as to improve the post that she gets matched to as compared to the post she got when she was truthful. Since there may be more than one popular matching in an instance, we assume that the central authority chooses an arbitrary popular matching. Let $G = (\mathcal{A} \cup \mathcal{P}, E)$ denote the instance where ranks on the edges represent true preferences of all the agents. Let H denote the instance obtained by falsifying the preference list of a_1 alone. We assume that G admits a popular matching and a_1 falsifies in order to create an instance H which also admits a popular matching. Note that it may be possible for a_1 to falsify her preference list such that H does not admit any popular matching. But we do not consider such a falsification.

Agent a_1 wishes to falsify her preference list to ensure that (i) every popular matching in H matches her to a post that is at least as good as the most-preferred post that she gets matched to in G, and (ii) some popular matching in H matches a_1 to a post better than the most-preferred post p that she gets matched to in G, assuming that p is not a_1's true first choice post. We term this strategy of a_1 as '*better always*' strategy.

1.1 Our contributions

- Let a_1 be the sole manipulative agent who wishes to get *better always*. The optimal strategy for a_1 can be computed in $O(m + n)$ time when preference lists are all strict and in $O(\sqrt{n}m)$ time when preference lists are allowed to contain ties.
- To compute the cheating strategies, we develop a *switching graph* characterization of the popular matchings problem involving ties. Such a characterization was studied for the case of strict lists by McDermid and Irving [10] and it was open for the case of ties. Using the switching graph, we show an $O(\sqrt{n}m)$ time algorithm to compute the set of *popular pairs*. An edge $(a, p) \in E$ is a popular pair if there exists a popular matching M in G such that $(a, p) \in M$. We also show that counting the total number of popular matchings in an instance with ties is #P-Complete. The switching graph characterization is of independent interest and answers a part of the open questions in [10].

1.2 Related work

The work in this paper is motivated by the work of Teo et al. [13] where they study the strategic issues of the stable marriage problem [2]. The stable marriage problem is a generalization of our problem where both the sides of the bipartition (usually referred to as men and women) rank members of the opposite side in order of their preference. Teo et al. [13] study the strategic issues of the stable marriage problem where women are required to give complete preference lists and there is a sole manipulative woman. Further, she is aware of the true preference lists of all the other women. Teo et al. [13] compute an optimal cheating strategy for a single woman under this model. Huang [4] studies the strategic issues of the stable room-mates problem [2] under a similar model. In the same spirit, we study the strategic issues of the popular matchings problem.

The notion of popular matchings was introduced by Gärdenfors [3] in the context of the stable marriage [2]. Abraham et al. [1] studied the problem for one-sided preference lists and gave a characterization of instances which admit a popular matching. Subsequent to this result, the popular matchings problem has received a lot of attention [8] [9] [7] [5] [6]. However, to the best of our knowledge none of them is motivated by the strategic issues of the popular matchings problem.

2 Background

We first review the following well known properties of maximum matchings in bipartite graphs. Let $G = (\mathcal{A} \cup \mathcal{P}, E)$ be a bipartite graph and let M be a maximum matching in G. The matching M defines a partition of the vertex set $\mathcal{A} \cup \mathcal{P}$ into three disjoint sets: a vertex $v \in \mathcal{A} \cup \mathcal{P}$ is *even* (resp. *odd*) if there is an even (resp. odd) length alternating path in G w.r.t. M from an unmatched vertex to v. A vertex v is *unreachable* if there is no alternating path from an unmatched vertex to v. Denote by $\mathcal{E}$, $\mathcal{O}$, and $\mathcal{U}$ the sets of even, odd, and unreachable vertices, respectively, in G. The following lemma is well known in matching theory; refer [12] for a detailed exposition and proof.

▶ **Lemma 2** ([12] Dulmage Mendelsohn). *Let $\mathcal{E}$, $\mathcal{O}$, and $\mathcal{U}$ be the sets of vertices defined by a maximum matching M in G. Then,*

(a) $\mathcal{E}$, $\mathcal{O}$, and $\mathcal{U}$ are pairwise disjoint, and independent of the maximum matching M in G.
(b) In any maximum matching of G, every vertex in $\mathcal{O}$ is matched with a vertex in $\mathcal{E}$, and every vertex in $\mathcal{U}$ is matched with another vertex in $\mathcal{U}$. The size of a maximum matching is $|\mathcal{O}| + |\mathcal{U}|/2$.
(c) No maximum matching of G contains an edge between a vertex in $\mathcal{O}$ and a vertex in $\mathcal{O} \cup \mathcal{U}$. Also, G contains no edge between a vertex in $\mathcal{E}$ and a vertex in $\mathcal{E} \cup \mathcal{U}$.

We now review the characterization of the popular matchings problem from [1]. As was done in [1], we create a unique last-resort post $\ell(a)$ for each agent a. In this way, we can assume that every agent is matched, since any unmatched agent a can be paired with $\ell(a)$. For an agent a, let $f(a)$ be the set of rank-1 posts for a. To define $s(a)$, let us consider the graph $G_1 = (\mathcal{A} \cup \mathcal{P}, E_1)$ on rank-1 edges in G and let M_1 be any maximum matching in G_1. Let $\mathcal{O}_1, \mathcal{E}_1, \mathcal{U}_1$ define the partition of vertices $\mathcal{A} \cup \mathcal{P}$ with respect to M_1 in G_1. For any agent a, let $s(a)$ denote the set of most preferred posts which belong to $\mathcal{E}_1$ by the above partition. Abraham et al. [1] proved the following theorem.

▶ **Theorem 3** ([1]). *A matching M is popular in G iff*
(1) $M \cap E_1$ is a maximum matching of $G_1 = (\mathcal{A} \cup \mathcal{P}, E_1)$, and
(2) for each agent a, $M(a) \in \{f(a) \cup s(a)\}$.

The algorithm for solving the popular matching problem is as follows: each $a \in \mathcal{A}$ determines the sets $f(a)$ and $s(a)$. An $\mathcal{A}$-complete matching (a matching that matches all agents) which is maximum in G_1 and matches each a to a post in $\{f(a) \cup s(a)\}$ needs to be determined. If no such matching exists, then G does not admit a popular matching. Abraham et al. [1] gave an $O(\sqrt{n}m)$ time algorithm to compute a popular matching in G which is presented as Algorithm 2.1. Steps 7–11 are added by us and will be used to define the switching graph in Section 3. Abraham et al. [1] also showed a simpler characterization in case of strict lists which results in an $O(m + n)$ time algorithm to return a popular matching if one exists.

Let $G' = (\mathcal{A} \cup \mathcal{P}, E')$ denote the graph in which every agent a has edges incident to $\{f(a) \cup s(a)\}$. Step 4 of Algorithm 2.1 deletes edges from G' which cannot be present in any maximum matching of G_1. We extend this further and in Step 9 delete edges from G' which cannot be present in any popular matching in G. For this, let us partition the vertex set $\mathcal{A} \cup \mathcal{P}$ as $\mathcal{O}_2, \mathcal{E}_2$ and $\mathcal{U}_2$ with respect to a popular matching M in G'. Since any popular matching M is a maximum matching in G', by Lemma 2(c), the matching M cannot contain edges of the form $\mathcal{O}_2\mathcal{O}_2$ and $\mathcal{O}_2\mathcal{U}_2$. However, since M matches every agent, it implies that $\mathcal{A} \cap \mathcal{E}_2 = \emptyset$ and $\mathcal{P} \cap \mathcal{O}_2 = \emptyset$. Thus, there are no $\mathcal{O}_2\mathcal{O}_2$ edges in G'. Hence, any edge (a, p) deleted in Step 9 is of the form $a \in \mathcal{O}_2$ and $p \in \mathcal{U}_2$. We can now claim the following.

▶ Claim 4. Let a be an agent such that $a \in \mathcal{U}_2$. Then, in Step 9 of Algorithm 2.1, no edge incident on a gets deleted. Let a be an agent such that $a \in \mathcal{E}_1$. Then, in Step 4 of Algorithm 2.1, no edge incident on a gets deleted.

Algorithm 2.1 $O(\sqrt{n}m)$-time algorithm for the popular matching problem [1] (Steps 1–6).

```
Input: $G = (\mathcal{A} \cup \mathcal{P}, E)$.
 1: Construct the graph $G' = (\mathcal{A} \cup \mathcal{P}, E')$, where $E' = \{(a, p) : a \in \mathcal{A}$ and $p \in f(a) \cup s(a)\}$.
 2: Construct the graph $G_1 = (\mathcal{A} \cup \mathcal{P}, E_1)$ and let $M_1$ be any maximum matching in $G_1$.
 3: Partition $\mathcal{A} \cup \mathcal{P}$ as $\mathcal{O}_1, \mathcal{E}_1, \mathcal{U}_1$ with respect to $M_1$ in $G_1$.
 4: Remove any edge in $G'$ between a node in $\mathcal{O}_1$ and a node in $\mathcal{O}_1 \cup \mathcal{U}_1$.
 5: Determine a maximum matching $M$ in $G'$ by augmenting $M_1$.
 6: Return $M$ if it is $\mathcal{A}$-complete, otherwise return "no popular matching".
 7: if $G$ admits a popular matching then
 8:    Partition $\mathcal{A} \cup \mathcal{P}$ as $\mathcal{O}_2, \mathcal{E}_2, \mathcal{U}_2$ with respect to $M$ in $G'$.
 9:    Remove any edge in $G'$ between a node in $\mathcal{O}_2$ and a node in $\mathcal{U}_2$.
10:    Denote the resulting graph as $G'' = (\mathcal{A} \cup \mathcal{P}, E'')$.
11: end if
```

▶ **Definition 5.** For $a \in \mathcal{A}$, let $choices(a) = \{p \in \mathcal{P} : (a, p)$ is an edge in $G''\}$.

3 The switching graph characterization

In this section we develop the *switching graph* for the popular matchings problem with ties. In case of strict lists, McDermid and Irving [10] defined a switching graph $G_M = (\mathcal{P}, E_M)$ as a directed graph on the posts of G and the edge set E_M was determined by a popular matching M in G. In fact, a similar graph was defined even before that by Mahdian [8] (again for strict lists) to study existence of popular matchings in random instances.

Let G be an instance of the popular matchings problem with ties and let M be a popular matching in G. The switching graph $G_M = (\mathcal{P}, E_M)$ is a directed weighted graph on the posts $\mathcal{P}$ of G and is defined with respect to a popular matching M in G. The edge set E_M is defined using the pruned graph $G'' = (\mathcal{A} \cup \mathcal{P}, E'')$ constructed in Step 10 of Algorithm 2.1. There exists an edge from p_i to p_j (with $p_i \neq p_j$) iff for some $a \in \mathcal{A}$, $p_i = M(a)$ and $(a, p_j) \in E''$. The weight of an edge $w(M(a), p_j)$ is defined as:

$$\begin{aligned} w(M(a), p_j) &= 0 \quad \text{if } a \text{ is indifferent between } M(a) \text{ and } p_j \\ &= -1 \quad \text{if } a \text{ prefers } M(a) \text{ to } p_j \\ &= +1 \quad \text{if } a \text{ prefers } p_j \text{ to } M(a). \end{aligned}$$

The graph $G_M = (\mathcal{P}, E_M)$ can be easily constructed in $O(\sqrt{n}m)$ time using Algorithm 2.1.

Consider a vertex p in G_M. A post p is a *sink* vertex in G_M if and only if p is unmatched by M in G. This follows from observing that $M(p)$, that is, the agent matched to p by M, has degree at least 2 in the graph G'. Further, any agent continues to have degree at least 2 in the graph G''. We refer the reader to the full version [11] for a detailed proof. Let $\mathcal{X}$ be a maximal weakly connected component of G_M. Call $\mathcal{X}$ a *sink component* if $\mathcal{X}$ contains one or more sink vertices, otherwise call $\mathcal{X}$ a *non-sink component*.

For a path T (resp. cycle C) in G_M, the weight of the path $w(T)$ (resp. $w(C)$) is the sum of the weights on the edges in T (resp. C). (Whenever we refer to paths and cycles in G_M we imply directed paths and directed cycles respectively.) A path $T = \langle p_1, \dots, p_k \rangle$ in G_M is called a *switching path* if T ends in a sink vertex and $w(T) = 0$. Similarly, a cycle $C = \langle p_1, \dots, p_k, p_1 \rangle$ in G_M is called a *switching cycle* if $w(C) = 0$. Let $\mathcal{A}_T = \{a_i : M(p_i) = a_i, \text{ for } i = 1 \dots k\ \}$ and denote by $M' = M \cdot T$ the matching obtained by *applying* the switching path to M, that is, for $a_i \in \mathcal{A}_T$, $M'(a_i) = p_{i+1}$ whereas for $a \notin \mathcal{A}_T$, $M'(a) = M(a)$. Similarly, for a switching cycle C, define $\mathcal{A}_C = \{a_i : M(p_i) = a_i, \text{for } i = 1 \dots k\ \}$ and denote by $M' = M \cdot C$ the matching obtained by *applying* the switching cycle to M, that is, for $a_i \in \mathcal{A}_C$, $M'(a_i) = p_{i+1} \mod k$ whereas for $a \notin \mathcal{A}_C$, $M'(a) = M(a)$.

▶ **Example 6.**

Consider an instance G where $\mathcal{A} = \{a_1, \dots, a_7\}$ and $\mathcal{P} = \{p_1, \dots, p_9\}$. The preference lists of the agents are shown in Figure 1(a). The preference lists can be read as follows: agent a_1 ranks posts p_1, p_2, p_3 as her rank-1, rank-2 and rank-3 posts respectively and the two posts p_6 and p_7 are tied as her rank-4 posts. For every agent a, the posts which are bold denote the set $f(a)$, whereas the posts which are underlined denote the set $s(a)$. The instance G admits a popular matching; M and M' shown below are both popular in G.

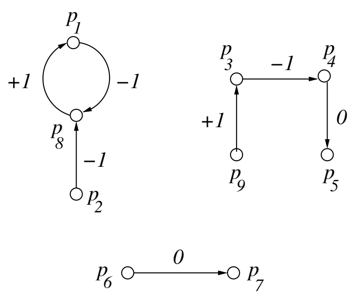
$$M = \{(a_1, p_6), (a_2, p_1), (a_3, p_8), (a_4, p_2), (a_5, p_3), (a_6, p_9), (a_7, p_4)\} \tag{1}$$

$$M' = \{(a_1, p_6), (a_2, p_1), (a_3, p_8), (a_4, p_2), (a_5, p_4), (a_6, p_3), (a_7, p_5)\} \tag{2}$$

Figure 1(b) shows the switching graph G_M with respect to the popular matching M. We note that the edges (a_4, p_3) and (a_1, p_1) get deleted in Step 4 and Step 9 of Algorithm 2.1, respectively. Hence the switching graph G_M does not have the edges $(M(a_4) = p_2, p_3)$ and $(M(a_1) = p_6, p_1)$ respectively. Consider the switching path $T = \langle p_9, p_3, p_4, p_5 \rangle$ in G_M. By *applying* T to M we get $M' = M \cdot T$ (see Equation (2)) which is also popular in G.

$a_1 :$	$\mathbf{p_1}$	p_2	p_3	$(\underline{p_6}, \underline{p_7})$
$a_2 :$	$\mathbf{p_1}$	p_2	$\underline{p_8}$	
$a_3 :$	$\mathbf{p_1}$	$\underline{p_8}$		
$a_4 :$	$(\mathbf{p_2}, \mathbf{p_3})$	p_1	$\underline{p_8}$	
$a_5 :$	$\mathbf{p_3}$	$(p_2, \underline{p_4})$		
$a_6 :$	$\mathbf{p_3}$	$\underline{p_9}$	p_1	
$a_7 :$	$(\underline{\mathbf{p_4}}, \underline{\mathbf{p_5}})$	p_1		

(a)



(b)

Figure 1 (a) Preference lists of agents $\{a_1, \dots, a_7\}$. The posts which are bold denote $f(a)$ and the posts which are underlined denote $s(a)$. (b) Switching graph G_M with respect to the popular matching M in G.

3.1 Some useful properties

In this section we state some useful properties of the switching graph G_M (refer [11] for proof of correctness of these properties). Recall that the vertices $\mathcal{A} \cup \mathcal{P}$ are partitioned as $\mathcal{O}_1, \mathcal{E}_1, \mathcal{U}_1$ w.r.t. a maximum matching M_1 in G_1 (see Step 3 of Algorithm 2.1). Further, the vertices $\mathcal{A} \cup \mathcal{P}$ are partitioned as $\mathcal{O}_2, \mathcal{E}_2, \mathcal{U}_2$ w.r.t. a popular matching M in G' (see Step 8 of Algorithm 2.1).

▶ Property 7. All sink vertices of G_M belong to the set $\mathcal{E}_1$.

▶ Property 8. Every post p belonging to a sink component has a path to a sink and hence belongs to the set $\mathcal{E}_2$. Every post belonging to a non-sink component belongs to the set $\mathcal{U}_2$.

▶ Property 9. For an edge (p_i, p_j) in G_M, the weight $w(p_i, p_j)$ is determined by the partition p_i and p_j belong to when vertices are partitioned as $\mathcal{O}_1, \mathcal{E}_1, \mathcal{U}_1$. The weight $w(p_i, p_j)$ can be determined using Table 1.

Table 1 Table shows $w(p_i, p_j)$ for an edge (p_i, p_j) in G_M. The weight is determined by the partition of vertices as $\mathcal{O}_1, \mathcal{E}_1, \mathcal{U}_1$. The $\times$ denotes that such an edge is not present in G_M.

p_i \ p_j	$\mathcal{O}_1$	$\mathcal{E}_1$	$\mathcal{U}_1$
$\mathcal{O}_1$	0	-1	$\times$
$\mathcal{E}_1$	$+1$	0	$\times$
$\mathcal{U}_1$	$\times$	-1	0

▶ Property 10. Every path T in G_M has $w(T) \in \{-1, 0, +1\}$. Every cycle C in G_M has $w(C) = 0$. There exists no path T in G_M ending in a sink vertex with $w(T) = +1$.

▶ Property 11. For any switching path T (or switching cycle C) in G_M, the matching $M' = M \cdot T$ ($M' = M \cdot C$ resp.) is a popular matching in G. Every popular matching M' in G can be obtained from M by applying to M one or more vertex disjoint switching paths and switching cycles in each of a subset of sink components of G_M together with one or more vertex disjoint switching cycles in each of a subset of the non-sink components of G_M.

Recall the definition of $choices(a)$ for an agent as given by Definition 5. It is easy to see that for any $a \in \mathcal{A}$, $choices(a) \subseteq \{f(a) \cup s(a)\}$. Further, if M is a popular matching in G, then $M(a) \in choices(a)$. We now define the notion of a *tight-pair*, that is, a set of agents $\mathcal{A}_1$ and a set of posts $\mathcal{P}_1$ with $|\mathcal{A}_1| = |\mathcal{P}_1|$. Further, for every $a \in \mathcal{A}_1$ we have $choices(a) \subseteq \mathcal{P}_1$. We show that a tight-pair exists whenever there is a non-sink component in the switching graph G_M.

▶ **Lemma 12.** *Let $\mathcal{Y}$ be a non-sink component in G_M and $q \in \mathcal{Y}$. Let,*

$$\mathcal{P}_q = \{q\} \cup \{p : \ q \text{ has a path to } p \text{ in } G_M \ \}.$$

Then there exists a set of agents $\mathcal{A}_q$ such that (i) $|\mathcal{A}_q| = |\mathcal{P}_q|$, and (ii) for every $a \in \mathcal{A}_q$, $choices(a) \subseteq \mathcal{P}_q$.

Proof. Let $\mathcal{A}_q = \{a : a = M(p) \text{ and } p \in \mathcal{P}_q\}$. Since every $p \in \mathcal{P}_q$ is matched, we note that $|\mathcal{A}_q| = |\mathcal{P}_q|$. For any $a \in \mathcal{A}_q$, we have $M(a) \in \mathcal{P}_q$ and note that $M(a) \in choices(a)$. Further, note that, for every $p' \in choices(a) \setminus \{M(a)\}$, we have an edge $(M(a), p')$ in G_M. Thus, every such p' also belongs to $\mathcal{P}_q$. This proves that for every $a \in \mathcal{A}_q$, $choices(a) \subseteq \mathcal{P}_q$. ◀

3.2 Generating popular pairs and counting popular matchings

Let $G = (\mathcal{A} \cup \mathcal{P}, E)$ be an instance of the popular matchings problem. Define

$$PopPairs = \{(a, p) \in E : \; M \text{ is a popular matching in } G \text{ and } M(a) = p\}. \qquad (3)$$

Using the switching graph defined in the previous section, it is easy to compute the set $PopPairs$ in G. Let G_M be the switching graph with respect to a popular matching M in G. From *Property 11* we can conclude that an edge $e = (a, p)$ is a popular pair if and only if (i) $e \in M$ or, (ii) the edge $(M(a), p)$ belongs to some switching path in G_M or, (iii) the edge $(M(a), p)$ belongs to some switching cycle in G_M.

We note that edges satisfying condition (i) can be marked in $O(\sqrt{n}m)$ time using Algorithm 2.1 and edges satisfying conditions (ii) or (iii) can be marked in linear time in the size of the switching graph. Thus, we conclude the following theorem (see [11] for proof).

▶ **Theorem 13.** *The set of popular pairs for an instance $G = (\mathcal{A} \cup \mathcal{P}, E)$ of the popular matchings problem with ties can be computed in $O(\sqrt{n}m)$ time.*

We now show that given an instance of the popular matchings problem with ties, the problem of counting the number of popular matchings is #P-Complete. The result readily follows by (i) reducing the problem of computing the number of perfect matchings in 3-regular bipartite graphs to the popular matchings problem, and (ii) the fact that k-regular bipartite graphs admit a perfect matching.

▶ **Theorem 14.** *Given an instance $G = (\mathcal{A} \cup \mathcal{P}, E)$ of the popular matchings problem with ties, counting the total number of popular matchings in G is #P-Complete.*

4 Cheating strategies – preliminaries

In this section we set up the notation useful in formulating our cheating strategies. We begin by partitioning the set of agents $\mathcal{A}$ depending on the posts that a particular agent gets matched to when each agent is truthful, that is, in the instance G.

$$\begin{aligned}
\mathcal{A}_f &= \{a : \text{every popular matching in } G \text{ matches } a \text{ to one of her rank-1 posts}\}\\
\mathcal{A}_s &= \{a : \text{every popular matching in } G \text{ matches } a \text{ to one of her non-rank-1 posts}\}\\
\mathcal{A}_{f/s} &= \mathcal{A} \setminus (\, \mathcal{A}_f \cup \, \mathcal{A}_s \,).
\end{aligned}$$

The set $\mathcal{A}_{f/s}$ denotes the set of agents a such that a gets matched to one of her rank-1 posts in some popular matching in G, whereas to one of her non-rank-1 posts in some other popular matching in G. It is easy to see that the above partition can be readily obtained once we have the set of popular pairs $PopPairs$ (defined by Equation (3)).

Let a_1 be the sole manipulative agent who is aware of the true preference lists of all other agents. Let $\mathcal{L} = P_1, P_2, \ldots, P_t, \ldots, P_l$ denote the true preference list of a_1 where P_i denotes the set of i-th rank posts of a_1. Since we will be working with another instance H obtained by falsifying the preference list of a_1, we now qualify the sets $f(a)$ and $s(a)$ for every agent with the instance under consideration. For an agent a, let $f_G(a)$ and $s_G(a)$ denote sets $f(a)$ and $s(a)$ respectively for an agent a in G. We note that $f_G(a_1) = P_1$. Assume that $s_G(a_1) \subseteq P_t$ is the set of t-th ranked posts of a_1, where $t > 1$.

Recall the strategy – *better always* defined for a single manipulative agent. If agent $a_1 \in \mathcal{A}_f$, then she does not have any incentive to manipulate her preference list. Thus, in this case we are done and $\mathcal{L}$ is her optimal strategy. We therefore focus on $a_1 \in \mathcal{A}_s \cup \mathcal{A}_{f/s}$. Let H denote the instance obtained by falsifying the preference list of a_1 alone.

- If $a_1 \in \mathcal{A}_s$, then in order to get *better always* her goal is to force at least some popular matching in H to match her to a post which she strictly prefers to her t-th ranked post.
- If $a_1 \in \mathcal{A}_{f/s}$, then in order to get *better always* her goal is to force every popular matching in H to match her to one of her true rank-1 posts.

Denote by $H \succ G$ with respect to a_1 if agent a_1 is *better always* in H. It is instructive to consider examples in order to get intuition regarding the cheating strategies.

▶ **Example 15.**

Consider the instance G as shown in Figure 1(a) and let a_5 be the manipulative agent. It can be seen that $a_5 \in \mathcal{A}_{f/s}$ in G. Now consider the instance H where a_5 alone falsifies her preference list. The preference list of a_5 in H is a strict list of length two and contains p_3 as her rank-1 post and p_8 as her rank-2 post. It is easy to verify that every popular matching in H matches a_5 to p_3 which is her true rank-1 post. The idea for an $\mathcal{A}_{f/s}$ agent a is to choose a post in $s_H(a)$ (here p_8) to which a can never be matched in a popular matching in H. We will show that such a post can be chosen whenever there exists a non-sink component in the switching graph and therefore a tight-pair (in this case $\mathcal{P}_1 = \{p_8, p_1\}$ and $\mathcal{A}_1 = \{a_2, a_3\}$).

▶ **Example 16.**

Consider the instance G shown in Figure 1(a) and let a_1 be the manipulative agent. Every popular matching in G matches a_1 to either p_6 or p_7 and therefore $a_1 \in \mathcal{A}_s$. Let H denote the instance where a_1 falsifies her preference list. The preference list of a_5 in H is a strict list of length two and contains p_3 as her rank-1 post and p_8 as her rank-2 post. It can be verified that every popular matching in H matches a_1 to p_3. The intuition here is that, a post to which a_1 wishes to get matched (here p_3), should have a path to an unmatched post or roughly speaking, belong to a sink component of G_M. We also choose a post in $s_H(a_1)$ (in this case p_8) to which a_1 can never get matched in any popular matching in H.

However, in this example, this is not the best that a_1 can get by falsifying. Let a_1 falsify her preference list to rank p_2 as her rank-1 post and p_8 as her rank-2 post. Consider the matching $M'' = \{(a_1, p_2), (a_2, p_1), (a_3, p_8), (a_4, p_3), (a_5, p_4), (a_6, p_9), (a_7, p_5)\}$ in H. It can be verified that M'' is popular in H and in fact every popular matching in H matches a_1 to p_2. However, our intuition that p_2 should belong to a sink component does not hold. This is because the edge (a_4, p_3) which got deleted in Step 4 of Algorithm 2.1 is being used after a_1 falsifies her preference list. In order to deal with such cases we will work with a modified instance as defined in Section 4.3.

We now formalize these intuitions in the rest of the section. In the interest of space we omit proof details and refer the reader to the full version [11].

4.1 $s(a)$ for other agents remains unchanged

Let H denote the instance obtained by falsifying the preference list of a_1 alone. Since the rest of the agents are truthful, for every agent $a \in \mathcal{A} \setminus \{a_1\}$, we have $f_H(a) = f_G(a)$. However, since $s_H(a)$ depends on the rank-1 posts of the rest of the agents, it may be the case that when a_1 falsifies her preference list, $s_H(a) \neq s_G(a)$ for an agent $a \in \mathcal{A} \setminus \{a_1\}$. We claim that if a_1 falsifies her preference list only to improve the rank of the post that she gets matched to, the rest of the agents do not change their $s(a)$. Recall that by definition, $s_H(a)$ is the set of most preferred posts of a which are *even* in the graph H_1 (the graph H on rank-1 edges). The following theorem summarizes the discussion.

▶ **Theorem 17.** *Let H be an instance such that $H \succ G$ w.r.t. a_1. Then, (i) $(\mathcal{E}_1)_G \cap \mathcal{P} = (\mathcal{E}_1)_H \cap \mathcal{P}$ and hence $s_H(a) = s_G(a)$ for every $a \in \mathcal{A} \setminus \{a_1\}$ and, (ii) $(\mathcal{O}_1)_G \cap \mathcal{A} = (\mathcal{O}_1)_H \cap \mathcal{A}$.*

4.2 An $\mathcal{A}_s$ agent cannot get one of her true rank-1 posts

In this section we show that if $a_1 \in \mathcal{A}_s$, then by falsifying her preference list alone, she cannot get matched to one of her true rank-1 posts in any popular matching in H. We state the result as Theorem 18 which requires the following claims. Let M be a popular matching in G and G_M denote the corresponding switching graph.

(I) If $a_1 \in \mathcal{A}_s$, then every post $q \in f_G(a_1)$ belongs to a non-sink component of G_M. We further claim that the edge $(M(a_1), q)$ does not belong to any cycle in G_M.

(II) Since every $q \in f_G(a_1)$ belongs to a non-sink component of G_M, using Lemma 12, we can define a tight-pair $\mathcal{P}_q$ and $\mathcal{A}_q$ w.r.t. q. Here, $\mathcal{P}_q$ denotes the set of posts reachable from q in G_M whereas $\mathcal{A}_q$ denotes the set of agents matched to the posts in $\mathcal{P}_q$. We claim that the post $M(a_1)$ does not belong to $\mathcal{P}_q$ and hence a_1 does not belong to $\mathcal{A}_q$.

(III) From the definition of tight-pair, we know that $|\mathcal{A}_q| = |\mathcal{P}_q|$ and for each $a \in \mathcal{A}_q$, $\mathit{choices}_G(a) \subseteq \mathcal{P}_q$. However, we claim that the same pair of sets is *tight* in H, that is, for every $a \in \mathcal{A}_q$, $\mathit{choices}_H(a) \subseteq \mathcal{P}_q$.

Using the above facts we prove the following theorem.

▶ **Theorem 18.** *Let $a_1 \in \mathcal{A}_s$. Then by falsifying her preference list alone, she cannot get matched to a post $q \in f_G(a_1)$ in any popular matching in the falsified instance.*

Proof. For contradiction assume that there exists a falsified instance H such that in a popular matching M' of H, agent a_1 gets matched to $q \in f_G(a_1)$. By (I), the post q belongs to a non-sink component of G_M. Further by (III), there exists a set of agents $\mathcal{A}_q$ and a set of posts $\mathcal{P}_q$ such that $|\mathcal{A}_q| = |\mathcal{P}_q|$, $a_1 \notin \mathcal{A}_q$ and for every $a \in \mathcal{A}_q$, we have $\mathit{choices}_H(a) \subseteq \mathcal{P}_q$. Thus, if a_1 gets matched to q in M', then there is at least one agent $a' \in \mathcal{A}_q$ which does not have a post to be matched to in $\mathit{choices}_H(a')$. This contradicts the fact that M' is a popular matching in H. ◀

4.3 The modified instance $\tilde{G}$

As mentioned earlier, we need to define a modified instance, call it $\tilde{G}$ to develop our cheating strategies. Recall from Example 16 that an edge which gets deleted from the graph G' in Step 4 of Algorithm 2.1 can be used in a popular matching in a falsified instance. Thus, we define $\tilde{G}$ from the instance G which has the following properties: (i) every popular matching in G is a popular matching in $\tilde{G}$ and, (ii) any edge (a, p) that gets deleted in Step 4 of Algorithm 2.1 when run on $\tilde{G}$ also gets deleted in Step 4 when Algorithm 2.1 is run on H such that $H \succ G$ w.r.t. a_1. However, the definition of $\tilde{G}$ is independent of the agent a_1.

The graph $\tilde{G}$ is defined as follows: Let G_1 be the graph on rank-1 edges of G and let $\{q_1, \ldots, q_k\}$ be the set of *unreachable* posts in G_1. Let us add to the instance G a dummy agent b whose preference list is of length one and has all the *unreachable* posts in G_1 tied as her rank-1 posts. That is, the preference list of b can be written as $(q_1, \ldots, q_k)$. The set of posts as well as the preference lists of all the agents $a \in \mathcal{A}$ remain the same as in G. Formally, $\tilde{G} = (\tilde{\mathcal{A}} \cup \mathcal{P}, \tilde{E})$ where $\tilde{\mathcal{A}} = \mathcal{A} \cup \{b\}$ and $\tilde{E} = E \cup \{(b, q_1), \ldots, (b, q_k)\}$ and each (b, q_i) is a rank-1 edge. By the choice of preference list of b, we note that $f_{\tilde{G}}(b) = \{q_1, \ldots, q_k\}$ and $s_{\tilde{G}}(b) = \ell(b)$, the unique last-resort post that we add for convenience.

We note that even after the addition of agent b, a maximum matching M_1 in G_1 continues to be a maximum matching in $\tilde{G}_1$. However, with respect to the partition of vertices on rank-1 edges in $\tilde{G}$, every vertex is either *odd* or *even* in $\tilde{G}_1$. Further, we claim that the set of *even* posts in $\tilde{G}_1$ is the same as the set of *even* posts in G_1. Thus, we can state the following lemma.

▶ **Lemma 19.** *For every $a \in \mathcal{A}$, we have $s_{\tilde{G}}(a) = s_G(a)$.*

Now let M be a popular matching in G, then let $\tilde{M}$ denote the corresponding matching in $\tilde{G}$ such that for every $a \in \mathcal{A}$ we have $\tilde{M}(a) = M(a)$ and $\tilde{M}(b) = \ell(b)$, the unique last-resort post of b. Note that $\tilde{M}$ is a maximum matching on rank-1 edges in $\tilde{G}$ and for every $a \in \mathcal{A}$, we have $\tilde{M}(a) \in \{f_{\tilde{G}}(a) \cup s_{\tilde{G}}(a)\}$. Finally, $\tilde{M}(b) \in s_{\tilde{G}}(b)$ since $s_{\tilde{G}}(b) = \{\ell(b)\}$. It is clear that $\tilde{M}$ satisfies both the properties of Theorem 3 and therefore is a popular matching in $\tilde{G}$. We can now construct the switching graph $\tilde{G}_{\tilde{M}}$ w.r.t. $\tilde{M}$ in $\tilde{G}$. With the definition of $\tilde{G}$ in place, we can now state the following lemmas.

▶ **Lemma 20.** *Let (a, p) be an edge which gets deleted in Step 4 of Algorithm 2.1 run on $\tilde{G}$. Then (a, p) gets deleted in Step 4 when Algorithm 2.1 is run on H where $H \succ G$ w.r.t. a_1.*

▶ **Lemma 21.** *Let $a \in \mathcal{A} \setminus \{a_1\}$ such that $\tilde{M}(a)$ belongs to a non-sink component of $\tilde{G}_{\tilde{M}}$. Let H be an instance such that $H \succ G$ w.r.t. a_1. Then $choices_H(a) \subseteq choices_{\tilde{G}}(a)$.*

5 Cheating strategies

In this section we develop an efficient characterization of the conditions under which a_1 can falsify her preference list. We formulate the strategy of a_1 depending on whether $a_1 \in \mathcal{A}_s$ or $a_1 \in \mathcal{A}_{f/s}$. Throughout, we assume that the true preference list of a_1 is denoted by $\mathcal{L} = P_1, \ldots, P_t, \ldots, P_l$ where P_i denotes the set of i-th ranked posts of a_1. Further, $f_G(a_1) = P_1$ and $s_G(a_1) \subseteq P_t$. We will use the modified instance $\tilde{G}$ to formulate our strategies.

5.1 $\mathcal{A}_s$ agent

Let $a_1 \in \mathcal{A}_s$ and let M be any popular matching in G and $\tilde{M}$ denote the corresponding popular matching in $\tilde{G}$ which matches b to $\ell(b)$. It follows from the definition of $\mathcal{A}_s$ that, $M(a_1) = \tilde{M}(a_1) \in s_G(a_1)$ and therefore, $M(a_1) \in P_t$. We first characterize whether a_1 can get *better always* using the graph $\tilde{G}$ and the switching graph $\tilde{G}_{\tilde{M}}$.

Our cheating strategy for a_1 (as shown in Figure 2) is simple: it checks if any of a_1's i-th ranked posts $p \in P_i$ where $i = 2 \ldots t-1$, either belongs to a sink component in $\tilde{G}_{\tilde{M}}$ or has a path to $\tilde{M}(a_1)$ in $\tilde{G}_{\tilde{M}}$. If there exists such a post p, our strategy ensures that every popular matching in the falsified instance H matches a_1 to p. We denote by $\mathcal{L}_f$ the falsified preference list of a_1. We now state the main theorem in this section.

1. For $i = 2 \ldots t-1$ check if there exists a post $p \in P_i$ in a_1's preference list such that
   (a) p belongs to a sink component in $\tilde{G}_{\tilde{M}}$ or,
   (b) p has a path to $\tilde{M}(a_1)$ in $\tilde{G}_{\tilde{M}}$.
2. If no post satisfies (a) or (b) above, then true preference list $\mathcal{L}$ is optimal for a_1.
3. Else let p denote the most preferred post of a_1 satisfying one of the above two properties. Set post p as a_1's rank-1 post in the falsified preference list.
4. To obtain the rank-2 post for a_1, let a_2 be some agent such that $\tilde{M}(a_2) \in f_G(a_1)$. Let $p' \in s_G(a_2)$. Set p' as the rank-2 post of a_1 in the falsified instance. $\mathcal{L}_f = p, p'$.

Figure 2 Cheating strategy for $a_1 \in \mathcal{A}_s$.

▶ **Theorem 22.** *Let $a \in \mathcal{A}_s$. Then there exists a cheating strategy for a_1 to get* better always *if and only if there exists a post p ranked $2 \ldots t-1$ on a_1's preference list satisfying either*

(a) p belongs to a sink component in $\tilde{G}_{\tilde{M}}$ or,
(b) p has a path to $\tilde{M}(a_1)$ in $\tilde{G}_{\tilde{M}}$.

Proof. (Sketch) Assume that a post p satisfying one of the above two properties exists. Let $\mathcal{L}_f = p, p'$ be the falsified preference list for a_1 as returned by Step 4 of Figure 2. Let H denote the instance where a_1 submits $\mathcal{L}_f$ and the rest of the agents are truthful. We show that every popular matching in H matches a_1 to p. The idea is to use the path starting at p which ends in an unmatched vertex and construct another matching which matches a_1 to p. Further, to show that every popular matching matches a_1 to p, we use the tight-sets $\mathcal{A}_{p'}$ and $\mathcal{P}_{p'}$. Finally, to show that our strategy is optimal, from Theorem 18, we know that a_1 cannot get matched to any of her true rank-1 posts. Now let q be a post which does not satisfy any of the conditions in Theorem 22 and is more preferred by a_1 than the post that it got matched to after running the strategy in Figure 2. We show the existence of tight-sets $\mathcal{A}_q$ and $\mathcal{P}_q$ for such a post q which implies that no popular matching in the falsified instance can match a_1 to q. Refer [11] for a full proof. ◀

5.2 $\mathcal{A}_{f/s}$ agent

Let $a_1 \in \mathcal{A}_{f/s}$ when she submits her true preference list. In order to get *better always*, the goal of a_1 is to falsify her preference list such that every popular matching in the falsified instance H matches a_1 to posts in P_1.

Let M be a popular matching in G such that $M(a_1) = p$ and $p \in f_G(a_1)$. Let $\tilde{M}$ denote the corresponding popular matching in $\tilde{G}$ which matches b to $\ell(b)$. Consider the switching graph $\tilde{G}_{\tilde{M}}$. Our strategy for a_1 to get better always (as described in Figure 3) is to search for an *even* post p' in G_1 which belongs to a non-sink component of $\tilde{G}_{\tilde{M}}$ and further the post p' does not have a path T to $\tilde{M}(a_1)$ in $\tilde{G}_{\tilde{M}}$ where $w(T) = +1$.

1. For every $p' \in (\mathcal{E}_1)_G \cap \mathcal{P}$ check if
   (a) p' belongs to a non-sink component, say $\mathcal{Y}_1$, of $\tilde{G}_{\tilde{M}}$ and,
   (b) p' does not have a path T to $\tilde{M}(a_1)$ in $\tilde{G}_{\tilde{M}}$ such that $w(T) = +1$.
2. If no post satisfies both properties, declare true preference list $\mathcal{L}$ is optimal for a_1.
3. Else set $M(a_1) = p$ and p' as the rank-1 and rank-2 posts respectively in the falsified preference list of a_1. $\mathcal{L}_f = p, p'$.

Figure 3 Cheating strategy for $a_1 \in \mathcal{A}_{f/s}$ to get better always.

▶ **Theorem 23.** *Let $a_1 \in \mathcal{A}_{f/s}$. There exists a cheating strategy for a_1 to get* better always *if and only if there exists a post p' in $(\mathcal{E}_1)_G$ satisfying the following two properties*
(a) p' belongs to a non-sink component, say $\mathcal{Y}_1$, of $\tilde{G}_{\tilde{M}}$, and
(b) there exists no path T from p' to $\tilde{M}(a_1)$ in $\tilde{G}_{\tilde{M}}$ such that $w(T) = +1$.

Proof. (Sketch) Assume that a post p' satisfying the above two properties exists. Then by falsifying her preference list as $\mathcal{L}_f = p, p'$, agent a_1 can force every popular matching in H to match a_1 to p. The proof uses the existence of the tight-pair $\mathcal{A}_{p'}, \mathcal{P}_{p'}$. On the other hand, assume that no such post exists and for the sake of contradiction, there exists an instance H such that every popular matching in H matches a_1 to a post in $f_G(a_1)$. In this case we show that there exists a popular matching M' in H such that $M'(a_1) \in s_H(a_1)$. Further, $s_H(a_1)$ cannot contain any of a_1's true rank-1 posts, therefore this contradicts the fact that every popular matching in H matches a_1 to one of her true rank-1 posts. ◀

Using Theorem 22 and Theorem 23 we conclude the following.

▶ **Theorem 24.** *The optimal falsified preference list for a single manipulative agent to get* better always *can be computed in $O(\sqrt{n}m)$ time if preference lists contain ties and in time $O(m+n)$ time if preference lists are all strict.*

Proof. The main steps of our strategy are (i) to compute the set of popular pairs, (ii) to construct the switching graph, (iii) run the algorithm given by Figure 2 or Figure 3 as appropriate for the single manipulative agent. We note that we use the modified graph $\tilde{G}$ for computing our strategies and let $\tilde{n}$ and $\tilde{m}$ denote the vertices and edges in $\tilde{G}$ respectively. Clearly, $\tilde{n} = n + 1$ and $\tilde{m} < m + n = O(m)$. Once the switching graph is constructed, we observe that the algorithms in Figure 2 and Figure 3 have checks which can be done in time which is linear in the size of the switching graph. Thus the steps (i) and (ii) defined above decide the complexity of our cheating strategy. In case of ties, we have shown that both the steps can be computed in $O(\sqrt{n}m)$ time. In case of strict lists, using the switching graph given by McDermid and Irving [10], both the steps can be computed in $O(m + n)$ time. Thus we have the desired result. ◀

Acknowledgment: The author is grateful to Prof. Vijaya Ramachandran for useful discussions on the problem and to an anonymous reviewer for the helpful comments.

References

1 D. J. Abraham, R. W. Irving, T. Kavitha, and K. Mehlhorn. Popular Matchings. *SIAM Journal on Computing*, 37(4):1030–1045, 2007.

2 D. Gale and L. Shapley. College Admissions and the Stability of Marriage. *American Mathematical Monthly*, 69:9–14, 1962.

3 P. Gärdenfors. Match making: Assignments based on bilateral preferences. *Behavioural Sciences*, 20:166–173, 1975.

4 C.-C. Huang. Cheating to Get Better Roommates in a Random Stable Matching. In *Proceedings of 24th Annual Symposium on Theoretical Aspects of Computer Science*, pages 453–464, 2007.

5 C.-C. Huang and T. Kavitha. Near-Popular Matchings in the Roommates Problem. In *Proceedings of the 19th Annual European Symposium on Algorithms*, pages 167–179, 2011.

6 T. Kavitha. Popularity vs maximum cardinality in the stable marriage setting. In *Proceedings of the 23rd ACM-SIAM Symposium on Discrete Algorithms*, pages 123–134, 2012.

7 T. Kavitha, J. Mestre, and M. Nasre. Popular Mixed Matchings. *Theoretical Computer Science*, 412(24):2679–2690, 2011.

8 M. Mahdian. Random Popular Matchings. In *Proceedings of 7th ACM Conference on Electronic Commerce*, pages 238–242, 2006.

9 R. M. McCutchen. The Least-Unpopularity-Factor and Least-Unpopularity-Margin Criteria for Matching Problems with One-Sided Preferences. In *Proceedings of the 15th Latin American Symposium on Theoretical Informatics*, pages 593–604, 2008.

10 E. McDermid and R. W. Irving. Popular matchings: structure and algorithms. *Journal of Combinatorial Optimization*, 22(3):339–358, 2011.

11 M. Nasre. Popular Matchings: Structure and Cheating Strategies. *CoRR*, abs/1301.0902, 2013.

12 W. R. Pulleyblank. *Handbook of Combinatorics (Vol. 1)*, chapter Matchings and Extensions, pages 179–232. MIT Press, Cambridge, MA, USA, 1995.

13 C.-P. Teo, J. Sethuraman, and W.-P. Tan. Gale-Shapley Stable Marriage Problem Revisited: Strategic Issues and Applications. *Management Science*, 47(9):1252–1267, 2001.

Fooling One-Sided Quantum Protocols*

Hartmut Klauck[1] and Ronald de Wolf[2]

1 **CQT and Nanyang Technological University**
Singapore
`hklauck@gmail.com`

2 **CWI and University of Amsterdam**
Amsterdam, The Netherlands
`rdewolf@cwi.nl`

Abstract

We use the venerable "fooling set" method to prove new lower bounds on the quantum communication complexity of various functions. Let $f : X \times Y \to \{0,1\}$ be a Boolean function, $\text{fool}^1(f)$ its maximal fooling set size among 1-inputs, $Q_1^*(f)$ its one-sided-error quantum communication complexity with prior entanglement, and $NQ(f)$ its nondeterministic quantum communication complexity (without prior entanglement; this model is trivial with shared randomness or entanglement). Our main results are the following, where logs are to base 2:

- If the maximal fooling set is "upper triangular" (which is for instance the case for the equality, disjointness, and greater-than functions), then we have $Q_1^*(f) \geq \frac{1}{2}\log \text{fool}^1(f) - \frac{1}{2}$, which (by superdense coding) is essentially optimal for functions like equality, disjointness, and greater-than. No super-constant lower bound for equality seems to follow from earlier techniques.
- For all f we have $Q_1^*(f) \geq \frac{1}{4}\log \text{fool}^1(f) - \frac{1}{2}$.
- $NQ(f) \geq \frac{1}{2}\log \text{fool}^1(f) + 1$. We do not know if the factor 1/2 is needed in this result, but it cannot be replaced by 1: we give an example where $NQ(f) \approx 0.613 \log \text{fool}^1(f)$.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Quantum computing, communication complexity, fooling set, lower bound

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.424

1 Introduction

1.1 Background: fooling classical communication protocols

Communication complexity [20, 11] is one of the most versatile and successful computational models we have, and *lower bounds* on communication complexity are one of the main sources of lower bounds in many other areas, from circuits to data structures to data streams. One of the simplest and most intuitive ways to prove lower bounds on communication protocols is by exhibiting a large *fooling set*, which was first done in [20, 15]. Suppose Alice and Bob want to compute some function $f : X \times Y \to \{0,1\}$, given inputs $x \in X$ and $y \in Y$, respectively. A 1-fooling set for f is a set $F = \{(x,y)\}$ of input pairs with the following properties:

* HK's research at the Centre for Quantum Technologies is funded by the Singapore Ministry of Education and the National Research Foundation. RdW is partially supported by a Vidi grant from the Netherlands Organization for Scientific Research (NWO) and by the European Commission under the project QCS (Grant No. 255961). Part of this work was done when RdW was visiting CQT, whose hospitality is gratefully acknowledged.

(1) If $(x, y) \in F$ then $f(x, y) = 1$
(2) If $(x, y), (x', y')$ are distinct pairs in F then $f(x, y') = 0$ or $f(x', y) = 0$

Note that these two conditions imply that if pairs $(x, y), (x', y') \in F$ are distinct (i.e., differ in at least one coordinate), then they differ in *both* coordinates. Hence a fooling set F forms a bijection between $|F|$ inputs on Alice's side and $|F|$ inputs on Bob's side. Accordingly, by renaming some of Bob's inputs we can always assume without loss of generality that F is of the form $\{(x, x)\}$.

To illustrate the concept of a fooling set, consider the n-bit equality function EQ, defined on $x, y \in \{0, 1\}^n$ as $\text{EQ}(x, y) = 1$ iff $x = y$. This has a 1-fooling set $F = \{(x, x)\}$ of size 2^n, since $\text{EQ}(x, x) = 1$ for all x and $\text{EQ}(x, y) = 0$ for all distinct x, y. The same fooling set also works for the n-bit greater-than function, which is defined as $\text{GT}(x, y) = 1$ iff $y \geq x$. The n-bit disjointness function $\textsc{Disj}$, defined as $\textsc{Disj}(x, y) = 1$ iff $|x \wedge y| = 0$, also has a 1-fooling set of size 2^n, which can be seen as follows: write its communication matrix as $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{\otimes n}$, and take the anti-diagonal as the 1-fooling set. All entries on the anti-diagonal are 1 (giving the first property) and all entries below the anti-diagonal are 0 (giving the second property).

Now consider for simplicity a deterministic protocol computing f. Suppose the last bit of the conversation is the output bit, so both parties end up knowing the output. Consider input pairs $(x, y), (x', y') \in F$. For both inputs, the first property of the fooling set says that the correct output value is 1. Suppose, by way of contradiction, that the conversation between Alice and Bob is the same on both input pairs. If we switch input pair (x, y) to (x, y') then nothing changes from Alice's perspective (neither her input nor the conversation changes), so the output will still be 1. Similarly, if we switch (x, y) to (x', y) then the output won't change from Bob's perspective. But by the second property of fooling sets, for at least one of (x, y') and (x', y), the correct output is 0! Hence the conversations on inputs (x, y) and (x', y') must have been different. Accordingly, the bigger our fooling set F is, the more distinct conversations we must allow and hence the more bits of communication are needed.

More precisely, the communication complexity is lower bounded by $\log |F| + 1$. A formal proof of this fact can be based on the notion of *monochromatic rectangles*. A rectangle is a set $R = A \times B$, where $A \subseteq X$ and $B \subseteq Y$. Such a rectangle is *1-monochromatic* if $f(x, y) = 1$ for all $(x, y) \in R$. Note that a rectangle containing 1-inputs $(x, y), (x', y') \in F$ cannot be 1-monochromatic, because by the rectangle property it also contains (x, y') and (x', y), at least one of which is a 0-input by fooling set property 2. Accordingly, if we want to include F in a set of 1-rectangles, we need a separate 1-rectangle for each element of F, so we need at least $|F|$ different rectangles. It is well-known that a deterministic c-bit communication protocol induces a partition of the set of all 1-inputs into 2^{c-1} 1-monochromatic rectangles, so the previous argument implies $2^{c-1} \geq |F|$; equivalently $c \geq \log |F| + 1$. In fact even *nondeterministic* communication complexity is lower bounded by $\log |F| + 1$: a c-bit nondeterministic protocol gives rise to a *cover* (rather than partition) of the set of all 1-inputs by 2^{c-1} 1-monochromatic rectangles, and we still need a separate rectangle for each element of F.

In contrast, a *quantum* communication protocol does not naturally induce a partition or cover of the 1-inputs into rectangles[1], so the above way of reasoning fails. In fact, in contrast to the classical case, the number of monochromatic rectangles needed to partition the 1-inputs does not provide a lower bound on exact quantum protocols, as witnessed by

[1] It can be viewed as approximately producing rectangles *with signs* [10, Section 3].

the exponential separation in [4]. Nevertheless, in this paper we show how fooling sets can still be used to lower bound quantum communication complexity. We do this in two settings: one-sided-error quantum protocols with unlimited prior entanglement and nondeterministic quantum protocols without entanglement. These results also imply lower bound for quantum "Las Vegas" or "zero-error" protocols (i.e., quantum protocols that never err, but have probability $\leq 1/2$ of giving up without a result).

1.2 Our results: fooling one-sided-error quantum protocols

First, we study one-sided-error protocols: protocols that always output 0 on inputs x, y where $f(x, y) = 0$, and that output 1 with probability at least 1/2 on inputs where $f(x, y) = 1$. We start by getting an essentially optimal bound for the case of "upper-triangular" fooling sets. We call a 1-fooling set $F = \{(x, x)\}$ upper-triangular if there is some total ordering '$\geq$' on the x's such that $x > y$ implies $f(x, y) = 0$. In other words, the matrix M with entries $M_{xy} = f(x, y)$ is 0 below the diagonal. In Section 2 we show that if f has an upper-triangular 1-fooling set of size N, then

$$Q_1^*(f) \geq \frac{1}{2}\log N - \frac{1}{2}.$$

For example, the n-bit equality, disjointness, and greater-than functions all have upper-triangular 1-fooling sets of size 2^n, and hence an $n/2-1/2$ lower bound on their one-sided-error complexity $Q_1^*(f)$. We have $Q_1^*(f) \leq n/2 + 1$ for any Boolean function where $X \subseteq \{0, 1\}^n$, because superdense coding [2] allows Alice to send 2 classical bits using one EPR-pair and one qubit of communication. Hence the above result is essentially tight for the functions mentioned.[2]

We can extend this to a slightly weaker result for all functions stated in terms of their (not necessarily upper-triangular) 1-fooling-set size:

$$Q_1^*(f) \geq \frac{1}{4}\log \mathrm{fool}^1(f) - \frac{1}{2}.$$

Surprisingly for such basic functions as equality and disjointness, these bounds were not known before. While it is possible to use Razborov's technique [16] combined with results about polynomial approximation with very small error [5] to show $Q_1^*(\textsc{Disj}) = \Omega(n)$, no super-constant lower bound was known for $Q_1^*(\mathrm{EQ})$. This gap in our knowledge was due to the fact that other existing lower bound methods cannot give good lower bounds for equality, as we explain now. General lower bound methods for quantum communication complexity can be grouped into rank-based methods and methods based on approximation norms (in particular based on the γ_2-norm [14]).[3] The linearity of norms makes it possible to prove lower bounds for quantum protocols in which Alice and Bob share prior entanglement. Rank-based methods, however, do not seem to directly apply to protocols with entanglement: in the case of exact quantum protocols a direct sum-based construction in [6] shows that the

[2] While the fooling set method gives very good bounds for these functions, it does not give good bounds for *all* functions. For example, a random function will with high probability have linear quantum communication complexity (which can be shown for instance using the discrepancy method), but only small fooling sets. Inner product mod 2 is an example of an explicit function with this property [11, Example 4.16].

[3] Information-theoretic methods [8] have also been used to lower bound quantum communication complexity. However, the notion is defined there for internal information cost, and in this case the information cost for equality is $O(1)$, even for classical protocols without error [3, Proposition 3.21].

logarithm of the rank is a lower bound even in the presence of entanglement.[4] In the case of two-sided error and entanglement, Lee and Shraibman [12] show that the approximation rank yields lower bounds by relating it to the γ_2-norm. Since the communication matrix of EQ is the identity matrix I, and $\gamma_2(I) = O(1)$ for I of any size, there is no hope to use a connection between a one-sided-error version of approximation rank and the γ_2-norm to establish a large lower bound on $Q_1^*(\mathrm{EQ})$. Whether a one-sided-error version of approximation rank gives lower bounds for Q_1^* remains open, but we note that the construction in [12] cannot be adapted to the one-sided-error scenario.

So neither of the two main approaches to quantum communication complexity lower bounds provides us with a good lower bound for $Q_1^*(\mathrm{EQ})$. Hence in this paper we take a different approach. We first simulate a quantum protocol with entanglement by a game without communication, in which Alice and Bob share entanglement, and they need to compute a function f conditioned on postselection on their local measurements. This approach itself is not new, and can for instance be used to show that the γ_2-norm is a lower bound, see [13]. We then analyze the impact of Alice and Bob's measurements on the single entangled state used in the game. The one-sided-error requirement places strong constraints on those measurements, which we exploit to derive our lower bound in terms of fooling sets.

In a quantum *Las Vegas* protocol Alice and Bob compute a function f without error, but they are allowed to give up without a result with probability 1/2. The quantum Las Vegas communication complexity with entanglement $Q_0^*(f)$ is the minimum worst-case communication of any protocol that computes f under these requirements.[5] Quantum Las Vegas protocols were investigated in [5, 9, 19] in the case where no prior entanglement is available. Since $Q_0^*(f) \geq \max\{Q_1^*(f), Q_1^*(\neg f)\}$ we immediately get large lower bounds on the quantum Las Vegas complexity of DISJ and EQ, and also the following general lower bound:

$$Q_0^*(f) \geq \frac{1}{4}\log \mathrm{fool}(f) - \frac{1}{2},$$

where $\mathrm{fool}(f)$ is the standard maximum fooling set size, i.e., the maximum over the largest 1-fooling set and 0-fooling set.

1.3 Our results: fooling nondeterministic quantum protocols

As a second main result, just like in the classical world fooling sets lower bound *nondeterministic* protocols, we show here that they also lower bound nondeterministic *quantum* protocols. For our purposes, we can define a nondeterministic protocol (quantum as well as classical) for a Boolean function f as one that has positive acceptance probability on input x, y iff $f(x, y) = 1$. In other words, this is the unbounded-error version of the one-sided-error model: the requirement of acceptance probability 0 on 0-inputs remains, but the requirement of *large* acceptance probability on 1-inputs is relaxed to *positive* acceptance probability on 1-inputs.[6] The quantum version of this model was introduced in [19], which also exhibits a

[4] Footnote 2 of [6] claims such a bound for *zero-error* quantum protocols for equality and disjointness without proof, but in retrospect they didn't seem to have a proof of this.

[5] It is possible to define Las Vegas protocols as protocols that never err and place bounds on *expected* communication. The corresponding complexity measure is always larger or equal to the one considered here, and is smaller than 2 times our measure.

[6] Nondeterministic communication complexity (classical as well as quantum) can be exponentially less than one-sided-error communication complexity, even if the latter is assisted by unlimited prior entanglement. The negation of the disjointness function is an example of this.

total function with an exponential separation between quantum and classical nondeterministic communication complexities.

Note that allowing unlimited prior entanglement trivializes the nondeterministic model, for the same reason that unlimited shared randomness trivializes it in the classical case: Alice and Bob can share a random variable r uniformly distributed over the set X of Alice's inputs; Alice sends a bit indicating whether $x = r$; if 'yes' then Bob outputs $f(r, y) = f(x, y)$, and if 'no' then he outputs 0. Hence if we were to allow unlimited prior randomness or entanglement, any function has nondeterministic communication complexity at most 1. Accordingly, we will study nondeterministic protocols which don't share anything at the start. In Section 3 we show the following lower bound on nondeterministic quantum communication complexity in terms of fooling sets:

$$NQ(f) \geq \frac{1}{2} \log \mathrm{fool}^1(f) + 1.$$

We do not know if the factor 1/2 is needed in this result, but it cannot be replaced by 1: in Section 3 we give an example of a function where $NQ(f) \leq \frac{\log 3}{\log 6} \log \mathrm{fool}^1(f) + 1$, where $\log 3/ \log 6 \approx 0.613$.

2 Lower bound for one-sided bounded-error quantum protocols

We assume familiarity with communication complexity. See [11] for more details about classical communication complexity and [18] for quantum communication complexity. Our key lemma is based on a reasonably well-known trick to replace quantum communication by the guessing of twice as many classical bits:

▶ **Lemma 1.** *Suppose there is a quantum protocol P with inputs from $X \times Y$ and output in $\{0, 1\}$, that uses some fixed starting state (possibly entangled) and q qubits of communication, and where a measurement of the last qubit on the channel gives the output. Then there exists another quantum protocol Q with a fixed starting state and no communication at all, where Alice outputs $a \in \{0, 1\}$ and Bob outputs $b \in \{0, 1\}$, such that*

$$\textit{for all inputs } x, y : \Pr[Q \textit{ outputs } a = b = 1] = 2^{-2q} \Pr[P \textit{ outputs } 1].$$

Proof. We assume without loss of generality that P communicates *exactly* q qubits on all possible inputs. By the well-known teleportation primitive [1], we can replace each qubit of communication in P by the use of one additional EPR-pair and two classical bits of communication. These 2 bits are the outcome of a measurement by the sending party, and indicate which of the 4 Pauli matrices the receiving party has to apply on their end of the EPR-pair in order to obtain the qubit that the sender wanted to send. If the bits happen to be 00 (which happens with probability 1/4), then the right Pauli is the identity matrix, so then they don't need to do anything. Call the resulting $2q$-bit protocol P_{clas}.

Protocol Q is now as follows. Alice and Bob run protocol P_{clas} *assuming* all messages are 0-bits (so they don't communicate anything). Alice checks if all her teleportation measurements gave outcome 00. If not then she outputs $a = 0$; if yes then she outputs P_{clas}'s output if she was the one supposed to output that, and otherwise she outputs $a = 1$. Bob does the same from his end, outputting $b \in \{0, 1\}$. Note that $a = b = 1$ iff all q teleportation measurements gave outcome 00 *and* the output of P would have been 1. The first event happens with probability 4^{-q} and the second event with $\Pr[P \text{ outputs } 1]$. Since these two events are independent we can multiply their probabilities to obtain the lemma. ◀

Note that the starting state of the new protocol Q is the starting state of the original protocol P, augmented with an additional q EPR-pairs. Using the above lemma, we can prove an essentially optimal lower bound in terms of upper-triangular 1-fooling sets:

▶ **Theorem 2.** *If $f : X \times Y \to \{0,1\}$ has an upper-triangular 1-fooling set of size N, then $Q_1^*(f) \geq \frac{1}{2}\log N - \frac{1}{2}$.*

Proof. We can assume without loss of generality that the fooling set is of the form $\{(x,x) : x \in [N]\}$, and $f(x,y) = 0$ whenever $x > y$. Let $q = Q_1^*(f)$ and let P be a q-qubit entanglement-assisted protocol for f. Apply Lemma 1 to this protocol to obtain a new protocol Q without communication, where Alice outputs $a \in \{0,1\}$, Bob outputs $b \in \{0,1\}$, satisfying

$\Pr[a = b = 1] \geq 2^{-2q-1}$ on inputs x, x
$\Pr[a = b = 1] = 0$ on inputs $x > y$

Let $|\psi\rangle$ be the entangled starting state of protocol Q, which we assume to be pure without loss of generality. On input x, Alice applies a POVM measurement with operators $A_x, I - A_x$ corresponding to outputs 1 and 0, respectively. Similarly Bob uses POVM elements $B_y, I - B_y$. The following technical claim is the core of the proof:

▶ Claim 1. Let $|w\rangle$ be a bipartite state such that for all $x, y \in [N]$ satisfying $x > y$, we have $\langle w|A_x \otimes B_y|w\rangle = 0$. Then $\sum_{x\in[N]}\langle w|A_x \otimes B_x|w\rangle \leq \|w\|^2$.

Proof. The proof is by induction on N. The base case $N = 1$ follows from the Cauchy-Schwarz inequality and the fact that $A_x \otimes B_x$ has operator norm ≤ 1.

For the inductive step: assume the claim holds for N, and now let x range over $[N+1]$. Fix some bipartite state $|w\rangle$ such that

(*) for all $x, y \in [N+1]$ satisfying $x > y$, we have $\langle w|A_x \otimes B_y|w\rangle = 0$.

Let $\mathrm{supp}(A_{N+1})$ denote the projection on the support of POVM element A_{N+1}. Define $|w_1\rangle = (\mathrm{supp}(A_{N+1}) \otimes I)|w\rangle$, and $|w_2\rangle = |w\rangle - |w_1\rangle$. By (*), for all $y \in [N]$ we have $\langle w|A_{N+1} \otimes B_y|w\rangle = 0$. This means that $|w\rangle$ is orthogonal to all eigenvectors $|a\rangle \otimes |b\rangle$ of $A_{N+1} \otimes B_y$, which in turn implies

(**) for all $y \in [N]$, $(\mathrm{supp}(A_{N+1}) \otimes B_y)|w\rangle$ is the 0-vector.

Write

$$\sum_{x\in[N+1]} \langle w|A_x \otimes B_x|w\rangle = \langle w|A_{N+1} \otimes B_{N+1}|w\rangle + \sum_{x\in[N]} \langle w|A_x \otimes B_x|w\rangle. \tag{1}$$

Since $(A_{N+1} \otimes I)|w_2\rangle = 0$, the first term on the right-hand side equals $\langle w_1|A_{N+1} \otimes B_{N+1}|w_1\rangle$, which is $\leq \|w_1\|^2$ by the base case.

For the second term, note that for all (not necessarily distinct) $x, y \in [N]$, we have

$$A_x \otimes B_y|w_1\rangle = (A_x \otimes B_y)(\mathrm{supp}(A_{N+1}) \otimes I)|w\rangle = (A_x \otimes I)(\mathrm{supp}(A_{N+1}) \otimes B_y)|w\rangle,$$

which is 0 because $(\mathrm{supp}(A_{N+1}) \otimes B_y)|w\rangle = 0$ by (**). Thus we have $A_x \otimes B_y|w\rangle = A_x \otimes B_y|w_2\rangle$, which by (*) also implies that for all $x, y \in [N]$ with $x > y$ we have $\langle w_2|A_x \otimes B_y|w_2\rangle = 0$. Now the second term on the right-hand side of (1) equals

$$\sum_{x\in[N]} \langle w_2|A_x \otimes B_x|w_2\rangle,$$

which is $\leq \|w_2\|^2$ by the induction hypothesis. Since $|w_1\rangle$ and $|w_2\rangle$ are orthogonal, the two terms on the right-hand side of (1) together are at most $\|w_1\|^2 + \|w_2\|^2 = \|w\|^2$. This concludes the inductive step, and hence the proof of the claim. ◀

Applying Claim 1 with the actual entangled state $|\psi\rangle$ used by protocol Q, we obtain

$$\begin{aligned} N2^{-2q-1} &\leq \sum_{x\in[N]} \Pr[\text{outcome } A_x \otimes B_x \text{ when measuring } |\psi\rangle] \\ &= \sum_{x\in[N]} \langle\psi|A_x \otimes B_x|\psi\rangle \leq \|\psi\|^2 = 1. \end{aligned}$$

Rearranging gives the theorem. ◀

▶ **Corollary 3.** *The n-bit equality, disjointness and greater-than functions have $Q_1^*(f) \geq n/2 - 1/2$.*

Proof. These three functions all have upper-triangular 1-fooling sets of size 2^n. ◀

Now we use a trick of combining two copies of the function to extend the result from upper-triangular fooling sets to all fooling sets, at the expense of a factor of 2 in the lower bound (we do not know if this loss is necessary). This is similar to the proof that fooling set size is at most quadratically bigger than rank [11, Lemma 4.15]:

▶ **Corollary 4.** *For all $f : X \times Y \to \{0,1\}$ we have $Q_1^*(f) \geq \frac{1}{4}\log \mathrm{fool}^1(f) - \frac{1}{2}$.*

Proof. Define a new function $g : X^2 \times Y^2 \to \{0,1\}$ by $g(xx', yy') = f(x,y)f(y',x')$. Note the reversed role of the two inputs in the second f. Alice and Bob can compute g with one-sided error $p = 1/4$ by separately computing $f(x,y)$ and $f^T(x',y') = f(y',x')$ with one-sided error 1/2 each, and outputting the product of the two output bits. This takes $Q_1^*(f)$ qubits of communication for each computation, so at most $2Q_1^*(f)$ in total.

Let $\{(x,x)\}$ be a 1-fooling set for f of size $N = \mathrm{fool}^1(f)$. Then it is easy to see that $\{(xx,xx)\}$ is a 1-fooling set for g, with the additional property that $g(xx,yy) = f(x,y)f(y,x) = 0$ whenever $x \neq y$. Hence the communication matrix for g contains the $N \times N$ identity as a submatrix (i.e., the equality function). The same proof as above gives a lower bound of $\frac{1}{2}\log N - 1$ for one-sided-error protocols for equality that accept 1-inputs with probability at least 1/4 (instead of at least 1/2 as above). Hence we have

$$\frac{1}{2}\log N - 1 \leq 2Q_1^*(f),$$

which implies the statement. ◀

3 Lower bound for nondeterministic quantum protocols

In this section we study nondeterministic quantum protocols. The following algebraic characterization of nondeterministic quantum communication complexity of f is known. The *communication matrix* M_f for f is the $|X| \times |Y|$ Boolean matrix $M_f(x,y) = f(x,y)$. A *nondeterministic matrix* for f is any real or complex matrix M with the same support as M_f, i.e., such that $M_{x,y} = 0$ iff $f(x,y) = 0$. The *nondeterministic rank* of f (abbreviated to $\mathrm{nrank}(f)$) of f is the minimal rank (over the reals) among all such matrices. [19, Theorem 3.3] shows that $NQ(f) = \lceil \log \mathrm{nrank}(f)\rceil + 1$.

The key to using fooling sets for nondeterministic quantum lower bounds is the following simple lemma:

▶ **Lemma 5.** *For every function $f : X \times Y \to \{0,1\}$ we have* $\mathrm{nrank}(f)^2 \geq \mathrm{fool}^1(f)$.

Proof. Let $N = \mathrm{fool}^1(f)$. Like in the proof of Corollary 4, define $g(xx', yy') = f(x,y)\cdot f(y',x')$ and observe that the communication matrix of g contains the $N \times N$ identity matrix I_N as a submatrix. If M is a nondeterministic matrix for f, then $M \otimes M^T$ is a nondeterministic matrix for g. Hence, choosing M of minimal rank, we have

$$\mathrm{nrank}(f)^2 = \mathrm{rank}(M)^2 = \mathrm{rank}(M \otimes M^T) \geq \mathrm{nrank}(g) \geq \mathrm{nrank}(I_N) = N.$$

◀

Taking logarithms and using that $NQ(f) = \lceil \log \mathrm{nrank}(f) \rceil + 1$, we get

▶ **Corollary 6.** $NQ(f) \geq \frac{1}{2} \log \mathrm{fool}^1(f) + 1$.

For example for the equality function, this shows $NQ(f) \geq n/2 + 1$. However, for the equality function we already knew $NQ(f) = n+1$ since obviously $\mathrm{nrank}(f) = 2^n$ [19]. Hence it is natural to ask whether the constant $1/2$ in the above corollary is needed. We don't know, but at least we can show that it needs to be less than 1. Specifically, we give an example where $NQ(f) \leq \frac{\log 3}{\log 6} \log \mathrm{fool}^1(f) + 1$, where $\frac{\log 3}{\log 6} \approx 0.613$. Consider the following 6×6 matrix:

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 & -1 & 0 \\ -1 & 1 & 1 & 0 & -1 & 0 \\ -1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}.$$

It is easy to see that this has rank 3. The Boolean matrix obtained by dropping the minus signs corresponds to a communication complexity function $g : [6] \times [6] \to \{0,1\}$ with a 1-fooling set of size 6 (just take the diagonal). Now let $f : X \times Y \to \{0,1\}$ be the AND of k independent instances of g (so $|X| = |Y| = 6^k$). Because 1-fooling set size is multiplicative under taking ANDs, we have $\mathrm{fool}^1(f) = 6^k$. On the other hand, taking the k-fold tensor product of the above rank-3 matrix gives a nondeterministic matrix for f of rank 3^k. Hence $NQ(f) = \lceil \log \mathrm{nrank}(f) \rceil + 1 \leq \frac{\log 3}{\log 6} \log \mathrm{fool}^1(f) + 1 \approx 0.613 \log \mathrm{fool}^1(f)$.

A simpler but slightly weaker separation can be obtained from the 3-input non-equality function, where $X = Y = [3]$ and the function take value 0 when the inputs x and y are equal. This has nrank $= 2$ vs $\mathrm{fool}^1 = 3$, hence taking a k-fold AND of this gives a function $f : X \times Y \to \{0,1\}$ with $|X| = |Y| = 3^k$ and $\mathrm{nrank}(f) = 2^k$ vs $\mathrm{fool}^1(f) = 3^k$. Taking logarithms, we have $NQ(f) \approx 0.63 \log \mathrm{fool}^1(f)$.

4 Conclusion and open problems

Equality and disjointness are two of the most important functions considered in communication complexity. Prior to this paper no large lower bound on the one-sided error or Las Vegas quantum communication complexity of these functions was known for the case of protocols with prior entanglement. In particular, for EQ previous lower bound methods were not applicable. We have shown that the fooling set method is applicable to one-sided-error protocols with entanglement, obtaining linear lower bounds for both functions.

It is interesting to note that for classical protocols there is essentially no need to consider fooling sets at all: the method is completely subsumed by the rectangle bound (i.e., bounding the size of the largest monochromatic rectangle under some distribution). However, the rectangle bound does not apply to quantum protocols with one-sided error and entanglement, nor to quantum nondeterministic communication complexity.

We conclude with some open problems:

- Can we improve the factor 1/4 in Corollary 4? We believe it should be 1/2, which is what we already showed here for upper-triangular 1-fooling sets.
- Another problem is to show that the factor 1/2 in Corollary 6 is necessary. It seems hard to come up with a matrix for which the nondeterministic rank is the square root of the rank, as would be required by a construction along the lines of our separation at the end of Section 3.
- One further goal would be to show that classical deterministic complexity $D(f)$ and quantum Las Vegas complexity $Q_0(f)$ are polynomially close for all total functions. This is a (possibly easier) variant of a general conjecture that for total functions quantum communication yields only polynomial improvements in communication complexity. Proving a linear lower bound in terms of classical nondeterministic complexity (i.e., $Q_0(f) = \Omega(N(f))$) would settle that, since it is known that $D(f) = O(N(f)^2)$. However, an example from [19] refutes that hope. Let $f(x, y) = 0$ if $|x \wedge y| = 1$ and $f(x, y) = 1$ otherwise. This function as well as its complement have linear $N(f)$, but $NQ(f), NQ(\neg f) = O(\sqrt{n})$. This does not, however, preclude a bound like $Q_0(f) = \Omega(\sqrt{N(f)})$, which would still achieve the above goal.

Acknowledgements

We thank Harry Buhrman and Matthias Christandl (as well as an anonymous referee) for pointing out an error in an earlier version of this paper, which we corrected here.

References

1 C. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. Wootters. Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Physical Review Letters*, 70:1895–1899, 1993.

2 C. Bennett and S. Wiesner. Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states. *Physical Review Letters*, 69:2881–2884, 1992.

3 M. Braverman. Interactive information complexity. In *Proceedings of 44th ACM STOC*, pages 505–524, 2012. Also ECCC report No. 123 (2011).

4 H. Buhrman, R. Cleve, and A. Wigderson. Quantum vs. classical communication and computation. In *Proceedings of 30th ACM STOC*, pages 63–68, 1998. quant-ph/9802040.

5 H. Buhrman, R. Cleve, R. de Wolf, and Ch. Zalka. Bounds for small-error and zero-error quantum algorithms. In *Proceedings of 40th IEEE FOCS*, pages 358–368, 1999. cs.CC/9904019.

6 H. Buhrman and R. de Wolf. Communication complexity lower bounds by polynomials. In *Proceedings of 16th IEEE Conference on Computational Complexity*, pages 120–130, 2001. cs.CC/9910010.

7 P. Høyer and R. de Wolf. Improved quantum communication complexity bounds for disjointness and equality. In *Proceedings of 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS'2002)*, volume 2285 of *Lecture Notes in Computer Science*, pages 299–310. Springer, 2002. quant-ph/0109068.

8 R. Jain, J. Radhakrishnan, and P. Sen. A lower bound for the bounded round quantum communication complexity of set disjointness. In *Proceedings of 44th IEEE FOCS*, pages 220–229, 2003.

9 H. Klauck. On quantum and probabilistic communication: Las Vegas and one-way protocols. In *Proceedings of 32nd ACM STOC*, pages 644–651, 2000.

10 H. Klauck. Lower bounds for quantum communication complexity. *SIAM Journal on Computing*, 37(1):20–46, 2007. Earlier version in FOCS'01. quant-ph/0106160.

11 E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, 1997.

12 T. Lee and A. Shraibman. An approximation algorithm for approximation rank. In *Proceedings of 24th IEEE Conference on Computational Complexity*, pages 351–357, 2009.

13 T. Lee and A. Shraibman. Lower bounds in communication complexity. *Foundations and Trends in Theoretical Computer Science*, 3(4):263–398, 2009.

14 N. Linial and A. Shraibman. Lower bounds in communication complexity based on factorization norms. *Random Struct. Algorithms*, 34(3):368–394, 2009. Earlier version in STOC'07.

15 R. J. Lipton and R. Sedgewick. Lower bounds for VLSI. In *Proceedings of 13th ACM STOC*, pages 300–307, 1981.

16 A. Razborov. Quantum communication complexity of symmetric predicates. *Izvestiya of the Russian Academy of Sciences, mathematics*, 67(1):159–176, 2003. quant-ph/0204025.

17 R. de Wolf. Characterization of non-deterministic quantum query and quantum communication complexity. In *Proceedings of 15th IEEE Conference on Computational Complexity*, pages 271–278, 2000. cs.CC/0001014.

18 R. de Wolf. Quantum communication and complexity. *Theoretical Computer Science*, 287(1):337–353, 2002.

19 R. de Wolf. Nondeterministic quantum query and quantum communication complexities. *SIAM Journal on Computing*, 32(3):681–699, 2003. Journal version of parts of [17] and [7].

20 A. C-C. Yao. Some complexity questions related to distributive computing. In *Proceedings of 11th ACM STOC*, pages 209–213, 1979.

Explicit relation between all lower bound techniques for quantum query complexity*

Loïck Magnin[1,3] and Jérémie Roland[2,3]

1 Centre for Quantum Technologies, National University of Singapore
Block S15, 3 Science Drive 2, Singapore 117543
loick@locc.la

2 QuIC, Ecole Polytechnique de Bruxelles, Université Libre de Bruxelles
50 av. F.D. Roosevelt - CP165/59, B-1050 Bruxelles, Belgium
jroland@ulb.ac.be

3 NEC Laboratories America
4 Independence Way, Suite 200, Princeton NJ 08540, USA

Abstract

The polynomial method and the adversary method are the two main techniques to prove lower bounds on quantum query complexity, and they have so far been considered as unrelated approaches. Here, we show an explicit reduction from the polynomial method to the multiplicative adversary method. The proof goes by extending the polynomial method from Boolean functions to quantum state generation problems. In the process, the bound is even strengthened. We then show that this extended polynomial method is a special case of the multiplicative adversary method with an adversary matrix that is independent of the function. This new result therefore provides insight on the reason why in some cases the adversary method is stronger than the polynomial method. It also reveals a clear picture of the relation between the different lower bound techniques, as it implies that all known techniques reduce to the multiplicative adversary method.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Quantum computation, lower bound, adversary method, polynomial method

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.434

1 Introduction

Polynomial and adversary methods. There are two main techniques to prove lower bounds on quantum query complexity: the polynomial method [12, 20, 27], based on bounding the degree of the function seen as a polynomial, and adversary methods [15, 2, 11, 21, 19], based on bounding the change in a progress function from one query to the next. In its original form [2], the adversary method bounds the additive change in the progress function, hence we will call it *additive*, and the progress function is based on a matrix assigning positive weights to pairs of inputs. The polynomial method and this original adversary method are not comparable. Indeed, the original adversary method is limited by the "certificate complexity barrier" [30, 29], that is, for total functions, $\mathrm{ADV}(f) \leq \sqrt{C_0(f)C_1(f)}$ where $C_b(f)$ denotes

* This work was supported by ARO/NSA under grant W911NF-09-1-0569. L.M. also acknowledges the support of the Ministry of Education and the National Research Foundation, Singapore. J.R. also acknowledges support from the action *Mandats de Retour* of the *Politique Scientifique Fédérale Belge* and the Belgian ARC project COPHYMA.

the certificate complexity of f for $f(x) = b$. It means that the original adversary method cannot prove lower bounds better than $\Omega(N^{1/2})$ for Element Distinctness. However, Aaronson and Shi [1] were able to prove a $\Omega(N^{2/3})$ lower bound using the polynomial method. On the other hand it is known that the adversary method can sometimes give better lower bounds than the polynomial method, in [4] Ambainis exhibits a function with polynomial degree d and adversary bound $\Omega(d^{1.3})$.

Høyer, Lee and Špalek have extended the additive adversary method by allowing negative weights in the matrix [18], and have shown that the corresponding bound, $\mathrm{ADV}^{\pm}(f)$, breaks the certificate complexity barrier. For simplicity, we will from now on refer to $\mathrm{ADV}^{\pm}(f)$ as the additive adversary bound, implicitly allowing negative weights.

Recently, a series of works [17, 7, 26, 25, 22] culminated in showing that this bound is tight in the bounded-error case for any function. However, this fundamental result does not answer all the questions about quantum query complexity as it suffers from two limitations. First, in some cases it is necessary to prove bounds for very small success probabilities, a regime where $\mathrm{ADV}^{\pm}(f)$ might not be tight. For this reason, while the optimality of the additive adversary bound implies that quantum query complexity satisfies a direct sum theorem, it cannot be used to prove a strong direct product theorem, which requires to prove nontrivial bounds for exponentially small success probabilities. Secondly, while the proof of optimality of $\mathrm{ADV}^{\pm}(f)$ implies that if a lower bound on the bounded-error quantum query complexity of a function can be proved with any method, it can also be proved with $\mathrm{ADV}^{\pm}(f)$, this reduction is not constructive. Concretely, there are still examples of lower bounds that can be proved using the polynomial method for which the optimal adversary matrix is unknown, a typical example being the Collision problem [1][1].

Multiplicative adversary method. The first limitation has been overcome thanks to the introduction of another adversary-type method. By formalizing an ad-hoc technique proposed by Ambainis, de Wolf and Špalek [5, 10], Špalek designed a new lower bound method which he called the multiplicative adversary method [28], as the idea is to bound the multiplicative change in the progress function for each query. Ambainis *et al.* [9] later showed that the multiplicative bound is always at least as strong as the additive one, and therefore also characterizes bounded-error quantum query complexity. Moreover, the multiplicative adversary method can prove better lower bounds for small success probability than the additive adversary method, and this was used to prove a strong direct product theorem for quantum query complexity [23].

Quantum state generation. Even when we are only interested in the quantum query complexity of functions, it is useful to also consider state generation problems: in that case, instead of producing the output $f(x)$ on input x, the algorithm is required to prepare a quantum state $|m_x\rangle$. Since unitary transformations independent of x may be applied without any query to x, a quantum state generation problem is completely defined by the Gram matrix $M = \sum_{x,x'} \langle m_{x'}|m_x\rangle |x\rangle\langle x'|$. In the special case of computing a function, M is a Boolean matrix. Thus every algorithm can be seen as generating a Gram matrix M. If the algorithm is allowed some error ε, then the set of Gram matrices that are acceptable outputs for the algorithm can be bounded by a so-called output condition. Different output conditions have been used before, for example, the original adversary method [2] was implicitly using a condition based on the L_∞ norm, while the adversary method with negative weights in [18]

[1] Until very recently it was also the case for the Element Distinctness problem, whose lower bound was proved by reduction to Collision, but a direct adversary lower bound has now been shown by Belovs [13], and later extended to the k−Sum problem by Belovs and Špalek [14].

$$\begin{tikzcd}
 & \mathrm{MADV}_\varepsilon(f) \arrow[dl, dotted, "\text{③}"] & \\
\mathrm{ADV}^\pm_\varepsilon(f) \arrow[ur, "\text{②}"] & & \mathrm{xpoly}_\varepsilon(f) \arrow[ul, "\text{④}"'] \\
\mathrm{ADV}_\varepsilon(f) \arrow[u, "\text{①}"] \arrow[r, phantom, "\overset{\text{⑤}}{\gtrless}"] & & \widetilde{\deg}_\varepsilon(f) \arrow[u, "\text{④}"'] \arrow[ull, dotted, "\text{③}"]
\end{tikzcd}$$

Figure 1 Relations between the different methods to prove lower bounds for quantum query complexity. An arrow from method A to method B implies that any lower bound that can be proved with A can also be proved with B (i.e., B is stronger than A). A solid yellow arrow means that the reduction is constructive, i.e., we can obtain a witness for B from a witness for A. ① [18] ② [9] ③ [25, 22] ④ [This article] ⑤ The original additive and the polynomial methods are incomparable [30, 29, 1, 4]

was implicitly using the factorization norm γ_2. Realizing that different output conditions could be combined with different (zero-error) lower bound methods was key to comparing the additive and multiplicative adversary methods in [9]. More recently, Lee and Roland [23] were able to characterize exactly the set of acceptable Gram matrices, hence providing an optimal output condition (see Claim 4), which allowed them to prove a strong direct product theorem for quantum query complexity. This also simplifies the study of lower bounds techniques as it implies that the bounded-error quantum query complexity of a problem can be studied by bounding the zero-error quantum query complexity of all Gram matrices that define valid output states for the problem. As a consequence it is sufficient to compare the zero-error bounds for two methods in order to compare them.

Our results. In this article, we tackle the second limitation by giving an explicit reduction from the polynomial method to the multiplicative adversary method. In order to do so, we introduce yet another lower bound technique for quantum query complexity, which we call the extended polynomial method (Definition 10 and Theorem 11) as it can be seen as an extension of the polynomial method to Gram matrices. As the degree of a Boolean function can be stated as the maximum index of its Fourier coefficients, that is, $\deg(f) = \max\{|S| : \langle\chi_S, f\rangle \neq 0\}$, we define the degree of a Gram matrix by the maximum index k such that the Gram matrix has support on a Fourier vector $|\chi_S\rangle$ with $|S| = k$, that is, $\deg(M) = \max\{|S| : \langle\chi_S|M|\chi_S\rangle \neq 0\}$.

For Boolean functions, the polynomial and the extended polynomial bounds are equal in the zero-error case. However, for the approximate case, the extended polynomial method uses the tight output condition, and is therefore possibly stronger than the polynomial method (Theorem 13).

We also compare the extended polynomial method to the multiplicative adversary method. More particularly, we show that in the limit $c \to \infty$, where c is the maximum multiplicative change in the progress function for one query, the multiplicative bound tends to the extended polynomial method (Theorem 14). This proof is constructive, i.e., we give an explicit multiplicative adversary matrix for which we have the equality. It might come as a surprise that this matrix does not depend on the problem: it is the same adversary matrix for every function. Let us note that it was proved in [9] that the multiplicative bound is stronger than the additive bound in the limit $c \to 1$, that is, at the other end of the possible range for c. This new result therefore completes the picture of the relations between the different lower bound techniques in quantum query complexity (see Figure 1), and shows in particular that all these methods reduce to the multiplicative adversary method.

Many proofs are omitted from this extended abstract and can be found in the full version of the paper.

2 Preliminaries

2.1 Gram matrices and fidelity

▶ **Definition 1.** A **density matrix** ρ is a positive semidefinite matrix $\rho \succeq 0$ such that $\mathrm{tr}(\rho) = 1$. A **normalized Gram matrix** A is a positive semidefinite matrix $A \succeq 0$ such that $A \circ \mathbb{I} = \mathbb{I}$, where $\circ$ denotes the Hadamard (entry-wise) product.

Note that any positive semidefinite matrix A can be written as a Gram matrix in the broader sense, i.e., there always exists a set of vectors $\{|a_x\rangle\}$ such that $A_{xy} = \langle a_x|a_y\rangle$. Here, the additional constraint $A \circ \mathbb{I} = \mathbb{I}$ means that we require those vectors to have norm 1. Since all Gram matrices will be normalized in what follows, we will from now on refer to normalized Gram matrices as simply *Gram matrices*.

▶ **Definition 2.** The **fidelity** $\mathcal{F}(\rho, \sigma)$ between two density matrices ρ and σ is defined by $\mathcal{F}(\rho, \sigma) = \mathrm{tr}\sqrt{\sqrt{\rho}\,\sigma\sqrt{\rho}}$.

The **Hadamard product fidelity** $\mathcal{F}_H(A, B)$ between two Gram matrices A and B is defined by $\mathcal{F}_H(A, B) = \min_{|u\rangle : \||u\rangle\|=1} \mathcal{F}(A \circ |u\rangle\langle u|, B \circ |u\rangle\langle u|)$.

The notation $\mathcal{F}_H$ and the name Hadamard product fidelity[2] are new to this article, but this quantity has been proved to be the tight output condition for the quantum query complexity in [23] (see Claim 4 below).

2.2 Quantum query complexity

Consider a Boolean function $f : \{0, 1\}^n \to \{0, 1\}$. In the *black-box model*, we are interested in computing $f(x)$ when x is given by an oracle $O_x : |i, b\rangle \mapsto (-1)^{b \cdot x_i}|i, b\rangle$. We denote by $Q_\varepsilon(f)$ the quantum query complexity of f, i.e., the minimum number of queries to O_x necessary for any algorithm to output $f(x)$ with error at most ε (see, e.g., [16]). Note that our choice of oracle computes the bits of x in the phase. Another variant of this model considers an oracle that computes the bits in a register, but it can be shown that these models are equivalent.

Even when we are only interested in the quantum query complexity of functions, it is useful to also consider state generation problems [9, 22]. In that case, instead of producing the output $f(x)$ on input x, the algorithm is required to prepare a quantum state $|m_x\rangle \in \mathcal{H}$. Since unitary transformations independent of x may be applied without any query to x, a quantum state generation problem is completely defined by the Gram matrix $M = \sum_{x,x'} \langle m_{x'}|m_x\rangle |x\rangle\langle x'|$. For a quantum state generation problem specified by a Gram matrix M, we define two

[2] The name is chosen by analogy to the Hadamard product trace norm γ_2 (equivalent to the Hadamard product operator norm and also called factorization norm), which for Hermitian matrices can be written in the very similar form $\gamma_2(A) = \max_{|u\rangle : \||u\rangle\| \le 1} \|A \circ |u\rangle\langle u|\|_{\mathrm{tr}}$.

different notions of query complexity. The coherent query complexity $Q_\varepsilon(M)$ is the minimum number of queries to the register oracle O_x necessary to generate a state $|n_x\rangle \in \mathcal{H} \otimes \mathcal{H}'$ such that $\Re(\langle n_x|(|m_x\rangle \otimes |\bar{0}\rangle)) \geq \sqrt{1-\varepsilon}$, where $\mathcal{H}'$ is the workspace of the algorithm, $|\bar{0}\rangle \in \mathcal{H}'$ is a default state for this workspace and $\Re(z)$ denotes the real part of a complex number z. The non-coherent query complexity $Q_\varepsilon^{\mathrm{nc}}(M)$ is defined similarly, except that it is enough to prepare a state $|n_x\rangle \in \mathcal{H} \otimes \mathcal{H}'$ such that $\Re(\langle n_x|(|m_x\rangle \otimes |m'_x\rangle)) \geq \sqrt{1-\varepsilon}$, for an arbitrary set of states $|m'_x\rangle \in \mathcal{H}'$ (that is, the workspace does not have to be reset to its default state).

For a Boolean function f, let us define the $\{1,-1\}$-valued function $\varphi : \{0,1\}^n \to \{1,-1\} : x \mapsto (-1)^{f(x)}$. There are two natural quantum state generation problems associated to f, corresponding to the Gram matrices $F = \sum_{x,x'} \delta_{f(x),f(x')} |x\rangle\langle x'|$ and $\Phi = \sum_{x,x'} \varphi(x)\varphi(x')|x'\rangle\langle x|$, where δ is the Kronecker delta. Indeed, generating the Gram matrix F non-coherently is exactly the same problem as computing f, and we therefore have $Q_\varepsilon(f) = Q_\varepsilon^{\mathrm{nc}}(F)$, while generating the Gram matrix Φ coherently corresponds to *computing the function in the phase*, i.e., we need to generate the state $\varphi(x)|\bar{0}\rangle$. The bounded-error complexities of these problems are closely related:

▶ **Claim 3** ([23]). $Q_{(1-\sqrt{1-\varepsilon})/2+\varepsilon/4}(f) \leq Q_\varepsilon(\Phi) \leq 2Q_{(1-\sqrt{1-\varepsilon})/2}(f)$.

This implies that to prove bounds on the bounded-error query complexity of f, it is sufficient to prove bounds on the query complexity of the related quantum state generation problem Φ, and this is precisely the approach that we will use in this article.

Another advantage of considering quantum state generation problems is that we can study the bounded-error query complexity of a problem by bounding the zero-error query complexity of all Gram matrices that define valid output states for the problem. It follows from the following claim that this set of valid Gram matrices is characterized by the Hadamard product fidelity:

▶ **Claim 4** ([23]). *For any Gram matrix M and any $\varepsilon \geq 0$, we have*

$$Q_\varepsilon(M) = \min_N \{Q_0(N) : \mathcal{F}_H(N,M) \geq \sqrt{1-\varepsilon},\ N \succeq 0,\ N \circ \mathbb{I} = \mathbb{I}\}.$$

2.3 The polynomial method

▶ **Definition 5.** For any $\varepsilon \geq 0$, the **approximate degree** $\widetilde{\deg}_\varepsilon(f)$ of a function $f : \{0,1\}^n \to \mathbb{R}$ is defined as $\widetilde{\deg}_\varepsilon(f) = \min_p \{\deg(p) : \forall x \in \{0,1\}^n,\ |p(x) - f(x)| \leq \varepsilon\}$, where the minimum is over n-variate polynomials $p : \mathbb{R}^n \to \mathbb{R}$.

▶ **Theorem 6** (Polynomial method [12]). *If f is a Boolean function, then $Q_\varepsilon(f) \geq \Omega\Big(\widetilde{\deg}_\varepsilon(f)\Big)$.*

In this article, we will use some basic Fourier analysis to relate degree of a function with Gram matrices. For the sake of readability, we will identify a set $S \subseteq \{1,\ldots,n\}$ with its characteristic vector $S \in \{0,1\}^n$: $S_i = 1$ if and only if $i \in S$, and thus $|S|$ can be either the cardinal of the set S or the Hamming weight of the vector S.

▶ **Definition 7.** For any $S \in \{0,1\}^n$, let us define $|\chi_S\rangle = \frac{1}{\sqrt{2^n}} \sum_x (-1)^{S.x}|x\rangle$. For a function $\varphi : \{0,1\}^n \to \mathbb{R}$, define the (non-normalized) state $|\varphi\rangle = \frac{1}{\sqrt{2^n}} \sum_x \varphi(x)|x\rangle$. We define the S-th **Fourier coefficient** of φ as $\hat{\varphi}(S) = \langle \chi_S|\varphi\rangle$.

Let us note that the set $\{|\chi_S\rangle\}_S$ is an orthonormal basis and that by definition, we then have $\hat{\varphi}(S) = \frac{1}{2^n}\sum_x (-1)^{S.x}\varphi(x)$ and $\varphi(x) = \sum_S (-1)^{S.x}\hat{\varphi}(S)$, which are the usual Fourier transform over the hypercube and its inverse. With these notations, we can also write the degree of a function φ as $\deg(\varphi) = \max_S\{|S| : \hat{\varphi}(S) \neq 0\}$.

2.4 The multiplicative adversary method

Let us consider a quantum algorithm generating the Gram matrix M with error at most ε using T queries. Let $|\psi_x^t\rangle$ be the state of the algorithm right after the t-th query when the input is x, and $M^t = \sum_{x,x'} \langle\psi_{x'}^t|\psi_x^t\rangle |x\rangle\langle x'|$ be the corresponding Gram matrix. Note that $M^0 = \mathbb{J}$ and $M^T \approx M$ (more precisely $\mathcal{F}_H(M^T, M) \geq \sqrt{1-\varepsilon}$). The basic idea of all adversary methods is to design a Hermitian matrix W defining a progress function $W[M] = \mathrm{tr}[WM]$ such that the initial value $W[\mathbb{J}]$ is low and the final value $W[M^T]$ is high (or vice versa), and then to bound the maximal change in the progress function for any oracle call. Whereas the additive method bounds the difference $|W[M^{t+1}] - W[M^t]|$, the multiplicative method bounds the ratio $W[M^{t+1}]/W[M^t]$. In this paper we use the definition of the multiplicative adversary method given by [23] which is a slight extension of the original multiplicative adversary method in [28].

▶ **Definition 8.** Let M be a Gram matrix specifying a quantum state generation problem and for all $i \in \{1, \cdots, n\}$, $D_i = \sum_{x,x'} (-1)^{x_i + x'_i} |x\rangle\langle x'|$ the action of the phase oracle on input i. Fix $c > 1$. The **multiplicative adversary bounds** are:

$$\mathrm{MADV}_0^c(M) = \frac{1}{\log c} \max_{W \succeq 0} \{\log \mathrm{tr}[WM] : \mathrm{tr}[W\mathbb{J}] = 1,\ W \circ D_i \preceq cW\ \forall i\},$$

$$\mathrm{MADV}_\varepsilon^c(M) = \min_N \left\{\mathrm{MADV}_0^c(N) : \mathcal{F}_H(N, M) \geq \sqrt{1-\varepsilon},\ N \succeq 0,\ N \circ \mathbb{I} = \mathbb{I}\right\},$$

$$\mathrm{MADV}_\varepsilon(M) = \sup_{c>1} \mathrm{MADV}_\varepsilon^c(M).$$

We call **adversary matrix** for $\mathrm{MADV}_0^c(M)$ any matrix $W \succeq 0$ such that $\mathrm{tr}[W\mathbb{J}] = 1$ and $W \circ D_i \preceq cW$ for all i.

▶ Remark. Let us note that the parameter c represents the maximum multiplicative change in the progress function that can result from one query. Since, for any matrix $W \succ 0$, the constraint $W \circ D_i \preceq cW$ is always satisfied for $c \geq \left\|(W \circ D_i)^{1/2} W^{-1/2}\right\|^2$, one could directly obtain the multiplicative bound MADV_0 by optimizing over W and taking $c = \left\|(W \circ D_i)^{1/2} W^{-1/2}\right\|^2$. However, it is useful to define the bound MADV_0^c for fixed c as this can be expressed as a semidefinite program (see [23]), where the objective value is optimized over W. The best bound on the quantum query complexity is then obtained by maximizing the objective value over both W and c.

▶ **Theorem 9** (Multiplicative adversary [28, 23])**.** *For any $\varepsilon \geq 0$ and any Gram matrix M, we have $Q_\varepsilon(M) \geq \mathrm{MADV}_\varepsilon(M)$.*

3 The extended polynomial method

We now extend the polynomial method from Boolean functions to Gram matrices.

▶ **Definition 10.** Let M be a Gram matrix specifying a quantum state generation problem. The **extended polynomial bounds** are

$$\mathrm{xpoly}_0(M) = \max_S \{|S| : \mathrm{tr}\left[|\chi_S\rangle\langle\chi_S| M\right] \neq 0\},$$

$$\mathrm{xpoly}_\varepsilon(M) = \min_N \left\{\mathrm{xpoly}_0(N) : \mathcal{F}_H(N, M) \geq \sqrt{1-\varepsilon},\ N \succeq 0,\ N \circ \mathbb{I} = \mathbb{I}\right\}.$$

▶ **Theorem 11.** *For any $\varepsilon \geq 0$ and any Gram matrix M, we have $Q_\varepsilon(M) \geq \mathrm{xpoly}_\varepsilon(M)$.*

Proof. We prove the statement for $\varepsilon = 0$ and the general case immediately follows from Claim 4 and the definition of $\mathrm{xpoly}_\varepsilon(M)$. This proof actually considers the extended polynomial method as an adversary method. Let us define the progress function

$$W[M^t] = \max_S \left\{|S| : \mathrm{tr}[|\chi_S\rangle\langle\chi_S| M^t] \neq 0\right\}.$$

Since $M^0 = \mathbb{J} = 2^n|\chi_\emptyset\rangle\langle\chi_\emptyset|$, its initial value is $W[M^0] = 0$. The final value is $W[M^T] = \mathrm{xpoly}_0(M)$. It suffices to show that one query increases the progress function by at most one.

Let $M^t = \sum_i M_i^t$ be the Gram matrix just before the $(t+1)$-th query, where M_i^t is the reduced Gram matrix corresponding to the part of the state where bit x_i is queried (see, e.g., [9] for details). Let $k = W[M^t]$ and note that by positivity, we have $\mathrm{tr}[|\chi_S\rangle\langle\chi_S|M^t] = 0$ if and only if $\mathrm{tr}[|\chi_S\rangle\langle\chi_S|M_i^t] = 0$ for all i. Therefore, we also have $W[M_i^t] \leq k$ for any i.

After the query, the Gram matrix of the algorithm will be $M^{t+1} = \sum_i M_i^t \circ D_i$. Let us observe that for any matrix A, we have $A \circ D_i = U_i A U_i^\dagger$ where $U_i = U_i^\dagger$ is the unitary matrix $U_i = \sum_x (-1)^{x_i}|x\rangle\langle x|$. In particular, $|\chi_S\rangle\langle\chi_S| \circ D_i = |\chi_{S'}\rangle\langle\chi_{S'}|$ where $S' = S \cup \{i\}$ if $i \notin S$ and $S' = S \setminus \{i\}$ if $i \in S$.

For all $S \in \{0,1\}^n$, we get:

$$\mathrm{tr}\left[|\chi_S\rangle\langle\chi_S|(M_i^t \circ D_i)\right] = \mathrm{tr}\left[(|\chi_S\rangle\langle\chi_S| \circ D_i)M_i^t\right] = \sum_{T:|T|\leq k} \mathrm{tr}\left[(|\chi_S\rangle\langle\chi_S| \circ D_i)|\chi_T\rangle\langle\chi_T|M_i^t\right].$$

This quantity is null for all S such that $|S| > k+1$, therefore the progress function can increase by at most one per query. ◀

We have defined the extended polynomial method with the Fourier basis, but one might wonder if choosing another basis could provide better bounds. It turns out that this is not the case.

▶ **Claim 12.** *Let $\{\Pi_k : 0 \leq k \leq K\}$ be a set of orthogonal projectors such that*

1. $\sum_k \Pi_k = \mathbb{I}_{\mathbb{C}^{2^n}}$,
2. $\mathrm{tr}(\Pi_0 \mathbb{J}) = 2^n$,
3. $\forall i \in \{1,\dots,n\}$, $\forall l,k$ *such that* $|l-k| > 1$, $\mathrm{tr}[(\Pi_l \circ D_i)\Pi_k] = 0$.

Then, for any Gram matrix M, we have $Q_0(M) \geq \mathrm{xpoly}_0(M) \geq \max_k \{k : \mathrm{tr}(\Pi_k M) \neq 0\}$.

Therefore, while any set of projectors provides a lower bound on quantum query complexity, the best bound is achieved by the extended polynomial method, which corresponds to the special case $K = n$ and $\Pi_k = \sum_{S:|S|=k} |\chi_S\rangle\langle\chi_S|$.

4 Relation between the polynomial and the extended polynomial methods

In this section, we compare the strength of the polynomial and the extended polynomial methods. Let f be a Boolean function and Φ the Gram matrix corresponding to computing f in the phase. By definition of the extended polynomial method, we have that $\mathrm{xpoly}_0(\Phi) = \deg(f)$. However the equality is lost in the approximate case:

▶ **Theorem 13.** *Let f be a Boolean function and Φ be the Gram matrix corresponding to computing f in the phase. For any $\varepsilon \geq 0$, we have* $\mathrm{xpoly}_\varepsilon(\Phi) \geq \widetilde{\deg}_{\varepsilon/2}(f)$.

Proof (sketch). Let N be a Gram matrix achieving the minimum in the definition of $\mathrm{xpoly}_\varepsilon(\Phi)$, that is, an optimal final Gram matrix of an algorithm for Φ. We first express this Gram matrix as $N = \sum_{x,y} \langle \psi_x | \psi_y \rangle |y\rangle\langle x|$, where $|\psi_x\rangle = \sum_i p_i(x)|i\rangle$ is the final state of the algorithm on input x expressed in the computational basis. By definition of the extended polynomial bound, we then have $\mathrm{xpoly}_\varepsilon(\Phi) = \max_i(\deg(p_i))$, where the maximum is over polynomials satisfying the normalization constraint $\sum_i p_i(x)^2 = 1$ and the correctness constraint $(-1)^{f(x)}\Re(p_0(x)) \geq \sqrt{1-\varepsilon}$, for all inputs x. The polynomial p_0 then witnesses that $\widetilde{\deg}_{\varepsilon/2}(f) \leq \deg(p_0) \leq \mathrm{xpoly}_0(N)$. ◀

5 Relation with the multiplicative adversary method

In [9], it was shown that in the limit $c \to 1$, the multiplicative adversary bound $\mathrm{MADV}_0^c(M)$ is at least as strong as the additive adversary bound $\mathrm{ADV}^\pm(M)$. Here, we show that the extended polynomial bound can be obtained by taking the limit $c \to \infty$.

▶ **Theorem 14.** *Let M be a Gram matrix, $\varepsilon \geq 0$, $T = \mathrm{xpoly}_\varepsilon(M)$ and $\Pi_{\geq T} = \sum_{S:|S|\geq T} |\chi_S\rangle\langle\chi_S|$. Moreover, let $\delta > 0$ be such that $\mathrm{tr}[\Pi_{\geq T} N] \geq \delta$ for any Gram matrix N such that $\mathcal{F}_H(N, M) \geq \sqrt{1-\varepsilon}$. Then, for any $c > 1$, we have*

$$\mathrm{xpoly}_\varepsilon(M) - \frac{n - \log\delta}{\log c} \leq \mathrm{MADV}_\varepsilon^c(M) \leq \mathrm{xpoly}_\varepsilon(M) + \frac{n}{\log c}.$$

In particular, in the limit $c \to \infty$, we have $\lim_{c\to\infty} \mathrm{MADV}_\varepsilon^c(M) = \mathrm{xpoly}_\varepsilon(M)$.

▶ Remark. Note that such a value of δ always exists. Assume by contradiction that $\mathrm{tr}[\Pi_{\geq T} N] = 0$, then $\mathrm{xpoly}_0(N) \leq T - 1$, however N is an ε-approximation of M that has a polynomial bound of T.

The general idea of the proof is to consider the multiplicative adversary matrix

$$W = \frac{1}{2^n} \sum_S c^{|S|} |\chi_S\rangle\langle\chi_S|$$

as a multiplicative adversary matrix. The lower bound then follows from the fact that in the limit $c \to \infty$, the value of the progress function $W[M] = \mathrm{tr}[WM]$ will be dominated by the term in $c^{|S|}$ for the set S with the largest size $|S| = k$ such that $\langle\chi_S|M|\chi_S\rangle \neq 0$, which therefore corresponds to the degree of the matrix M. As for the upper bound, we show that the matrix W becomes an optimal multiplicative adversary matrix in the limit $c \to \infty$. This can be shown by observing that one oracle call can only map a Fourier basis state $|\chi_S\rangle$ to another Fourier basis state $|\chi_{S'}\rangle$ with $|S'| = |S| \pm 1$ which implies bounds on the elements of any possible multiplicative adversary matrix written in the Fourier basis.

Proof. We prove it for the zero-error case, the general case follows immediately.

Consider the matrix $W = \frac{1}{2^n}\sum_S c^{|S|}|\chi_S\rangle\langle\chi_S|$. It is a valid adversary matrix for $\mathrm{MADV}_0^c(M)$ since $\mathrm{tr}[W\mathbb{J}] = 1$ and $W \circ D_i \preceq cW$, $\forall i \in \{1, \dots, n\}$. This inequality follows from $W \circ D_i = \frac{1}{2^n}\left(\sum_{S:i\in S} c^{|S|-1}|\chi_S\rangle\langle\chi_S| + \sum_{S:i\notin S} c^{|S|+1}|\chi_S\rangle\langle\chi_S|\right)$, see proof of Theorem 11. Let W' be an optimal multiplicative adversary matrix for $\mathrm{MADV}_0^c(M)$. Let us show that $\mathrm{tr}[WM] \leq \mathrm{tr}[W'M] \leq 2^n \,\mathrm{tr}[WM]$.

The first inequality is a direct consequence of the fact that W is an adversary matrix for $\mathrm{MADV}_0^c(M)$ and the definition of the multiplicative adversary bound.

To prove the second inequality, let us first show by induction on $k = |S|$ that $\langle\chi_S|W'|\chi_S\rangle \leq \frac{1}{2^n}c^{|S|}$ for any set S. For $k = 0$, the condition $\mathrm{tr}[W'\mathbb{J}] = 1$ is equivalent to $\langle\chi_\emptyset|W'|\chi_\emptyset\rangle = \frac{1}{2^n}$.

Let us fix $0 \leq k \leq n$, and assume that $\forall S$ such that $|S| = k$, we have $\langle \chi_S | W' | \chi_S \rangle \leq \frac{1}{2^n} c^k$. Let S' be a set of size $k+1$ and decompose it into $S' = S \cup \{i\}$. Observe first that $\langle \chi_S | W' \circ D_i | \chi_S \rangle = \langle \chi_S | U_i W' U_i | \chi_S \rangle = \langle \chi_{S'} | W' | \chi_{S'} \rangle$ where $U_i = \sum_x (-1)^{x_i} |x\rangle\langle x|$ as defined in the proof of Theorem 11. Hence by sandwiching $W' \circ D_i \preceq cW'$ with $|\chi_S\rangle$, we get $\langle \chi_{S'} | W' | \chi_{S'} \rangle \leq c \langle \chi_S | W' | \chi_S \rangle \leq \frac{1}{2^n} c^{|S|+1}$.

We can now proceed with the rest of the proof:

$$\begin{aligned}\operatorname{tr}[W'M] &= \sum_S \langle \chi_S | W'M | \chi_S \rangle = \sum_{S,S'} \langle \chi_S | W' | \chi_{S'} \rangle \langle \chi_{S'} | M | \chi_S \rangle \\ &\leq \sum_{S,S'} |\langle \chi_S | W' | \chi_{S'} \rangle| \, |\langle \chi_{S'} | M | \chi_S \rangle| \,.\end{aligned}$$

We now use the property that for any positive semidefinite matrix A, $|A_{ij}| \leq \sqrt{A_{ii} A_{jj}}$,

$$\operatorname{tr}[W'M] \leq \left(\sum_S \sqrt{\langle \chi_S | W' | \chi_S \rangle \langle \chi_S | M | \chi_S \rangle} \right)^2 .$$

Using the Cauchy-Schwarz inequality, we get:

$$\operatorname{tr}[W'M] \leq 2^n \sum_S \langle \chi_S | W' | \chi_S \rangle \langle \chi_S | M | \chi_S \rangle \leq \sum_S c^{|S|} \langle \chi_S | M | \chi_S \rangle = 2^n \operatorname{tr}[WM].$$

We are now ready to conclude the proof. From $\operatorname{tr}[WM] \leq \operatorname{tr}[W'M] \leq 2^n \operatorname{tr}[WM]$, we have by definition of $\mathrm{MADV}_0^c(M)$

$$\frac{\log \operatorname{tr}[WM]}{\log c} \leq \mathrm{MADV}_0^c(M) \leq \frac{n + \log \operatorname{tr}[WM]}{\log c}.$$

For $T = \mathrm{xpoly}_\varepsilon(M)$, we find from the first inequality

$$\mathrm{MADV}_0^c(M) \geq \frac{\log \frac{1}{2^n} c^T \operatorname{tr}[\Pi_{\geq T} M]}{\log c} = T + \frac{\log(\operatorname{tr}[\Pi_{\geq T} M]) - n}{\log c}.$$

Similarly, from the second inequality, we have

$$\mathrm{MADV}_0^c(M) \leq \frac{\log \sum_S c^{|S|} \langle \chi_S | M | \chi_S \rangle}{\log c} \leq T + \frac{\log \sum_S \langle \chi_S | M | \chi_S \rangle}{\log c} = T + \frac{n}{\log c},$$

where we used the facts that $\langle \chi_S | M | \chi_S \rangle = 0$ whenever $|S| > T$, and $\sum_S \langle \chi_S | M | \chi_S \rangle = \operatorname{tr}[M] = 2^n$. ◀

We note that $\mathrm{MADV}_\varepsilon^c(M)$ approaches its limiting value $\mathrm{xpoly}_\varepsilon(M)$ if c is large enough compared to $2^n/\delta$. In general, we cannot give a lower bound on δ in order to determine how large c should be. However, for the special case of Boolean functions, and comparing to the standard polynomial method, i.e., the approximate degree $\widetilde{\deg}_\varepsilon(f)$, instead of $\mathrm{xpoly}_\varepsilon(M)$, we can show that $\mathrm{MADV}_\varepsilon^c(\Phi)$ becomes at least as strong as $\widetilde{\deg}_\varepsilon(f)$ as soon as c is large compared to $2^n/\varepsilon$.

▶ **Lemma 15.** *Let f be a Boolean function with associated phase matrix Φ. Then, for any $c > 1$, we have* $\mathrm{MADV}_\varepsilon^c(\Phi) \geq \widetilde{\deg}_\varepsilon(f) - 2 \cdot \frac{n - \log \varepsilon}{\log c}$.

Proof. Just as in the proof of Theorem 13, we express the Gram matrix achieving the minimum in the definition of $\mathrm{MADV}_\varepsilon^c(\Phi)$ as $N = \sum_{x,y} \langle \psi_x | \psi_y \rangle |y\rangle\langle x|$, where $|\psi_x\rangle = \sum_i p_i(x) |i\rangle$ is the final state of the algorithm on input x expressed in the computational basis. After relaxing the normalization condition on the states $|\psi_x\rangle$, we obtain that $\mathrm{MADV}^c_\varepsilon(\Phi) \geq \frac{1}{\log c} \log \frac{1}{2^n} \sum_S c^{|S|} |\hat{p}(S)|^2$, where p is the minimum taken over all polynomials $q : \{0,1\}^n \mapsto \mathbb{R}$ satisfying $\sqrt{1-\varepsilon} \leq (-1)^{f(x)} q(x) \leq 1$ for any $x \in \{0,1\}^n$.

By definition of $\mathrm{MADV}^c_\varepsilon(\Phi)$, we can then show that similarly to the lower bound in Theorem 14, we have $\mathrm{MADV}^c_\varepsilon(\Phi) \geq T - \frac{n-\log\delta}{\log c}$, where in this case $T = \widetilde{\deg}_\varepsilon(f)$ and $\delta = \sum_{S:|S|\geq T} |\hat{p}(S)|^2$. It can then be shown that δ must be at least $\frac{\varepsilon^2}{2^n}$, otherwise truncating the high Fourier coefficients from p would yield a polynomial witnessing that $\widetilde{\deg}_\varepsilon(f) < T$, a contradiction. ◀

Note that a similar argument cannot be used for the extended polynomial method because truncating the large Fourier coefficients from a Gram matrix N might yield a matrix that is not normalized (i.e., violating the constraint $N \circ \mathbb{I} = \mathbb{I}$).

6 Discussion and open questions

Strong connections have been known for quite some time between the approximate degree of a function and its query complexity: they are polynomially related for all (total) functions for classical complexity [24] as well as for quantum complexity [12]. The latter is actually often equal to the approximate degree (at least up to a constant factor) for many functions, including all symmetric functions and random functions. With a large number of tight bounds proved using the polynomial method [12, 1, 3, 8] to cite only a few, this method might even seem ubiquitous. However, it is not always tight as in some rare cases the adversary method is known to yield better bounds. By clarifying the relation between the polynomial and adversary bounds, this work provides some new insight on why this can be the case.

First, we showed that the polynomial method is a relaxation of a more general method which we called the extended polynomial method. This has a particularly nice interpretation when one wants to compute the value of a function in a register, i.e., the goal is to prepare the state $|f(x)\rangle$.[3] When error ε is allowed, measuring this register should yield outcome $f(x)$ with probability at least $1-\varepsilon$, that is, the probability $p(x)$ of obtaining outcome 1 should be close to 1 when $f(x) = 1$ and close to 0 when $f(x) = 0$. While the polynomial method only considers the degree of the probability $p(x)$, the extended polynomial method considers the degree of all the amplitudes in the final state of the algorithm, including the erroneous part. In terms of Gram matrices this corresponds to relaxing the condition $N \circ \mathbb{I} = \mathbb{I}$ to $N \circ \mathbb{I} \preceq \mathbb{I}$.[4]

In general it is not known how large the gap between the polynomial and the extended polynomial method can be. It appears to be larger by at least a factor two for some functions. Indeed, Ambainis *et al.* improved the lower bound for random Boolean functions from $n/4 - o(n)$ using the polynomial method, to $n/2 - o(n)$ (which is tight) by bounding the degree of all amplitudes in the final state of the algorithm [6] (their argument can be seen as a special case of the extended polynomial method).

[3] This is the standard problem studied in most articles on quantum query complexity, even though some recent works including this one have considered the problem of computing the function in the phase. Recall that Claim 4 implies that both problems are equivalent.

[4] Note that with the relaxed condition $N \circ \mathbb{I} \preceq \mathbb{I}$, the matrix N does not have to be a *normalized* Gram matrix anymore, in which case the Hadamard product fidelity is not defined. However, one can use another output condition, for example $\gamma_2(N - M) \leq \sqrt{2\varepsilon}$, where γ_2 denotes the Hadamard product trace norm. These output conditions are related up to a constant [22, 23], so that it only affects the lower bound by at most a constant factor for bounded-error query complexity.

Secondly this provides a partial answer on how the multiplicative adversary method MADV^c varies with c. Indeed, while it was already known that $\mathrm{MADV}_\varepsilon^{c\to 1}(f) \geq \mathrm{ADV}_\varepsilon^{\pm}(f)$, we have proved that $\mathrm{MADV}_\varepsilon^{c\to\infty}(f) \geq \widetilde{\deg}_\varepsilon(f)$, and in particular, $\mathrm{MADV}_0^{c\to\infty}(f) = \deg(f)$ in the zero-error case. This implies that the gap between MADV and $\mathrm{MADV}^{c\to\infty}$ can be at least polynomially large by considering the Ambainis function [4], for which the polynomial method fails to give a tight bound, contrary to the adversary method. This gap might be explained by the fact that in the limit $c \to \infty$, the eigenbasis of the best adversary matrix is restricted to be the Fourier basis, while for smaller values, other bases can provide better bounds.

To summarize our current knowledge, the situation is the following. On the one hand, when c tends to one, the multiplicative adversary method is tight for bounded-error ([9]) but not for zero-error (e.g., for the OR function, there is a quadratic gap). On the other hand, when c tends to infinity, the multiplicative method seems better for zero-error as it proves the $\Omega(n)$ lower bound for OR, but it is not always tight (Ambainis function). As for low success probability, it seems that taking c bounded away from one provides an advantage, as shown in particular by the strong direct product theorems proved using the multiplicative [28, 23] and polynomial methods [20, 27].

This leaves open a few interesting questions about the behavior of the multiplicative adversary method. Can we say more about the dependence of MADV^c on c? Can we improve the relation $\mathrm{MADV}_\varepsilon^{c\to 1}(M) \geq \mathrm{ADV}_\varepsilon^{\pm}(M)$ to an equality in general? Can we characterize the set of functions for which the (extended or not) polynomial method does not provide a tight bound? Finally, does the multiplicative adversary method characterize the quantum query complexity, i.e., is it tight for any error?

Acknowledgements Most of this work was done at NEC Laboratories America. The authors thank M. Rötteler, D. Gavinsky, and T. Lee for stimulating discussions; and R. de Wolf and R. Špalek for interesting comments. They also thank R. Špalek for proposing the alternative proof of Lemma 15 using dual polynomials.

References

1 S. Aaronson and Y. Shi. Quantum lower bounds for the collision and the element distinctness problems. *J. ACM*, 51(4):595–605, 2004.

2 A. Ambainis. Quantum lower bounds by quantum arguments. *J. Comput. Sys. Sci.*, 64(4):750–767, 2002.

3 A. Ambainis. Polynomial degree and lower bounds in quantum complexity: Collision and element distinctness with small range. *Theor. Comput.*, 1:37–46, 2005.

4 A. Ambainis. Polynomial degree vs. quantum query complexity. *J. Comput. Sys. Sci.*, 72(2):220–238, 2006.

5 A. Ambainis. A new quantum lower bound method, with an application to strong direct product theorem for quantum search. *Theor. Comput.*, 6:1–25, 2010.

6 A. Ambainis, A. Bačkurs, J. Smotrovs, and R. de Wolf. Optimal quantum query bounds for almost all Boolean functions. In *Proc. STACS'13*, 2013.

7 A. Ambainis, A. M. Childs, B. W. Reichardt, R. Špalek, and S. Zhang. Any AND-OR formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer. *SIAM J. Comput.*, 39(6):2513–2530, 2010.

8 A. Ambainis and R. de Wolf. How low can approximate degree and quantum query complexity be for total boolean functions? *arXiv:1206.0717*, 2012.

9 A. Ambainis, L. Magnin, M. Roetteler, and J. Roland. Symmetry-assisted adversaries for quantum state generation. In *Proc. CCC'11*, pages 167–177, 2011.

10 A. Ambainis, R. Špalek, and R. de Wolf. A new quantum lower bound method, with applications to direct product theorems and time-space tradeoffs. In *Proc. STOC'06*, pages 618–633, 2006.

11 H. Barnum and M. Saks. A lower bound on the quantum query complexity of read-once functions. *J. Comput. Sys. Sci.*, 69(2):244–258, 2004.

12 R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf. Quantum lower bounds by polynomials. *J. ACM*, 48:778–797, 2001.

13 A. Belovs. Adversary lower bound for element distinctness. *arXiv:1204.5074*, 2012.

14 A. Belovs and R. Špalek. Adversary lower bound for the k-sum problem. In *Proc. ITCS'13*, 2013.

15 C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani. Strengths and weaknesses of quantum computing. *SIAM J. Comput.*, 26(5):1510–1523, 1997.

16 H. Buhrman and R. de Wolf. Complexity measures and decision tree complexity: A survey. *Theor. Comput. Sci.*, 288(1):21–43, 2002.

17 E. Farhi, J. Goldstone, and S. Gutmann. A quantum algorithm for the Hamiltonian NAND tree. *Theor. Comput.*, 4:169–190, 2008.

18 P. Høyer, T. Lee, and R. Špalek. Negative weights make adversaries stronger. In *Proc. STOC'07*, pages 526–535, 2007.

19 P. Høyer, J. Neerbek, and Y. Shi. Quantum complexities of ordered searching, sorting, and element distinctness. *Algorithmica*, 34(4):429–448, 2008.

20 H. Klauck, R. Špalek, and R. de Wolf. Quantum and classical strong direct product theorems and optimal time-space tradeoffs. *SIAM J. Comput.*, 36(5):1472–1493, 2007.

21 S. Laplante and F. Magniez. Lower bounds for randomized and quantum query complexity using Kolmogorov arguments. *SIAM J. Comput.*, 38(1):46–62, 2008.

22 T. Lee, R. Mittal, B. W. Reichardt, R. Špalek, and M. Szegedy. Quantum query complexity of state conversion. In *Proc. FOCS'11*, pages 344–353, 2011.

23 T. Lee and J. Roland. A strong direct product theorem for quantum query complexity. In *Proc. CCC'12*, pages 236 – 246, 2012.

24 N. Nisan and M. Szegedy. On the degree of Boolean functions as real polynomials. *Comput. Complex.*, 4:301–313, 1994.

25 B. W. Reichardt. Reflections for quantum query algorithms. In *Proc. SODA'11*, pages 560–569, 2011.

26 B. W. Reichardt and R. Špalek. Span-program-based quantum algorithm for evaluating formulas. In *Proc. STOC'08*, pages 103–112, 2008.

27 A. A. Sherstov. Strong direct product theorems for quantum communication and query complexity. In *Proc. STOC'11*, pages 41–50, 2011.

28 R. Špalek. The multiplicative quantum adversary. In *Proc. CCC'08*, pages 237–248, 2008.

29 R. Špalek and M. Szegedy. All quantum adversary methods are equivalent. *Theor. Comput.*, 2:1–18, 2006.

30 S. Zhang. On the power of Ambainis lower bounds. *Theor. Comput. Sci.*, 339(2):241–256, 2005.

Optimal quantum query bounds for almost all Boolean functions*

Andris Ambainis[1], Arturs Bačkurs[2], Juris Smotrovs[1], and Ronald de Wolf[3]

1 University of Latvia
Riga, Latvia
{ambainis,Juris.Smotrovs}@lu.lv

2 MIT, Cambridge, MA
(work done while at University of Latvia)
abackurs@gmail.com

3 CWI and University of Amsterdam
Amsterdam, The Netherlands
rdewolf@cwi.nl

Abstract

We show that almost all n-bit Boolean functions have bounded-error quantum query complexity at least $n/2$, up to lower-order terms. This improves over an earlier $n/4$ lower bound of Ambainis [1], and shows that van Dam's oracle interrogation [9] is essentially optimal for almost all functions. Our proof uses the fact that the acceptance probability of a T-query algorithm can be written as the sum of squares of degree-T polynomials.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases quantum computing, query complexity, lower bounds, polynomial method

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.446

1 Introduction

Most known quantum algorithms have been developed in the setting of quantum query complexity, which is the quantum generalization of the model of decision tree complexity. Here an algorithm is charged for each "query" to the input bits, while intermediate computation is free (see [8] for more details about this model). For certain specific functions one can obtain large quantum-speedups in this model. For example, Grover's algorithm [14] computes the n-bit OR function with $O(\sqrt{n})$ queries, while any classical algorithm needs $\Omega(n)$ queries. Many more such polynomial speed-ups are known, see for example [3, 18, 11, 6]. If one considers partial functions there are even exponential speed-ups, for example [10, 20, 19, 5]. Substantial quantum speed-ups are quite rare, and exploit very specific structure in problems that makes those problems amenable to quantum speed-ups.

On the other hand, one can also obtain a smaller speed-up that holds for *almost all* Boolean functions. Classically, almost all Boolean functions $f : \{0,1\}^n \to \{0,1\}$ have

* AA, AB, RdW are partially supported by the European Commission under the project QCS (Grant No. 255961). AA and JS are partially supported by ESF project 1DP/1.1.1.2.0/09/APIA/VIAA/044. RdW is partially supported by a Vidi grant from the Netherlands Organization for Scientific Research (NWO).

bounded-error query complexity n, minus lower-order terms. This is quite intuitive: if we have only seen 99% of the n input bits, then the restriction of a random function to the 1% remaining variables will still be roughly balanced between 0 and 1-inputs. In contrast, van Dam [9] exhibited a beautiful quantum algorithm that recovers the complete n-bit input x with high probability using roughly $n/2$ quantum queries. Briefly, his algorithm is as follows:

1. With $T = n/2 + O(\sqrt{n \log(1/\varepsilon)})$ and $B = \sum_{i=0}^{T} \binom{n}{i}$ being the number of $y \in \{0,1\}^n$ with Hamming weight $|y| \leq T$, set up the n-qubit superposition $\frac{1}{\sqrt{B}} \sum_{y \in \{0,1\}^n : |y| \leq T} |y\rangle$.
2. Apply the unitary $|y\rangle \mapsto (-1)^{x \cdot y}|y\rangle$. We can implement this using T queries to the input x, for all basis states $|y\rangle$ with $|y| \leq T$.
3. Apply a Hadamard transform to all qubits and measure.

To see correctness of this algorithm, note that the fraction of n-bit strings y of Hamming weight larger than T is $\ll \varepsilon$. Hence the state obtained in step 2 is very close to the state $\frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} (-1)^{x \cdot y}|y\rangle$, whose Hadamard transform is exactly $|x\rangle$.

Since obtaining x suffices to compute $f(x)$ for any f of our choice, van Dam's algorithm implies that the ε-error quantum query complexity of f is

$$Q_\varepsilon(f) \leq n/2 + O(\sqrt{n \log(1/\varepsilon)}) \qquad \text{for all Boolean functions.}$$

It is known that this upper bound is essentially tight for *some* Boolean functions. For example, $Q_\varepsilon(f) = \lceil n/2 \rceil$ for the n-bit Parity function [4, 12]. Our goal in this paper is to show that it is tight for *almost all* Boolean functions, i.e., that $Q_\varepsilon(f)$ is essentially lower bounded by $n/2$ for almost all f (and fixed ε). How can we prove such a lower bound? Two general methods are known for proving quantum query lower bounds: the polynomial method [4] and the adversary method [2, 15]. As we explain below, in their standard form neither method is strong enough to prove our desired $n/2$ lower bound.

First, the adversary method in its strongest incarnation [15, Theorem 2] has the form

$$Q_\varepsilon(f) \geq \frac{1}{2}(1 - \sqrt{\varepsilon(1-\varepsilon)})\mathrm{ADV}^{\pm}(f),$$

where the "negative-weights adversary bound" $\mathrm{ADV}^{\pm}(f)$ is a quantity that is at most n. Accordingly, for constant error probability ε the adversary method can only prove lower bounds of the form cn for some $c < 1/2$.

Second, the polynomial method uses the fact (first proved in [13, 4]) that the acceptance probability of a T-query algorithm can be written as a degree-$2T$ n-variate multilinear real polynomial $p(x)$ of the input. If the algorithm computes f with error probability $\leq \varepsilon$, then $p(x)$ will approximate $f(x)$: $p(x) \in [0, \varepsilon]$ for every $x \in f^{-1}(0)$ and $p(x) \in [1-\varepsilon, 1]$ for every $x \in f^{-1}(1)$. Accordingly, a lower bound of d on the ε-approximate polynomial degree $\deg_\varepsilon(f)$ implies a lower bound of $d/2$ on the ε-error quantum query complexity of f. This is how Ambainis [1] proved the current best lower bound of roughly $n/4$ that holds for almost all n-bit Boolean functions: he showed that almost all f satisfy $\deg_\varepsilon(f) \geq (1/2 - o(1))n$. However, O'Donnell and Servedio [17] proved a nearly matching upper bound: $\deg_\varepsilon(f) \leq (1/2 + o(1))n$ for almost all f. Hence Ambainis's lower bound approach via approximate degree cannot be improved to obtain our desired lower bound of $n/2$ on $Q_\varepsilon(f)$.[1] This suggests that also the polynomial method is unable to obtain the conjectured factor 1/2 in the lower bound.

[1] In fact, the *unbounded-error* quantum query complexity of almost all Boolean functions is only $n/4$ up to lower-order terms. This follows from the degree upper bound of [17] combined with [7, Theorem 1] and the fact that d-bit Parity can be computed with $\lceil d/2 \rceil$ quantum queries.

However, looking under the hood of the polynomial method, it actually gives a bit more information about the acceptance probability: $p(x)$ is not an arbitrary degree-$2T$ polynomial, but the sum of squares of degree-T polynomials. Using this extra information, we prove in this paper that indeed $Q_\varepsilon(f) \geq n/2$ up to lower-order terms for almost all f.[2]

Our main technical result will be a claim about certain random matrices (Claim 1 below), which may have further applications. It says the following. Let $\mathcal{B} = \{x \in \{0,1\}^n : |x| \leq T\}$ be the set of strings of weight at most T, and $B = |\mathcal{B}|$ its size. Suppose F is a $2^n \times 2^n$ diagonal matrix with randomly chosen signs on its diagonal, and $\widehat{F} = HFH$ is F conjugated with the unitary Hadamard transform. Then the principal minor of $\widehat{F}$ restricted to entries in $\mathcal{B} \times \mathcal{B}$ has (with probability $1 - o(1)$) operator norm $O(\sqrt{nB^{1+o(1)}/2^n})$. In particular, if $T \leq (1/2 - \varepsilon)n$ for any fixed positive ε then with high probability this operator norm is $o(1)$.

2 Proof

Suppose we have a quantum algorithm that uses T queries to its n-bit input x. Then by [4, Lemma 4.1], its final state can be written as a function of the input as

$$\sum_z \alpha_z(x)|z\rangle,$$

where z ranges over the computational basis states of the algorithm's space, and the amplitudes $\alpha_z(x)$ are complex-valued multilinear n-variate polynomials of degree $\leq T$. We assume w.l.o.g. that the algorithm determines its Boolean output by measuring the first qubit of the final state. Then the acceptance probability (as a function of input x) is the following polynomial of degree $\leq 2T$:

$$p(x) = \sum_{z:z_1=1} |\alpha_z(x)|^2.$$

Let $\alpha_z \in \mathbb{C}^{2^n}$ denote the vector with entries $\alpha_z(x)$. Define the following $2^n \times 2^n$ matrix P:

$$P = \sum_{z:z_1=1} \alpha_z \alpha_z^*.$$

The diagonal entry P_{xx} of this matrix is $p(x)$. Since P is positive semidefinite, we have[3]

$$\|P\|_1 = \mathrm{Tr}(P) = \sum_{x\in\{0,1\}^n} p(x).$$

With H denoting the n-qubit Hadamard transform, $H\alpha_z$ is proportional to the Fourier transform $\widehat{\alpha_z}$, which has support only on the $B = \sum_{i=0}^{T} \binom{n}{i}$ monomials of degree $\leq T$. Hence the matrix HPH has support only on a $B \times B$ submatrix.

It will be convenient to use $+1$ and -1 as the range of a Boolean function, rather than 0 and 1. Consider Boolean function $f : \{0,1\}^n \to \{\pm 1\}$. For $s \in \{0,1\}^n$, the corresponding

[2] Magnin and Roland [16] independently found similar ways to strengthen the standard polynomial method; however they do not apply their tools to the analysis of random Boolean functions.

[3] We use the following matrix-analytic notation. For $m \times m$ matrices A and A', define inner product $\langle A, A'\rangle = \mathrm{Tr}(A^*A') = \sum_{i,j} A^*_{ij}A'_{ij}$. Note that this inner product is basis-independent: for every unitary U we have $\langle UAU^*, UA'U^*\rangle = \langle A, A'\rangle$. Let $\|A\|_p$ denote the (unitarily invariant) Schatten p-norm of A, which is the p-norm of the m-dimensional vector of singular values of A. In particular, $\|A\|_1$ is the sum of the singular values of A, and $\|A\|_\infty$ is its largest singular value, which is the operator norm of A. It is easy to see that $\|A\|_2^2 = \mathrm{Tr}(A^*A) = \sum_{i,j} |A_{ij}|^2$, and $\langle A, B\rangle \leq \|A\|_1\|B\|_\infty$.

Fourier coefficient of f is defined as $\widehat{f}(s) = \frac{1}{2^n}\sum_x (-1)^{s\cdot x} f(x)$. Let F be the $2^n \times 2^n$ diagonal matrix with diagonal entries $f(x)$. Define $\widehat{F} = HFH$. Then for $s,t \in \{0,1\}^n$, we have

$$\widehat{F}_{s,t} = \langle s|HFH|t\rangle = \frac{1}{2^n}\sum_{x,y}(-1)^{s\cdot x}(-1)^{t\cdot y}F_{xy} = \frac{1}{2^n}\sum_x (-1)^{(s\oplus t)\cdot x} f(x) = \widehat{f}(s\oplus t).$$

Let $\widehat{F}_T$ denote $\widehat{F}$ after zeroing out all s,t-entries where $|s| > T$ and/or $|t| > T$. Note that HPH doesn't have support on the entries that are zeroed out, hence $\langle HPH, \widehat{F}\rangle = \langle HPH, \widehat{F}_T\rangle$.

Suppose our T-query quantum algorithm computes f with worst-case error probability at most some fixed constant $\leq \varepsilon$. Output 1 means the algorithm thinks $f(x) = 1$, and output 0 means it thinks $f(x) = -1$. Then for every $x \in \{0,1\}^n$, $2p(x) - 1$ differs from $f(x)$ by at most 2ε. Hence:

$$\begin{aligned}
(1-2\varepsilon)2^n &\leq \langle 2P - I, F\rangle \\
&= 2\langle P, F\rangle - \sum_x f(x) \\
&= 2\langle HPH, \widehat{F}\rangle - \sum_x f(x) \\
&= 2\langle HPH, \widehat{F}_T\rangle - \sum_x f(x) \\
&\leq 2\|P\|_1 \left\|\widehat{F}_T\right\|_\infty - \sum_x f(x) \\
&= 2\left\|\widehat{F}_T\right\|_\infty \sum_x p(x) - \sum_x f(x).
\end{aligned}$$

We can assume w.l.o.g. that $\sum_x f(x) \geq 0$ (if this doesn't hold for f then just take its negation, which has the same query complexity as f). Since $\sum_x p(x) \leq 2^n$, we get

$$\left\|\widehat{F}_T\right\|_\infty \geq 1/2 - \varepsilon. \tag{1}$$

The technically hard part is to upper bound $\left\|\widehat{F}_T\right\|_\infty$ for most f. So consider the case where $f : \{0,1\}^n \to \{\pm 1\}$ is a *uniformly random* function, meaning that the 2^n values $f(x)$ are independent uniformly random signs. In the next subsection we show

▶ Claim 1. With probability $1 - o(1)$ (over the choice of f) we have $\left\|\widehat{F}_T\right\|_\infty = O\left(\sqrt{\frac{nB^{1+o(1)}}{2^n}}\right)$.

Combining this with the lower bound (1), we get that $B \geq 2^{n-o(n)}$. On the other hand, a well-known upper bound on the sum of binomial coefficients is $B = \sum_{i=0}^{T}\binom{n}{i} \leq 2^{nH(T/n)}$, where $H(q) = -q\log q - (1-q)\log(1-q)$ denotes the binary entropy function. Hence, $2^{n-o(n)} \leq 2^{nH(T/n)}$ which implies $T \geq n/2 - o(n)$. This shows that $Q_\epsilon(f) \geq n/2 - o(n)$ for almost all f (and fixed constant ε).

2.1 Proof of Claim 1

Below, unless mentioned otherwise, probabilities and expectations will be taken over the random choice of f. We choose $T = n/2 - o(n)$ sufficiently small that $B = \sum_{i=0}^{T}\binom{n}{i} = o(2^n)$, i.e., the $o(n)$ term in T is taken to be $\omega(\sqrt{n})$.

Let λ_i be the i-th eigenvalue of $\widehat{F}_T$. Since $\widehat{F}_T$ is symmetric we have

$$\left\|\widehat{F}_T\right\|_\infty = \max_i |\lambda_i| = \sqrt[2k]{\max_i \lambda_i^{2k}} \leq \sqrt[2k]{\sum_i \lambda_i^{2k}} = \sqrt[2k]{\mathrm{Tr}(\widehat{F}_T^{2k})}.$$

We are going to show that

$$\mathbb{E}\left[\mathrm{Tr}(\widehat{F}_T^{2k})\right] = O\left(B\left(B/2^n\right)^k\right) \tag{2}$$

for every constant k (with a big-O constant depending on k). This means that, using Markov's inequality,

$$\begin{aligned}\Pr\left[\left\|\widehat{F}_T\right\|_\infty > C\sqrt{nB^{1+1/k}/2^n}\right] &\le \Pr\left[\sqrt[2k]{\mathrm{Tr}(\widehat{F}_T^{2k})} > C\sqrt{nB^{1+1/k}/2^n}\right]\\ &= \Pr\left[\mathrm{Tr}(\widehat{F}_T^{2k}) > C^{2k}n^kB^{k+1}/2^{nk}\right]\\ &\le \frac{\mathbb{E}\left[\mathrm{Tr}(\widehat{F}_T^{2k})\right]}{C^{2k}n^kB^{k+1}/2^{nk}} = o(1).\end{aligned}$$

Since this is true for any constant k, Claim 1 follows.

So now our goal is to prove (2). Below we let each of $s_1,\dots,s_{2k}$ range over the B n-bit strings of weight $\le T$, and each of $x_1,\dots,x_{2k}$ range over $\{0,1\}^n$. For simplicity we abbreviate $\vec{s} = s_1,s_2,\dots,s_{2k}$ and $\vec{x} = x_1,x_2,\dots,x_{2k}$. Writing out the $2k$-fold matrix product, we have

$$\begin{aligned}\mathbb{E}\left[\mathrm{Tr}(\widehat{F}_T^{2k})\right] &= \mathbb{E}\left[\sum_{\vec{s}}\widehat{f}(s_1\oplus s_2)\widehat{f}(s_2\oplus s_3)\cdots\widehat{f}(s_{2k}\oplus s_1)\right] &(3)\\ &= \frac{1}{2^{2nk}}\sum_{\vec{s}}\sum_{\vec{x}}\mathbb{E}\left[(-1)^{(s_1\oplus s_2)\cdot x_1}f(x_1)\cdots(-1)^{(s_{2k}\oplus s_1)\cdot x_{2k}}f(x_{2k})\right] &(4)\\ &= \frac{1}{2^{2nk}}\sum_{\vec{s}}\sum_{\vec{x}}(-1)^{(s_1\oplus s_2)\cdot x_1+\dots+(s_{2k}\oplus s_1)\cdot x_{2k}}\,\mathbb{E}\left[f(x_1)\cdots f(x_{2k})\right]. &(5)\end{aligned}$$

For a particular $y\in\{0,1\}^n$, there are as many Boolean functions having $f(y)=1$ as having $f(y)=-1$, independently of what is known about values of f on other inputs. Thus, if any y occurs an odd number of times in $\vec{x}=(x_1,\dots,x_{2k})$, then $\mathbb{E}[f(x_1)\cdots f(x_{2k})]=0$. So only those summands are left where all multiplicities of distinct values among $x_1,\dots,x_{2k}$ are even. We call such $\vec{x}$ *even.* We have

$$\begin{aligned}\mathbb{E}\left[\mathrm{Tr}(\widehat{F}_T^{2k})\right] &= \frac{1}{2^{2nk}}\sum_{\vec{s}}\sum_{\vec{x}\text{ even}}(-1)^{\sum_{i=1}^{2k}(s_i\oplus s_{i+1})\cdot x_i}\\ &= \frac{1}{2^{2nk}}\sum_{r}\sum_{\substack{\text{partition of}\\ \{1,\dots,2k\}\text{ into even}\\ \text{non-empty } I_1,\dots,I_r}}\sum_{\vec{s}}\sum_{\substack{x^{(1)},\dots,x^{(r)}\\ \text{different}}}(-1)^{\sum_{j=1}^{r}\left(\bigoplus_{i\in I_j}(s_i\oplus s_{i+1})\right)\cdot x^{(j)}} &(6)\end{aligned}$$

where $s_{2k+1}=s_1$ and the second summation is over all partitions of $\{1,\dots,2k\}$ into even-sized non-empty parts $I_1,\dots,I_r$ with the implied condition that $x_i=x_j$ iff i and j belong to the same part. Since the number of such partitions $(I_1,I_2,\dots,I_r)$ depends only on k (which is a constant), it suffices to prove that each term in the sum is of the order $O(B(B/2^n)^k)$. We will do this by proving

▶ Claim 2. For any fixed m and any partition $I_1,\dots,I_r$ of $\{1,\dots,m\}$:

$$\sum_{\vec{s}}\sum_{\substack{x^{(1)},\dots,x^{(r)}\\ \text{different}}}(-1)^{\sum_{j=1}^{r}t_j(\vec{s})\cdot x^{(j)}} = O(B^{m-r+1}\cdot 2^{nr}) \tag{7}$$

where $t_j(\vec{s})=\bigoplus_{i\in I_j}(s_i\oplus s_{i+1})$, $s_{m+1}=s_1$, and the big-O constant depends on m and the partition.

We first show that Claim 2 implies Claim 1. In our case, $m = 2k$. Since $B = o(2^n)$, the upper bound $B^{2k-r+1} \cdot 2^{nr}$ increases when r increases. Since each partition of $\{1, \ldots, 2k\}$ into even-sized non-empty parts $I_1, \ldots, I_r$ must contain at least 2 elements in each I_j, we must have $r \le (2k)/2 = k$ and every term of the sum (6) is upper bounded by

$$\frac{1}{2^{2nk}} O\left(B^{2k-k+1} \cdot 2^{nk}\right) = O\left(B\left(B/2^n\right)^k\right).$$

It remains to prove Claim 2, which we do by induction on r. If $r = 1$ then $t_1(\vec{s}) = \oplus_{i=1}^m (s_i \oplus s_{i+1})$ includes each s_i exactly twice and hence sums to the all-0 string, hence

$$\sum_{\vec{s}} \sum_{x \in \{0,1\}^n} (-1)^{t_1(\vec{s}) \cdot x} = \sum_{\vec{s}} \sum_{x \in \{0,1\}^n} (-1)^{0 \cdot x} = B^m \cdot 2^n.$$

For the inductive step, suppose Claim 2 is true for $r - 1$. Rewrite the left-hand side of (7) as

$$\sum_{\vec{s}} \sum_{\substack{x^{(1)}, \ldots, x^{(r)} \\ \text{different}}} (-1)^{\sum_{j=1}^r t_j(\vec{s}) \cdot x^{(j)}}$$

$$= \sum_{\vec{s}} \sum_{x^{(1)}} \sum_{\substack{x^{(2)}, \ldots, x^{(r)} \\ \text{different}}} (-1)^{\sum_{j=1}^r t_j(\vec{s}) \cdot x^{(j)}} - \sum_{\vec{s}} \sum_{a=2}^{r} \sum_{\substack{x^{(2)}, \ldots, x^{(r)} \\ \text{different, } x^{(1)} = x^{(a)}}} (-1)^{\sum_{j=1}^r t_j(\vec{s}) \cdot x^{(j)}}. \tag{8}$$

Let us estimate both sums of (8). Since $\sum_{x^{(1)}} (-1)^{t_1(\vec{s}) x^{(1)}}$ equals 2^n if $t_1(\vec{s}) = 0^n$, and that sum equals 0 otherwise, the first sum of (8) equals

$$2^n \sum_{\vec{s}: t_1(\vec{s}) = 0} \sum_{\substack{x^{(2)}, \ldots, x^{(r)} \\ \text{different}}} (-1)^{\sum_{j=2}^r t_j(\vec{s}) \cdot x^{(j)}}. \tag{9}$$

We now transform this sum into the form of the left-hand side of (7), with both m and r smaller by 1 compared to their current values. After that, we will apply the induction hypothesis.

Let ℓ be such that $\ell \in I_1$, $\ell - 1 \notin I_1$. Then $t_1(\vec{s})$ contains s_ℓ with coefficient 1 (because $t_1(\vec{s})$ includes $s_\ell \oplus s_{\ell+1}$ but not $s_{\ell-1} \oplus s_\ell$). We can use the condition $t_1(\vec{s}) = 0$ to express s_ℓ in terms of $s_1, \ldots, s_{\ell-1}$ and $s_{\ell+1}, \ldots, s_m$ as follows:

$$s_\ell = s_{\ell+1} \oplus \bigoplus_{i \in I_1 : i \ne \ell} (s_i \oplus s_{i+1}). \tag{10}$$

Let b be such that $\ell - 1 \in I_b$. Then $t_b(\vec{s})$ contains $s_{\ell-1} \oplus s_\ell$ and we can substitute (10) into $t_b(\vec{s})$, obtaining

$$t_b(\vec{s}) = s_{\ell-1} \oplus s_{\ell+1} \oplus \bigoplus_{i \in I_1 : i \ne \ell} (s_i \oplus s_{i+1}) \oplus \bigoplus_{i \in I_b : i \ne \ell - 1} (s_i \oplus s_{i+1}).$$

We can now remove the variable s_ℓ (because it was only contained in $s_{\ell-1} \oplus s_\ell$ and $s_\ell \oplus s_{\ell+1}$) and redefine I_b to be $I_1 \cup I_b \setminus \{\ell\}$. Then we get that (9) is equal to

$$2^n \sum_{\substack{s_1, \ldots, s_{\ell-1} \\ s_{\ell+1}, \ldots, s_m}} \sum_{\substack{x^{(2)}, \ldots, x^{(r)} \\ \text{different}}} (-1)^{\sum_{j=2}^r t_j(\vec{s}) \cdot x^{(j)}} = 2^n \cdot O\left(B^{m-r+1} \cdot 2^{n(r-1)}\right) = O\left(B^{m-r+1} \cdot 2^{nr}\right)$$

with the estimate following from the induction hypothesis (with both m and r being smaller by 1).

As for the second sum of (8), it is equal to

$$\sum_{a=2}^{r}\sum_{\vec{s}}\sum_{\substack{x^{(2)},\ldots,x^{(r)}\\ \text{different}}}(-1)^{\sum_{j=2}^{r} t_j^{(a)}(\vec{s})\cdot x^{(j)}} = O\left(B^{m-r+2}\cdot 2^{n(r-1)}\right)$$

where $t_j^{(a)}(\vec{s}) = t_j(\vec{s})$ except for $t_a^{(a)}(\vec{s}) = t_a(\vec{s}) \oplus t_1(\vec{s})$ (thus merging the partition parts I_1 and I_a). We have eliminated $x^{(1)}$ and apply the induction hypothesis (with r being smaller by 1 and m remaining the same). The outer sum over a introduces only a factor depending on $r \leq m$.

Since $B = o(2^n)$ we have $B^{m-r+2} \cdot 2^{n(r-1)} = o(B^{m-r+1} \cdot 2^{nr})$. Hence the bound on the first sum in (8) is of a larger order and we have completed the proof of Claim 2.

Acknowledgement

We thank Loïck Magnin and Jérémie Roland for sending us a copy of [16].

References

1 A. Ambainis. A note on quantum black-box complexity of almost all Boolean functions. *Information Processing Letters*, 71(1):5–7, 1999. quant-ph/9811080.

2 A. Ambainis. Quantum lower bounds by quantum arguments. *Journal of Computer and System Sciences*, 64(4):750–767, 2002. Earlier version in STOC'00. quant-ph/0002066.

3 A. Ambainis. Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007. Earlier version in FOCS'04. quant-ph/0311001.

4 R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf. Quantum lower bounds by polynomials. *Journal of the ACM*, 48(4):778–797, 2001. Earlier version in FOCS'98. quant-ph/9802049.

5 N. de Beaudrap, R. Cleve, and J. Watrous. Sharp quantum vs. classical query complexity separations. *Algorithmica*, 34(4):449–461, 2002. quant-ph/0011065.

6 A. Belovs. Span programs for functions with constant-sized 1-certificates. In *Proceedings of 43rd ACM STOC*, pages 77–84, 2012. arXiv:1105.4024.

7 H. Buhrman, N. Vereshchagin, and R. de Wolf. On computation and communication with small bias. In *Proceedings of 22nd IEEE Conference on Computational Complexity*, pages 24–32, 2007.

8 H. Buhrman and R. de Wolf. Complexity measures and decision tree complexity: A survey. *Theoretical Computer Science*, 288(1):21–43, 2002.

9 W. van Dam. Quantum oracle interrogation: Getting all information for almost half the price. In *Proceedings of 39th IEEE FOCS*, pages 362–367, 1998. quant-ph/9805006.

10 D. Deutsch and R. Jozsa. Rapid solution of problems by quantum computation. In *Proceedings of the Royal Society of London*, volume A439, pages 553–558, 1992.

11 C. Dürr, M. Heiligman, P. Høyer, and M. Mhalla. Quantum query complexity of some graph problems. *SIAM Journal on Computing*, 35(6):1310–1328, 2006. Earlier version in ICALP'04.

12 E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser. A limit on the speed of quantum computation in determining parity. *Physical Review Letters*, 81:5442–5444, 1998. quant-ph/9802045.

13 L. Fortnow and J. Rogers. Complexity limitations on quantum computation. *Journal of Computer and System Sciences*, 59(2):240–252, 1999. Earlier version in Complexity'98. Also cs.CC/9811023.

14 L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of 28th ACM STOC*, pages 212–219, 1996. quant-ph/9605043.

15 P. Høyer, T. Lee, and R. Špalek. Negative weights make adversaries stronger. In *Proceedings of 39th ACM STOC*, pages 526–535, 2007. quant-ph/0611054.

16 L. Magnin and J. Roland. Explicit relation between all lower bound techniques for quantum query complexity. In *Proceedings of 30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, 2013. arXiv:1209.2713.

17 R. O'Donnell and R. Servedio. Extremal properties of polynomial threshold functions. *Journal of Computer and System Sciences*, 74(3):298–312, 2008. Earlier version in Complexity'03.

18 M. Santha. Quantum walk based search algorithms. In *Proceedings of 5th TAMC*, pages 31–46, 2008. arXiv/0808.0059.

19 P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. Earlier version in FOCS'94. quant-ph/9508027.

20 D. Simon. On the power of quantum computation. *SIAM Journal on Computing*, 26(5):1474–1483, 1997. Earlier version in FOCS'94.

Streaming Complexity of Checking Priority Queues*†

Nathanaël François[1] and Frédéric Magniez[2]

1 Univ Paris Diderot, Sorbonne Paris-Cité, LIAFA, CNRS, 75205 Paris, France
nathanael.francois@liafa.univ-paris-diderot.fr

2 CNRS, LIAFA, Univ Paris Diderot, Sorbonne Paris-Cité, 75205 Paris, France
frederic.magniez@univ-paris-diderot.fr

Abstract

This work is in the line of designing efficient checkers for testing the reliability of some massive data structures. Given a sequential access to the insert/extract operations on such a structure, one would like to decide, a posteriori only, if it corresponds to the evolution of a reliable structure. In a context of massive data, one would like to minimize both the amount of reliable memory of the checker and the number of passes on the sequence of operations.

Chu, Kannan and McGregor [9] initiated the study of checking priority queues in this setting. They showed that the use of timestamps allows to check a priority queue with a single pass and memory space $\tilde{O}(\sqrt{N})$. Later, Chakrabarti, Cormode, Kondapally and McGregor [7] removed the use of timestamps, and proved that more passes do not help.

We show that, even in the presence of timestamps, more passes do not help, solving an open problem of [9, 7]. On the other hand, we show that a second pass, but in *reverse* direction, shrinks the memory space to $\tilde{O}((\log N)^2)$, extending a phenomenon the first time observed by Magniez, Mathieu and Nayak [15] for checking well-parenthesized expressions.

1998 ACM Subject Classification F.2.0 Analysis of Algorithms and Problem Complexity

Keywords and phrases Streaming Algorithms, Communication Complexity, Priority Queue

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.454

1 Introduction

The reliability of memory is central and becomes challenging when it is massive. In the context of program checking [4] this problem has been addressed by Blum, Evans, Gemmell, Kannan and Naor [3]. They designed on-line checkers that use a small amount of reliable memory to test the behavior of some data structures. Checkers are allowed to be randomized and to err with small error probability. In that case the error probability is not over the inputs but over the random coins of the algorithm.

Chu, Kannan and McGregor [9] revisited this problem for priority queue data structures, where the checker only has to detect an error after processing an entire sequence of data accesses. This can be rephrased as a one-pass streaming recognition problem. Streaming algorithms sequentially scan the whole input piece by piece in one sequential pass, or in a small number of passes, while using sublinear memory space. In our context, the stream is defined by the sequence of insertions and extractions on the priority queue. Using a streaming

* Full version available on http://arxiv.org/abs/1209.4971

† Supported by the French ANR Defis program under contract ANR-08-EMER-012 (QRAC project) and by the French ANR Blanc program under contract ANR-12-BS02-005 (RDAM project)

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE algorithm, the objective is then to decide if the stream corresponds to a correct implementation of a priority queue. We also consider collection data structures that implement multisets.

▶ **Definition 1** (Collection,PQ). Let Σ_0 be some alphabet. Let $\Sigma = \{\texttt{ins}(a), \texttt{ext}(a) : a \in \Sigma_0\}$. For $w \in \Sigma^N$, define inductively multisets M_i by $M_0 = \emptyset$, $M_i = M_{i-1} \setminus \{a\}$ if $w[i] = \texttt{ext}(a)$, and $M_i = M_{i-1} \cup \{a\}$ if $w[i] = \texttt{ins}(a)$.
Then $w \in \textsc{Collection}(\Sigma_0)$ if and only if $M_n = \emptyset$ and $a \in M_{i-1}$ when $w[i] = \texttt{ext}(a)$, for $i = 1, \ldots, N$. Moreover, $w \in \mathrm{PQ}(U)$, for $U \in \mathbb{N}$, if and only if $w \in \textsc{Collection}(\{0, 1, \ldots, U\})$ and $a = \max(M_{i-1})$ when $w[i] = \texttt{ext}(a)$, for $i = 1, \ldots, N$.

Streaming algorithms were initially designed with a single pass: when a piece of the stream has been read, it is gone for ever. This makes those algorithms of practical interest for online context, such as network monitoring, for which first streaming algorithms were developed [1]. Motivated by the explosion in the size of the data that algorithms are called upon to process in everyday real-time applications, the area of streaming algorithms has experienced tremendous growth over the last decade in many applications. In particular, a streaming algorithm can model an external read-only memory. Examples of such applications occur in bioinformatics for genome decoding, or in Web databases for the search of documents. In that context, considering multi-pass streaming algorithm is relevant.

Using standard arguments one can establish that every p-pass randomized streaming algorithm needs memory space $\Omega(N/p)$ for recognizing Collection. Nonetheless, Chakrabarti, Cormode, Kondapally and McGregor [7] gave a one-pass randomized algorithm for PQ using memory space $\tilde{O}(\sqrt{N})$. They also showed that several passes do not help, since any p-pass randomized algorithm would require memory space $\Omega(\sqrt{N}/p)$. A similar lower bound was showed independently, but using different tools, by Jain and Nayak [10]. The case of a single pass was established previously by Magniez, Mathieu and Nayak [15] for checking the well-formedness of parenthesis expressions, or equivalently the behavior of a stack.

A simpler variant of PQ with timestamps was in fact first studied by Chu, Kannan and McGregor [9], where now each item is inserted to the queue with its index.

▶ **Definition 2** (PQ-TS). Let $\Sigma = \{\texttt{ins}(a), \texttt{ext}(a) : a \in \{0, 1, \ldots, U\}\} \times \mathbb{N}$. Let $w \in \Sigma^N$. Then $w \in \text{PQ-TS}(U)$ if and only if $w \in \textsc{Collection}(\Sigma)$, $w[1, \ldots, N][1] \in \mathrm{PQ}(U)$, and $w[i][2] = i$ when $w[i][1] = \texttt{ins}(a)$.

Nonetheless the two works [9, 7] left open problems. The lower bound of [7] was proved only for PQ, and no significant lower bounds for PQ-TS established. Moreover, the streaming complexity of PQ for algorithms processing the stream in both directions was not studied.

Even though recognizing PQ-TS is obviously easier than recognizing PQ, our first contribution (Section 3) consists in showing that they both obey the same limitation, even with multiple passes in the same direction.

▶ **Theorem 3.** *Every p-pass randomized streaming algorithm recognizing* PQ-TS$(3N/2)$ *with bounded error* $1/3$ *requires memory space* $\Omega(\sqrt{N}/p)$ *for inputs of length* N.

As a consequence, since this lower bound uses very restricted hard instances, it models most of possible variations. For instance, assuming that the input is in Collection and has no duplicates is not sufficient to guarantee a faster algorithm. Theorem 3 is proved by introducing a related communication problem with $\Theta(\sqrt{N})$ players. Then we reduce the number of players to 3, and prove a lower bound on the information carried by players, leading to the desired lower bound. We are following the *information cost* approach taken in [8, 17, 2, 12, 11], among other works. Recently, the information cost appeared as one of the most central notion in communication complexity [6, 5, 13]. The information cost of a protocol is the amount of information that messages carry about players' inputs. We adapt this notion to suit both the nature of streaming algorithms and of our problem.

Even if our result suggests that allowing multiple passes does not help, one could also consider the case of bidirectional passes. We believe that it is a natural relaxation of multi-pass streaming algorithms where the stream models some external read-only memory. In that case, we show that a second pass, but in reverse order, makes the problem of checking PQ easy, even with no timestamps (Section 4). A similar phenomenon has been established previously in [15] for checking the well-formedness of parenthesis expressions. Their problem is simpler than ours, and therefore our algorithm is more general.

▶ **Theorem 4.** *There is a bidirectional* 2*-pass randomized streaming algorithm recognizing* $\mathrm{PQ}(U)$ *with memory space* $\mathrm{O}((\log N)(\log U + \log N))$*, time per processing item* $\mathrm{polylog}(N, U)$*, and one-sided bounded error* N^{-c}*, for inputs of length* N *and any constant* $c > 0$*.*

Our algorithm uses a hierarchical data structure similar to the one introduced in [15] for checking well-parenthesized expressions. At high level, it also behaves similarly. It performs one pass in each direction and makes an on-line compression of past information in at most $\log N$ hashcodes. While this compression can lose information, the compression technique ensures that a mistake is always detected in one of the two directions. Nonetheless our algorithm differs on two main points. First, unlike parenthesized expressions, PQ is not symmetric. Therefore one has to design an algorithm for each pass. Second, the one-pass algorithm for PQ [7] is technically more advanced than the one of [15]. Thus designing a bidirectional 2-pass algorithm for PQ is more challenging.

Theorems 3 and 4 point out a strange situation but not isolated at all. Languages studied in [9, 15, 7, 14] and in this paper have space complexity $\Theta(\sqrt{N}\mathrm{polylog}(N))$ for a single pass, $\Omega(\sqrt{N}/p)$ for p passes in the same direction, and $\mathrm{polylog}(N)$ for 2 passes but one in each direction. We hope this paper makes progress in the study of that phenomenon.

2 Preliminaries

In streaming algorithms (see [16] for an introduction), a *pass* on an input $w \in \Sigma^N$, for some alphabet Σ, means that w is given as an *input stream* $w[1], w[2], \ldots, w[N]$, which arrives sequentially, i.e., letter by letter in this order. For simplicity, we assume throughout this article that the input length N is always given to the algorithm in advance. Nonetheless, all our algorithms can be adapted to the case in which N is unknown until the end of a pass.

▶ **Definition 5** (Streaming algorithm)**.** A p-pass randomized *streaming algorithm* with space $s(N)$ and time $t(N)$ is a randomized algorithm that, given $w \in \Sigma^N$ as an input stream,
- performs p sequential passes on w;
- maintains a memory space of size at most $s(N)$ bits while reading w;
- has running time at most $t(N)$ per processed letter $w[i]$;
- has preprocessing and postprocessing time at most $t(N)$.

The algorithm is *bidirectional* if it is allowed to access to the input in the reverse order, after reaching the end of the input. Then p is the total number of passes in either direction.

The proof of our lower bound uses the language of communication complexity with multi-players, and is based on information theory arguments. We consider *number-in-hand* and *message-passing* communication protocols. Each player is given some input, and can communicate with another player according to the rules of the protocol. Our players are embedded into a directed circle, so that each player can receive (resp. transmit) a message from its unique predecessor (resp. successor). Each player send a message after receiving one, until the end of the protocol is reached. Players have no space and time restriction. Only the number of rounds and the size of messages are constrained.

Consider a randomized multi-player communication protocol P. We consider only two types of random source, that we call *coins*. Each player has access to its own independent source of *private coins*. In addition, all players share another common source of *public coins*. The output of P is announced by the last player. This is therefore the last message of the last player. We say that P is with bounded error ϵ when P errs with probability at most ε over the private and public coins. The *transcript* Π of P is the concatenation of all messages sent by all players, including all public coins. In particular, it contains the output of P, since it is given by the last player. Given a subset S of players, we let Π_S be the concatenation of all messages sent by players in S, including again all public coins.

We now remind the usual notions of entropy H and mutual information I. Let X, Y, Z be random variables. Then $\mathrm{H}(X) = -\mathbb{E}_{x\leftarrow X}\log\Pr(X = x)$, $\mathrm{H}(X|Y = y) = -\mathbb{E}_{y\leftarrow Y}\log\Pr(X = x|Y = y)$, $\mathrm{H}(X|Y) = \mathbb{E}_{y\leftarrow Y}\mathrm{H}(X|Y = y)$, and $\mathrm{I}(X : Y|Z) = \mathrm{H}(X|Z) - \mathrm{H}(X|Y, Z)$. The entropy and the mutual information are non negative and satisfy $\mathrm{I}(X : Y|Z) = \mathrm{I}(Y : X|Z)$.

The mutual information between two random variables is connected to the Hellinger distance h between their respective distribution probabilities. Given a random variable X we also denote by X its underlying distribution.

▶ **Proposition 6** (Average encoding). *Let X, Y be random variables. Then $\mathbb{E}_{y\leftarrow Y}\,\mathrm{h}^2(X|_{Y=y}, X) \le \kappa\mathrm{I}(X : Y)$, where $\kappa = \frac{\ln 2}{2}$.*

The Hellinger distance also generalizes the cut-and-paste lemma to randomized protocols.

▶ **Proposition 7** (Cut and paste). *Let P be a 2-player randomized protocol. Let $\Pi(x, y)$ denote the random variable representing the transcript in P when Players A, B have resp. inputs x, y. Then $\mathrm{h}(\Pi(x, y), \Pi(u, v)) = \mathrm{h}(\Pi(x, v), \Pi(u, y))$, for all pairs (x, y) and (u, v).*

Last we use that the square of the Hellinger distance is convex, and the following connection to the more convention ℓ_1-distance: $\mathrm{h}(X, Y)^2 \le \frac{1}{2}\|X - Y\|_1 \le \sqrt{2}\mathrm{h}(X, Y)$. For a reference on these results, see [10].

3 Lower bound for PQ-TS

The proof of our lower bound consists in first translating it into a $3m$-player communication problem, for some large m; then reducing the number of players to 3 using the information cost approach; and last studying the base case of 3 players using information theory arguments.

3.1 From streaming algorithms to communication protocols

In this section, we write a instead of `ins`(a) and $\bar{a}$ instead of `ext`(a). Consider the following set of hard instances of size $N = (2n + 2)m$:

Raindrops(m, n) (see LHS of Figure 1)

- For $i = 1, 2, \dots, m$, repeat the following motif:
  - For $j = 1, 2, \dots, n$, insert either $v_{i,j} = 3(ni - j)$ or $v_{i,j} = 3(ni - j) + 2$
  - Insert either $a_i = 3(ni-(k_i-1))+1$ or $a_i = 3(ni-k_i)+1$, for some $k_i \in \{2, \dots, n\}$
  - Extract $v_{i,1}, v_{i,2}, \dots, v_{i,k_i-1}, a_i$ in decreasing order
- Extract everything left in decreasing order

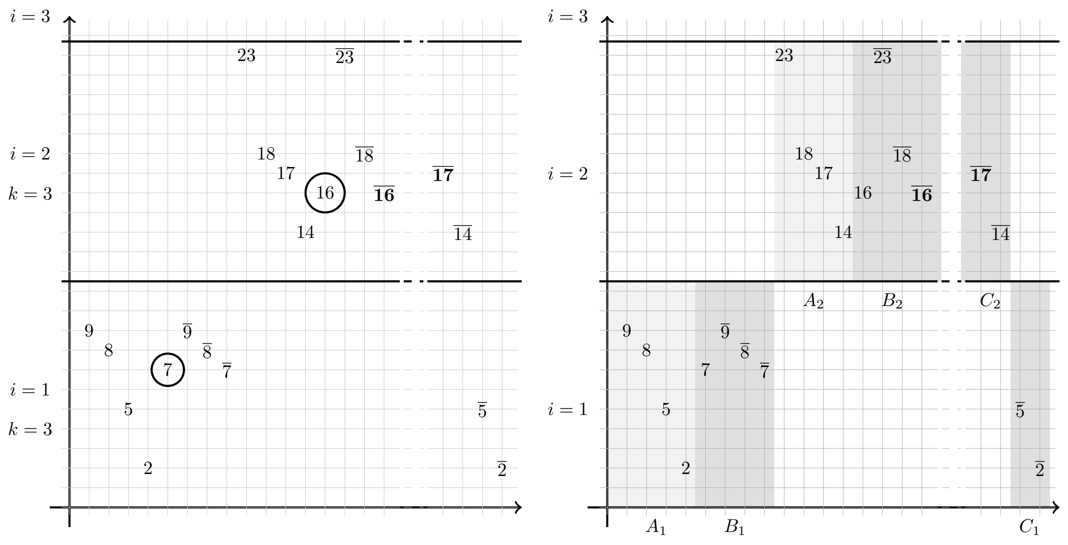


Figure 1 Left: Instance of RAINDROPS$(m, 4)$ with one error: 17 is extracted after 16. Insertions a_i are circled. Right: Cutting RAINDROPS$(m, 4)$ into $3m$ pieces to make it a communication problem. Players' input are within each corresponding region.

Observe that such an instance is in COLLECTION. One can compute the timestamps for each value by maintaining only $O(\log N)$ additionnal bits. Last, there is only one potential error in each motif that can make it outside of PQ-TS. Indeed, $v_{i,1}, v_{i,2}, \ldots, v_{i,k_i-1}, a_i$ are in decreasing order up to a switch between a_i and v_{i,k_i-1}.

Given such an instance as a stream, an algorithm for PQ-TS must decide if an error occurs between $\overline{a_i}$ and $\overline{v_{i,k_i}}$, for some i. Intuitively, if the memory space is less than εn, for a small enough constant $\varepsilon > 0$, then the algorithm cannot remember all the values $(v_{i,j})_j$ when a_i is extracted, and therefore cannot check a potential error with a_i. The next opportunity is during the last sequence of extractions. But then, the algorithm has to remember all values $(a_i)_i$, which is again impossible if the memory space is less than εm.

In order to formalize this intuition, Lemma 8 first translates our problem into a communication one between $3m$ players in the same way as [15], as shown on the RHS of Figure 1. Then we analyze its complexity using information theory arguments in Section 3.2.

Any insertion and extraction of an instance in RAINDROPS(m, n) can be described by its index and a single bit. Let $x_i[j] \in \{0, 1\}$ such that $v_{i,j} = 3(ni - j) + 2x_i[j]$. Similarly, let $d_i \in \{0, 1\}$ such that $a_i = 3(ni - k_i) + 1 + 3d_i$. For simplicity, we write $\mathbf{x}$ instead of $(x_i)_{1 \le i \le m}$. Similarly, we use the notations $\mathbf{k}$ and $\mathbf{d}$. Then our related communication problem is:

WEAKINDEX(m, n)

- Input for players $(A_i, B_i, C_i)_{1 \le i \le m}$:
  - Player A_i has a sequence $x_i \in \{0, 1\}^n$
  - Player B_i has $x_i[1, k_i - 1]$, with $k_i \in \{2, \ldots, n\}$ and $d_i \in \{0, 1\}$
  - Player C_i has $x_i[k_i, n]$
- Output: $f_m(\mathbf{x}, \mathbf{k}, \mathbf{d}) = \bigvee_{i=1}^{m} f(x_i, k_i, d_i)$, where $f(x, k, d) = [(d = 0) \wedge (x[k] = 1)]$
- Communication settings:
  - One round: each player sends a message to the next player according to the diagram $A_1 \to B_1 \to A_2 \to \cdots \to B_m \to C_m \to C_{m-1} \to \cdots \to C_1$.
  - Multiple rounds: If there is at least one round left, C_1 sends a message to A_1, and then players continue with the next round.

▶ **Lemma 8.** *Assume there is a p-pass randomized streaming algorithm for deciding if an instance of* Raindrops(n,m) *is in* PQ-TS$(3mn)$ *with memory space $s(m,n)$ and bounded error ε. Then there is a p-round randomized protocol for* WeakIndex(n,m) *with bounded error ε such that each message has size at most $s(m,n)$.*

We are now ready to give the structure of the proof of Theorem 3, which has techniques based on information theory. Define the following collapsing distribution μ_0 of hard inputs (x,k,d), encoding instances of Raindrops$(1,n)$, where f always takes value 0. Distribution μ_0 is such that (x,k) is uniform on $\{0,1\}^n \times \{2,\dots,n\}$ and, given x,k, the bit $d \in \{0,1\}$ is uniform if $x[k]=0$, and $d=1$ if $x[k]=1$. From now on, (X,K,D) are random variables distributed according to μ_0, and (x,k,d) denote any of their values.

Then the proof of Theorem 3 consists in studying the information cost of any communication protocol for WeakIndex(n,m), which is a lower bound on its communication complexity. Using that μ_0 is collapsing for f, Lemma 9 establishes a direct sum on the information cost of WeakIndex(n,m). Then, even if f is constant on μ_0, Lemma 12 lower bounds the information cost of a single instance of WeakIndex$(n,1)$.

Proof of Theorem 3. Let n,N be positive integers such that $N=(2n+2)n$. Assume that there exists a p-pass randomized algorithm that recognizes PQ-TS$(3N/2)$, with memory space αn and bounded error ε, for inputs of size N. Then, by Lemma 8, there a p-round randomized protocol P for WeakIndex(n,n) such that each message has size at most αn. By Lemma 9, one can derive from P another $(p+1)$-round randomized protocol P' for WeakIndex$(n,1)$ with bounded error ε, and transcript Π' satisfying $|\Pi'| \leq 3(t+1)\alpha n$ and $\max\{\mathrm{I}(D:\Pi'_B|X,K),\mathrm{I}(K,D:\Pi'_C|X)\} \leq (p+1)\alpha$. Then by Lemma 12, $3(p+1)\alpha \geq (1-2\varepsilon)/10$, that is $\alpha = \mathrm{O}(1/p)$, concluding the proof. ◀

3.2 Communication complexity lower bound

We first reduce the general problem WeakIndex(n,m) with $3m$ players to a single instance of WeakIndex$(n,1)$ with 3 players. In order to do so we exploit the direct sum property of the information cost. The use of a collapsing distribution where f is always 0 is crucial.

▶ **Lemma 9.** *If there is a p-round randomized protocol P for* WeakIndex(n,m) *with bounded error ε and messages of size at most $s(m,n)$, then there is a $(p+1)$-round randomized protocol P' for* WeakIndex$(n,1)$ *with bounded error ϵ, and transcript P' satisfying $|\Pi'| \leq 3(p+1)s(m,n)$ and $\max\{\mathrm{I}(D:\Pi'_B|X,K),\mathrm{I}(K,D:\Pi'_C|X)\} \leq \frac{p+1}{m}s(m,n)$.*

Sketch of proof. Given a protocol P, we show how to construct another protocol P' for any instance (x,k,d) of WeakIndex$(n,1)$. In order to avoid any confusion, we denote by A, B and C the three players of P', and by $(A_i,B_i,C_i)_i$ the ones of P.

Protocol P'

- Using public coins, all players generate uniformly at random $j \in \{1,\dots,m\}$, and $x_i \in \{0,1\}^n$ for $i \neq j$
- Players A, B and C set respectively their inputs to the ones of A_j, B_j, C_j
- For all $i > j$, Player B generates, using its private coins, uniformly at random $k_i \in \{2,\dots,n\}$, and then it generates uniformly at random d_i such that $f(x_i,k_i,d_i)=0$
- For all $i < j$, Player C generates, using its private coins, uniformly at random $k_i \in \{2,\dots,n\}$, and then it generates uniformly at random d_i such that $f(x_i,k_i,d_i)=0$
- Players A, B and C run P as follows. A simulates A_j only, B simulates B_j and $(A_i,B_i,C_i)_{i>j}$, and C simulates C_j and $(A_i,B_i,C_i)_{i<j}$.

Observe that A starts the protocol if $j = 1$, and C starts otherwise. Moreover C stops the simulation after p rounds if $j = 1$, and after $p+1$ rounds otherwise. For all $i \neq j$, entries are generated such that $f(x_i, k_i, a_i) = 0$, therefore $f_m(\mathbf{X}, \mathbf{k}, \mathbf{d}) = f(x_j, k_j, a_j) = f(x, k, a)$, and P' has the same bounded error than P.

By applying the chain rule, one can see that P' satisfies the required conditions of the lemma. ◀

We now prove a trade-off between the bounded error of a protocol for a single instance of WeakIndex$(n, 1)$ and its information cost. The proof involves some of the tools of [10] but with some additional obstacles to apply them. The inherent difficulty is due to that we have 3 players whereas the cute-and-paste property applies to 2-player protocols. Therefore we have to group 2 players together.

Given some parameters (x, k, a) for an input of WeakIndex$(n, 1)$, we denote by $\Pi(x, k, a)$ the random variable describing the transcript Π of our protocol. We start by two lemmas exploiting the average encoding theorem (proofs omitted).

▶ **Lemma 10.** *Let P be a randomized protocol for* WeakIndex$(n, 1)$ *with transcript* Π *satisfying* $|\Pi| \leq \alpha n$ *and* $\mathrm{I}(K, D : \Pi_C | X) \leq \alpha$. *Then*

$$\mathop{\mathbb{E}}_{x[1,l-1],l} \mathrm{h}^2(\Pi(x[1, l-1]0X[l+1, n], l, 1), \Pi(x[1, l-1]1X[l+1, n], l, 1)) \leq 28\alpha,$$

where $l \in [\frac{n}{2} + 1, n]$ *and* $x[1, l-1]$ *are uniformly distributed.*

▶ **Lemma 11.** *Let P be a randomized protocol for* WeakIndex$(n, 1)$ *with transcript* Π *satisfying* $\mathrm{I}(D : \Pi_B | X, K) \leq \alpha$. *Then*

$$\mathop{\mathbb{E}}_{x[1,l-1],l} \mathrm{h}^2(\Pi(x[1, l-1]0X[l+1, n], l, 0), \Pi(x[1, l-1]0X[l+1, n], l, 1)) \leq 12\alpha,$$

where $l \in [\frac{n}{2} + 1, n]$ *and* $x[1, l-1]$ *are uniformly distributed.*

We now end with the main lemma which combines both previous ones and applies the cut-and-paste property, where Players A, C are grouped.

▶ **Lemma 12.** *Let P be a randomized protocol for* WeakIndex$(n, 1)$ *with bounded error* ϵ, *and transcript* Π *satisfying* $|\Pi| \leq \alpha n$ *and* $\max\{I(D : \Pi_B | X, K), I(K, D : \Pi_C | X)\} \leq \alpha$. *Then* $\alpha \geq (1 - 2\varepsilon)/10$.

Proof. Let L be a uniform integer random variable in $[\frac{n}{2} + 1, n]$. Remind that we enforce the output of P to be part of Π. Therefore, any player, and in particular B, can compute f with bounded error ε given Π. Since $f(x[1, l-1]0X[l+1, n], l, 0) = 0$ and $f(x[1, l-1]1X[l+1, n], l, 1) = 1$, the error parameter ε must satisfies

$$\mathop{\mathbb{E}}_{x[1,l-1],l} \|\Pi(x[1, l-1]0X[l+1, n], l, 0) - \Pi(x[1, l-1]1X[l+1, n], l, 0)\|_1 \geq 2(1 - 2\varepsilon).$$

The rest of the proof consists in upper bounding the LHS by 19α.

Applying the triangle inequality and that $(u + v)^2 \leq 2(u^2 + v^2)$ on the inequalities of Lemmas 10 and 11 gives

$$\mathop{\mathbb{E}}_{x[1,l-1],l} \mathrm{h}^2(\Pi(x[1, l-1]0X[l+1, n], l, 0), \Pi(x[1, l-1]1X[l+1, n], l, 1)) \leq 30\alpha.$$

We then apply the cut-and-paste property by considering (A, C) as a single player with transcript $\Pi_{A,C}$. Therefore

$$\mathop{\mathbb{E}}_{x[1,l-1],l} \mathrm{h}^2(\Pi(x[1, l-1]0X[l+1, n], l, 1), \Pi(x[1, l-1]1X[l+1, n], l, 0)) \leq 30\alpha.$$

Combining again with the inequality from Lemma 11 gives

$$\mathop{\mathbb{E}}_{x[1,l-1],l} \mathrm{h}^2(\Pi(x[1,l-1]0X[l+1,n],l,0),\Pi(x[1,l-1]1X[l+1,n],l,0)) \leq 42\alpha.$$

Last, we get the requested upper bound by using the connexion between the Hellinger distance and the ℓ_1-distance, and the convexity of the square function. ◀

4 Bidirectional streaming algorithm for PQ

Remember that in this section our stream is given without any timestamps. Therefore we consider in this section only streams w of $\texttt{ins}(a), \texttt{ext}(a)$, where $a \in [0, U]$. For the sake of clarity, we assume for now that the stream has no duplicate. Our algorithms can be extended to the general case, but the technical difficulties shadow the main ideas.

Up to padding we can assume that N is a power of 2: we append a sequence of $\texttt{ins}(a)\texttt{ext}(a)\texttt{ins}(a+1)\texttt{ext}(a+1)\dots$ of suitable length, where a is large enough so that there is no duplicate (assuming that w is of even size, otherwise $w \notin \mathrm{PQ}(U)$). We use $\mathrm{O}(\log N)$ bits of memory to store, after the first pass, the number of letters padded.

We use a hash function based on the one used by the Karp-Rabin algorithm for pattern matching. For all this section, let p be a prime number in $\{\max(2U+1, N^{c+1}), \dots, 2\max(2U+1, N^{c+1})\}$, for some fixed constant $c \geq 1$. Since our hash function is linear we only define it for single insertion/extraction as

$$\mathrm{hash}(\texttt{ins}(a)) = \alpha^a \mod p, \quad \text{and} \quad \mathrm{hash}(\texttt{ext}(a)) = -\alpha^a \mod p,$$

where α is a randomly chosen integer in $[0, p-1]$. This is the unique source of randomness of our algorithm. A hashcode h *encodes* a sequence w if $h = \mathrm{hash}(w)$ as a formal polynomial in α. In that case we say that h *includes* $w[i]$, for all i. Moreover w is *balanced* if the same integers have been inserted and extracted. In that case it must be that $h = 0$. We also say that h is balanced it it encodes a balanced sequence w. The converse is also true with high probability by the Schwartz-Zippel lemma.

▶ **Fact 13.** *Let w be some unbalanced sequence. Then* $\Pr(\mathrm{hash}(w) = 0) \leq \frac{N}{p} \leq \frac{1}{N^c}$.

The forward-pass algorithm was introduced in [7], but the reverse-pass one is even simpler. As a warming up, we start by introducing the later algorithm. In order to keep it simple to understand, we do not optimize it fully. Last define the instruction $\texttt{Update}(h, v)$ that returns $(h + \mathrm{hash}(v) \mod p)$ *and* updates h to that value.

4.1 One-reverse-pass algorithm for PQ

Our reverse-pass algorithm decomposes the stream w into blocks. We call a valley an extraction $w[t] = \texttt{ext}(a)$ with $w[t+1] = \texttt{ins}(b)$. A new block starts at each valley. To the i-th block we associate a hashcode h_i and an integer m_i. Hashcode h_i encodes all extractions within the block and matching insertions. Integer m_i is the minimum of extractions in the block. With the values $(m_i)_i$, one can encode insertions in the correct h_i if $w \in \mathrm{PQ}$. Observe that we use index notations for block indices and bracket notations for stream positions.

Algorithm 1 uses memory space $\mathrm{O}(r)$, where r is the number of valleys in w. We could make it run with memory space $\mathrm{O}(\sqrt{N \log N})$ by reducing the number of valleys as in [7]. We do not need to as we use another compression in the two-pass algorithm.

We first state a crucial property of Algorithm 1, and then show that it satisfies Theorem 15, when there is no duplicate. We remind that we process the stream from right to left.

Algorithm 1 One-reverse-pass algorithm for PQ

```
$m_0 \leftarrow -\infty$; $h_0 \leftarrow 0$; $t \leftarrow N$; $i \leftarrow 0$ // $i$ is called the block index
While $t > 0$
    If $w[t] = \texttt{ins}(a)$
        $k \leftarrow \max\{j \leq i : m_j \leq a\}$; //Compute the hashcode index of $a$
        Update($h_k, w[t]$)
    Else $w[t] = \texttt{ext}(a)$
        If $w[t+1] = \texttt{ins}(b)$ //This is a valley. We start a new block
            $i \leftarrow i+1$; $m_i \leftarrow a$; $h_i \leftarrow 0$ //Create a new hashcode
        Else $w[t+1] = \texttt{ext}(b)$
            Check($a \geq b$) //Check that extractions are well-ordered
        Update($h_i, w[t]$)
    $t \leftarrow t-1$
For $j = 0$ to $i$: Check($h_j = 0$) //Check that hashcodes are balanced w.h.p.
Accept // $w$ succeeded to all checks
```

▶ **Lemma 14.** *Consider Algorithm 1 right after processing* $\texttt{ins}(a)$*. Assume that* $\texttt{ext}(a)$ *has been already processed. Let* $h_k, h_{k'}$ *be the respective hashcodes including* $\texttt{ext}(a)$, $\texttt{ins}(a)$*. Then* $k = k'$ *if and only if all* $\texttt{ext}(b)$ *occurring between* $\texttt{ext}(a)$ *and* $\texttt{ins}(a)$ *satisfy* $b > a$*.*

▶ **Theorem 15.** *There is a 1-reverse-pass randomized streaming algorithm for* $\mathrm{PQ}(U)$ *with memory space* $\mathrm{O}(r(\log N + \log U))$ *and one-sided bounded error* N^{-c}*, for inputs of length* N *with* r *valleys, and any constant* $c > 0$*.*

Proof. We show that Algorithm 1 suits the conditions, assuming there is no duplicate. Let $w \in \mathrm{PQ}(U)$. Then w always passes the test at line 10. Moreover, by Lemma 14, each insertion $\texttt{ins}(a)$ is necessarily in the same hashcode than its matching extraction $\texttt{ext}(a)$. Therefore, all hashcodes equal 0 at line 13 since they are balanced. In conclusion, the algorithm accepts w with probability 1.

Assume now that $w \notin \mathrm{PQ}$. First we show that unbalanced w are rejected with high probability, that is at least $1 - N^{-c}$, at line 13, if they are not rejected before. Indeed, since each $w[t]$ is encoded in some h_j, at least one h_j must be unbalanced. Then by Fact 13, the algorithm rejects w.h.p. We end the proof assuming w balanced. We remind that we process the stream from right to left. The two remaining possible errors are: (1) $\texttt{ins}(a)$ is processed before $\texttt{ext}(a)$, for some a; and (2) $\texttt{ext}(a), \texttt{ext}(b), \texttt{ins}(a)$ are processed in this order with $b < a$ and possibly intermediate insertions/extractions. In both cases, we show that some hashcodes are unbalanced at line 13, and therefore fail the test w.h.p by Fact 13, except if the algorithm rejects before.

Consider case (1). Since $\texttt{ins}(a)$ is processed before $\texttt{ext}(a)$, there is at least one valley between $\texttt{ins}(a)$ and $\texttt{ext}(a)$. Therefore $\texttt{ins}(a)$ and $\texttt{ext}(a)$ are encoded into different hashcodes, that are unbalanced at line 13. Consider now case (2). Lemma 14 gives that $\texttt{ext}(a)$ and $\texttt{ins}(a)$ are encoded in different hashcodes, that are again unbalanced at line 13. ◀

4.2 Bidirectional two-pass algorithm

Our algorithm performs one pass in each direction using Algorithms 2 and 2. We use the hierarchical data structure of [15] in order to reduce the number of blocks. A block of size 2^i is of the form $[(q-1)2^i + 1, q2^i]$, for $1 \leq q \leq N/2^i$. Observe that, given two such blocks, either they are disjoint or one is included in the other. We decompose dynamically the letters of w, that have been already processed, into nested blocks of 2^i letters as follows. Each new

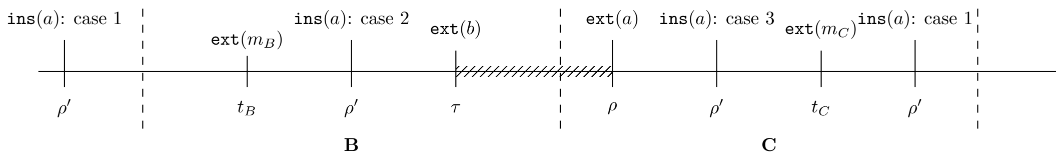


Figure 2 Relative positions of insertions and extractions used in the proof of Theorem 4

Algorithm 2 Pass from left to right

```
S ← [(0,−∞,0)]  // Initialization of S
While stream is not empty
    Read(next letter v on stream)  // See below
    While the 2 topmost elements of S have same block size ℓ
        (h_1, m_1, ℓ) ←Pop(S);  (h_2, m_2, ℓ) ←Pop(S)
        Push(S,(h_1 + h_2 mod p, min(m_1, m_2), 2ℓ))  // Merge of 2 blocks
Check(S = [(0,−∞,0),(0,0,N)])
Return

Function Read(v):
Case v = ins(a)  // When reading an insertion
    Let (h,m,ℓ) be the first item of S from top such that a ≥ m
    Replace (h,m,ℓ) by (Update(h,v),m,ℓ)
    Push (S,(0,+∞,1))
Case v = ext(a)  // When reading an extraction
    For all items (h,m,ℓ) on S such that m > a: Check(h = 0)
    Let (h,m,ℓ) be the first item of S from top such that a > m
    Replace (h,m,ℓ) by (Update(h,v),m,ℓ)
    Push(S,(0,a,1))
```

processed letter of w defines a new block. When two blocks have same size, they merge. All processed blocks are pushed on a stack. Therefore, only the two topmost blocks of the stack may potentially merge. Because the size of each block is a power of 2 and at most two blocks have the same size (before merging), there are at most $\log N + 1$ blocks at any time.

Moreover, since our stream size is a power of 2, all blocks eventually appear in the hierarchical decomposition, whether we read the stream from left to right or from right to left. In fact, if two same-sized blocks appear simultaneously in one decomposition before merging, the same is true in the other decomposition. This point is crucial for our analysis.

Our algorithm uses the following description of a block B: its hashcode h_B, the minimum m_B of its extractions, and its size ℓ_B. For the analysis, let t_B be such that $w[t_B] = \texttt{ext}(m_B)$. Only h_B can change without B being merged with another block. On the pass from right to left, all extractions from the block and matching insertions are included in h_B. On the pass from left to right, insertions are included in the hashcode of the earliest possible block where they could have been, and extractions are included with their matching insertions. The minimums $(m_B)_B$ are used to decide where to include values. Observe the importance of checking $h_B = 0$ during the execution and not at the end, when only one block is left.

When there is some ambiguity, we denote by $h_B^{\rightarrow}$ and $h_B^{\leftarrow}$ the hashcodes for the left-to-right and right-to-left passes. Observe that m_B, t_B, ℓ_B are identical in both directions.

Proof of Theorem 4. We show that execution of both Algorithms 2 and 3 suits the conditions, assuming no duplicates. The space constraints are satisfied because elements of S have size $O(\log N + \log U)$ and S has size $O(\log N)$. The processing time is from inspection.

Algorithm 3 Pass from right to left

```
1  $S \leftarrow []$; // Initialization of $S$
2  While stream is not empty
3      Read(next letter $v$ on stream) // See below
4      While the 2 topmost elements of $S$ have same block size $\ell$
5          $(h_1, m_1, \ell) \leftarrow$Pop($S$); $(h_2, m_2, \ell) \leftarrow$Pop($S$)
6          Push($S$,$(h_1 + h_2 \mod p, \min(m_1, m_2), 2\ell)$) // Merge of 2 blocks
7  Check($S = [(0, 0, N)]$)}
8  Return
9
10 Function Read($v$):
11 Case $v = \texttt{ins}(a)$ // When reading an insertion
12     Let $(h, m, \ell)$ be the first item of $S$ from top such that $a \geq m$
13     Replace $(h, m, \ell)$ by $(\texttt{Update}(h, v), m, \ell)$
14     Push (S, $(0, +\infty, 1)$)
15 Case $v = \texttt{ext}(a)$ // When reading an extraction
16     For all items $(h, m, \ell)$ on $S$ such that $m > a$: Check($h = 0$)
17     Push($S$,$(\text{hash}(v), a, 1)$)
```

As with Theorem 15, inputs in $\text{PQ}(U)$ are accepted with probability 1, and unbalanced inputs are rejected with high probability (at least $1 - N^{-c}$). Let $w \notin \text{PQ}$ be balanced. For ease of notations, let $w[-1] = \texttt{ins}(-\infty)$ and $w[0] = \texttt{ext}(-\infty)$. Then, there are $\tau < \rho$ such that $w[\tau] = \texttt{ext}(b)$, $w[\rho] = \texttt{ext}(a)$, with $a > b$ and $w[t] \neq \texttt{ins}(a)$ for all $\tau < t < \rho$.

Among the pairs (τ, ρ), consider the ones with the smallest ρ. From those, select the one with the smallest b, with $w[\tau] = \texttt{ext}(b)$. Let B, C be the largest possible disjoint blocks such that τ is in B and ρ in C. Then B and C have same size, are contiguous, and appear simultaneously in each direction before they merge. Let ρ' and τ' be such that $w[\rho'] = \texttt{ins}(a)$ and $w[\tau'] = \texttt{ins}(b)$. Then $w[t]$ is an insertion for all $\tau < t < \rho$ by minimality of ρ and b. Indeed if $w[t] = \texttt{ext}(c)$ either $b > c$, contradicting the minimality of b, or $c > b$ and (τ, t), contradicting the minimality of ρ. In particular, $t_C \geq \rho$ and $t_B \leq \tau$. Similarly, $\tau' < \tau$.

We distinguish three cases based on the position ρ' of $\texttt{ins}(a)$ (see Figure 2): $\rho' \notin [t_B, t_C]$, $t_B < \rho' < \tau$, and $\rho < \rho' < t_C$. These cases determine in which hashcode $\texttt{ins}(a)$ is included. We analyze Algorithms 2 and 3 when some letter is processed before blocks potentially merge.

Case 1: $\rho' \notin [t_B, t_C]$. Then $h^{\rightarrow}_B$ and $h^{\leftarrow}_C$ are unbalanced respectively when $w[t_C]$ and $w[t_B]$ are processed; therefore w.h.p. Algorithm 2 detects $h^{\rightarrow}_B \neq 0$ or Algorithm 3 detects $h^{\leftarrow}_C \neq 0$, depending on whether $m_B > m_C$. The full proof is omitted because of space constraints.

Case 2: $t_B < \rho' < \tau$. We show that when Algorithm 3 processes $w[t_B] = \texttt{ext}(m_B)$, it checks $h^{\leftarrow}_D = 0$ at line 16 for some $h^{\leftarrow}_D$ including $\texttt{ins}(a)$ but not $\texttt{ext}(a)$. Thus it rejects w.h.p.

When $w[\rho'] = \texttt{ins}(a)$ is processed on the right-to-left pass, $\tau \in B_1$ with B_1 a block in the stack. $\tau \in B$, therefore B_1 intersects B. Because $B_1 \not\subseteq B$, we have $B_1 \subseteq B$. Because $w[\tau] = \texttt{ext}(b)$, we have $a > b \geq m_{B_1}$, and block B_1 is eligible at line 12 of Algorithm 3, meaning that $w[\rho'] = \texttt{ins}(a)$ is included in either $h^{\leftarrow}_{B_1}$ or a more recent hashcode $h^{\leftarrow}_{B_2}$. Since $\rho' \in B$, again $B_2 \subseteq B$. Last, when Algorithm 3 processes $w[t_B] = \texttt{ext}(m_B)$, since we are still within B, some hashcode h_{B_3}, with $B_3 \subseteq B$, includes $w[\rho']$. Moreover, $h^{\leftarrow}_{B_3}$ does not include $w[\rho] = \texttt{ext}(a)$ since $\rho \in C$ and C comes before B. Last, $m_{B_3} > m_B$, by definition of m_B. Hence, Algorithm 3 checks $h^{\leftarrow}_{B_3} = 0$ at line 16 when processing $w[t_B]$. B_3 satisfies the conditions for D when $w[t_B]$ is processed, and Algorithm 3 rejects w.h.p.

Case 3: $\rho < \rho' < t_C$. The proof is the same as case 2, replacing τ, B, B_1, etc.. with ρ, C, C_1, etc... and Algorithm 2 with Algorithm 3. Note that we only have $a \geq m_{C_1}$ this time. ◀

4.3 Generalization when duplicates occur

We maintain two additional parameters δ_B and C_B for each block B. The difference between the number of insertions and extractions included in h_B is stored in δ_B. Whenever $\delta_B = 0$, we check $h_B = 0$. The number of unmatched occurrences of $\texttt{ins}(m_B)$ for the left-to-right pass (resp. $\texttt{ext}(m_B)$ for the right-to-left pass) is stored in C_B. We can then appropriately determine whether each $\texttt{ext}(m_B)$ (resp. $\texttt{ins}(m_B)$) should be included in h_B.

The inequality at line 12 of Algorithm 2 has to become strict instead of large, which the proof of case 3 of the theorem longer and breaks the symmetry.

References

1 N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

2 Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *Journal of Computer and System Sciences*, 68(4):702–732, 2004.

3 M. Blum, W. S. Evans, P. Gemmell, S. Kannan, and M. Naor. Checking the correctness of memories. *Algorithmica*, 12(2):225–244, 1994.

4 M. Blum and S. Kannan. Designing programs that check their work. *Journal of the ACM*, 42(1):269–291, 1995.

5 M. Braverman. Interactive information complexity. In *Proc. of ACM Symp. on Theory of Computing*, pages 505–524, 2012.

6 M. Braverman and A. Rao. Information equals amortized communication. In 748-757, editor, *Proc. of IEEE Symp. on Foundations of Computer Science*, 2011.

7 A. Chakrabarti, G. Cormode, R. Kondapally, and A. McGregor. Information cost tradeoffs for augmented index and streaming language recognition. In *Proc. of IEEE Symp. on Foundations of Computer Science*, pages 387–396, 2010.

8 A. Chakrabarti, Y. Shi, A. Wirth, and A. C.-C. Yao. Informational complexity and the direct sum problem for simultaneous message complexity. In *Proc. of IEEE Symp. on Foundations of Computer Science*, pages 270–278, 2001.

9 M. Chu, S. Kannan, and A. McGregor. Checking and spot-checking the correctness of priority queues. In *Proc. of Int. Colloquium on Automata, Languages and Programming*, pages 728–739, 2007.

10 R. Jain and A. Nayak. The space complexity of recognizing well-parenthesized expressions in the streaming model: the index function revisited, 2010. ECCC Tech. Rep. TR10-071.

11 R. Jain, J. Radhakrishnan, and P. Sen. A lower bound for the bounded round quantum communication complexity of Set Disjointness. In *Proc. of IEEE Symp. on Foundations of Computer Science*, pages 220–229, 2003.

12 T. S. Jayram, Ravi Kumar, and D.Sivakumar. Two applications of information complexity. In *Proc. of ACM Symp. on Theory of Computing*, pages 673–682, 2003.

13 I. Kerenidis, S. Laplante, V. Lerays, J. Roland, and D. Xiao. Lower bounds on information complexity via zero-communication protocols and applications. In *Proc. of IEEE Symp. on Foundations of Computer Science*, 2012. To appear.

14 C. Konrad and F. Magniez. Validating XML documents in the streaming model with external memory. In *Proc. of Int. Conf. on Database Theory*, pages 34–45, 2012.

15 F. Magniez, C. Mathieu, and A. Nayak. Recognizing well-parenthesized expressions in the streaming model. In *Proc. of ACM Symp. on Theory of Computing*, pages 261–270, 2010.

16 S. Muthukrishnan. *Data Streams: Algorithms and Applications.* Now Publishers Inc., 2005.

17 M. Saks and X. Sun. Space lower bounds for distance approximation in the data stream model. In *Proc. of ACM Symp. on Theory of Computing*, pages 360–369, 2002.

Deterministic algorithms for skewed matrix products*

Konstantin Kutzkov

IT University of Copenhagen
Denmark
`konk@itu.dk`

Abstract

Recently, Pagh presented a randomized approximation algorithm for the multiplication of real-valued matrices building upon work for detecting the most frequent items in data streams. We continue this line of research and present new *deterministic* matrix multiplication algorithms.

Motivated by applications in data mining, we first consider the case of real-valued, nonnegative n-by-n input matrices A and B, and show how to obtain a deterministic approximation of the weights of individual entries, as well as the entrywise p-norm, of the product AB. The algorithm is simple, space efficient and runs in one pass over the input matrices. For a user defined $b \in (0, n^2)$ the algorithm runs in time $O(nb+n\cdot\text{Sort}(n))$ and space $O(n+b)$ and returns an approximation of the entries of AB within an additive factor of $\|AB\|_{E1}/b$, where $\|C\|_{E1} = \sum_{i,j} |C_{ij}|$ is the entrywise 1-norm of a matrix C and Sort(n) is the time required to sort n real numbers in linear space. Building upon a result by Berinde et al. we show that for skewed matrix products (a common situation in many real-life applications) the algorithm is more efficient and achieves better approximation guarantees than previously known randomized algorithms.

When the input matrices are not restricted to nonnegative entries, we present a new deterministic group testing algorithm detecting nonzero entries in the matrix product with large absolute value. The algorithm is clearly outperformed by randomized matrix multiplication algorithms, but as a byproduct we obtain the first $O(n^{2+\varepsilon})$-time deterministic algorithm for matrix products with $O(\sqrt{n})$ nonzero entries.

1998 ACM Subject Classification F.2.0 Analysis of algorithms and problem complexity

Keywords and phrases approximate deterministic memory-efficient matrix multiplication

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.466

1 Introduction

The complexity of matrix multiplication is one of the fundamental problems in theoretical computer science. Since Strassen's sub-cubic algorithm for matrix multiplication over a ring from the late 1960's [33], the topic has received considerable attention, see [9] for a historical overview on the subject. It is conjectured that matrix multiplication admits an algorithm running in time $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$. For more than 20 years the record holder was the algorithm by Coppersmith and Winograd [14] running in time $O(n^{2.376})$. Recently two results improving on [14] were announced. In his PhD thesis Stothers [32] presents a refinement of the Coppersmith-Winograd algorithm running in time $O(n^{2.3737})$ and Vassilevska Williams [35] developes a general framework for obtaining a tighter upper bound on the complexity of the Coppersmith-Winograd algorithm. The latter yields the

* Work supported by the Danish National Research Council under the Sapere Aude program.

best known bound of $O(n^{2.3727})$. Several algorithms computing exactly the matrix product for special classes have been designed. For example, algorithms with running time better than $O(n^{2.3727})$ are known for Boolean matrix multiplication with sparse output [25] or for the case when the input or output matrices are sparse [2, 34]. In a recent work Iwen and Spencer [23] present a new class of matrices whose product can be computed in time $O(n^{2+\varepsilon})$ by a deterministic algorithm: namely when the output matrix is guaranteed to contain at most $n^{0.29462}$ non-zero entries in each column (or by symmetry row). All improved algorithms use as a black-box the algebraic matrix multiplication algorithm which, unfortunately, is only of theoretical importance. It uses sophisticated algebraic approaches resulting in large constants hidden in the big-Oh notation and does not admit an efficient implementation. This motivates the need of simple “combinatorial-like" algorithms.

Approximate matrix multiplication. The first approximation algorithm with rigorously understood complexity by Cohen and Lewis is based on sampling [13]. For input matrices with nonnegative entries they show a concentration around the estimate for individual entries in the product matrix with high probability.

The amount of data to be handled has been growing at a faster rate than the available memory in modern computers. Algorithms for massive data sets, where only sequential access to the input is allowed, have become a major research topic in computer science in the last decade. Drineas et al. [18] first recognized the need for memory-efficient methods for matrix multiplication when access to single columns and rows of the input matrices is possible. They present a randomized algorithm for approximating the product of two matrices based on sampling. The complexity of the algorithm as well as its accuracy depend on user-defined parameters. The algorithm is “pass-efficient" since columns and rows of the input matrices are sequentially loaded into memory. The obtained approximation guarantees are expressed in terms of the Frobenius norm of the input matrices and the user-defined parameters but their method does not result in a strong guarantee for individual entries in the output matrix. Sarlós [30] observed that instead of sampling rows and columns one can use random projections to obtain a sketch of the matrix product. A notable difference to [18] is that by sketching one obtains an additive error for each individual entry depending on the 2-norm of the corresponding row and column vector in the input matrices.

Recently Pagh [28] introduced a new randomized approximation algorithm. Instead of sketching the input matrices and then multiplying the resulting smaller matrices, we treat the product as a stream of outer products and sketch each outer product. Using Fast Fourier Transformation in a clever way, Pagh shows how to efficiently adapt the Count-Sketch algorithm [11] to an outer product. The algorithm runs in one pass over the input matrices and provides approximation guarantees in terms of the Frobenius norm of their product.

Our contribution.

- A new algorithm for the case where the input matrices consist of nonnegative entries only. This is the first nontrivial deterministic approximation algorithm for the multiplication of nonnegative matrices in a streaming setting. Motivated by practical applications, we analyze the approximation guarantee and the algorithm complexity under the assumption that the entries adhere to Zipfian distribution. We compare it to previously known randomized algorithms and show that for certain natural settings it is more efficient and achieves better approximation guarantees.
- We present a new matrix multiplication algorithm for arbitrary real-valued input matrices by adapting the group testing algorithm for streams with updates in the turnstile model

outlined in [27] for detecting the entries with large absolute value in matrix product. As a byproduct we obtain the first deterministic algorithm running in $O(n^{2+\varepsilon})$ steps for matrix products with $O(\sqrt{n})$ nonzero entries.

Note that our algorithms easily generalize to rectangular matrix multiplication but for the ease of presentation we consider the case of square input matrices. Also, we will state the time complexity of our first algorithm using a function $\mathrm{Sort}(n)$ denoting the running time of a linear space deterministic sorting algorithm. Clearly, $\mathrm{Sort}(n) = O(n \log n)$ for comparison based sorting but under some assumptions on the elements to be sorted also better bounds are known, e.g. the $O(n \log\log n)$ time integer sorting algorithm by Han [21].

2 Preliminaries

2.1 Definitions

Linear algebra. Let $\mathbb{R}_+$ denote the field of nonnegative real numbers. Given matrices $A, B \in {\mathbb{R}_+}^{n\times n}$ we denote their product by $C := AB$. The ith row of a matrix A is written as $A_{i,*}$, the jth column as $A_{*,j}$. We use the term *entry* to identify a position in the matrix, not its value. Thus, *the weight* of the entry (i, j) in A is the value in the ith row and jth column, A_{ij}, $i, j \in [n]$, for $[n] := \{0, 1, \ldots, n-1\}$. When clear from the context however, we will omit weight. For example, nonzero entries will refer to entries whose weight is different from 0 and by heavy entries we mean entries with large weight.

The *outer product* of a column vector $u \in {\mathbb{R}_+}^n$ and a row vector $v \in {\mathbb{R}_+}^n$ is a matrix $uv \in {\mathbb{R}_+}^{n\times n}$ such that $uv_{i,j} = u_i v_j$, $i, j \in [n]$. The *rank* of a positive real number $a \in \mathbb{R}_+$ in a matrix A, denoted as $r_A(a)$, is the number of entries strictly smaller than a, plus 1. Note that a does not need to be present in A.

The *p-norm* of a vector $u \in \mathbb{R}^n$ is $\|u\|_p = (\sum_{i=1}^n |u_i|^p)^{\frac{1}{p}}$ for $p > 0$. Similarly, we define the *entrywise p-norm* of a matrix $A \in \mathbb{R}^{n\times n}$ as $\|A\|_{Ep} := (\sum_{i,j\in[n]} |A_{i,j}|^p)^{1/p}$ for $p \in \mathbb{N}$. The case $p = 2$ is the *Frobenius norm* of A denoted as $\|A\|_F$. The *k-residual entrywise p-norm* $\|A\|_{E^k p}$ is the entrywise p-norm of the matrix obtained from A after replacing the k entries with the largest absolute values in A, ties resolved arbitrarily, with 0.

Data streaming. Our algorithms have strong connection to data streaming, therefore we will use the respective terminology. A *stream* S is a sequence of N updates (i, v) for items $i \in \mathcal{I}$ and $v \in \mathbb{R}$. We assume $\mathcal{I} = [n]$. The *frequency* of i is $f_i = \sum_{(i,v)\in S} v$ and $\mathbf{f}_S = (f_0, \ldots, f_{n-1})$ is the *frequency vector* of the stream S. The *insert-only* model assumes $v > 0$ for all updates and in the *non-strict turnstile* model there are no restrictions on v and the values in $\mathbf{f}_S$ [27]. Similarly to matrix entries, we will also refer to the frequency of an item i as the *weight* of i. Items with weight above $||f||_1/b$, for a user-defined b, will be called *b-heavy hitters* or just *heavy hitters* when b is clear from the context. Ordering the items in S according to their absolute weight, the heaviest b items in S are called the *top-b entries* in S.

Skewed distributions. A common formalization of the skewness in real-life datasets is the assumption of *Zipfian distribution* [36]. The elements in a given set M over N different elements with positive weights follow *Zipfian distribution* with parameter $z > 0$ if the weight of the element of rank i is $\frac{|M|}{\zeta(z)i^z}$ where $\zeta(z) = \sum_{i=1}^N i^{-z}$ and $|M|$ denotes the total weight of elements in M. We will analyze only the case when the skew in the data is not light and $z > 1$. For $z > 1$, $\sum_{i=1}^N i^{-z}$ converges to a small constant. We will also use the facts that for $z > 1$, $\sum_{i=b+1}^N i^{-z} = O(b^{1-z})$ and for $z > 1/2$, $\sum_{i=b+1}^N i^{-2z} = O(b^{1-2z})$.

2.2 The column row method and memory efficient matrix multiplication

The naïve algorithm for the multiplication of input matrices $A, B \in \mathbb{R}^{n \times n}$ works by computing the inner product of $A_{i,*}$ and $B_{*,j}$ in order to obtain AB_{ij} for all $i, j \in [n]$. An alternative view of the approach is the *column row method* computing the sum of outer products $\sum_{i \in [n]} A_{*,i} B_{i,*}$. While this approach does not yield a better running time, it turns out to admit algorithmic modifications resulting in more efficient algorithms. Schnorr and Subramanian [31] and Lingas [25] build upon the approach and obtain faster algorithms for Boolean matrix multiplication. Assuming that A is stored as column-major ordered triples and B as row-major ordered triples [8], the approach yields a memory efficient algorithm since the matrix product AB can be computed in a single scan over the input matrices. Recently, Pagh [28] presented a new randomized algorithm combining the approach with frequent items mining algorithms [1, 11]. Inspired by this, we present another approach to modify the column-row method building upon ideas from deterministic frequent items mining algorithms [15, 16, 24, 26].

3 An algorithm for nonnegative matrix products

3.1 Intuition and key lemma

Recall first how the Majority algorithm [6] works. We are given a multiset M of cardinality N and want to find a majority element, i.e. an element occurring at least $N/2 + 1$ times in M. While there are two distinct objects in M we remove them from M. It is easy to see that if there exists a majority element a, at the end only occurrences of a will be in M.

The Frequent algorithm [16, 24, 26] builds upon this simple idea and detects b-heavy hitters in an unweighted stream S of N updates $(i, 1)$ for items $i \in [n]$. We keep a *summary* of b distinct entries together with a counter lower bounding their weight. Whenever a new item i arrives we check whether it is already in the summary and, if so, update the corresponding counter. Otherwise, if there is an empty slot in the summary we insert i with a counter set to 1. In the case all b slots are occupied we decrease the weight of all items by 1 and proceed with the next item in the stream. The last step corresponds to removing $b + 1$ distinct items from the multiset of items occurring in S and a simple argument shows that b-heavy hitters will be in the summary after processing the stream. By returning the estimated weight of the item in the summary and 0 for not recorded items, the weight of each item is underestimated by at most $\|\mathbf{f}\|_1/b$ where $\mathbf{f}$ is the frequency vector of the stream. Implementing the summary as a hash table and charging the cost of each item deletion to the cost incurred at its arrival the expected amortized cost per item update is constant. A sophisticated approach for decreasing the items weights in the summary leads to a worst case constant time per item update [16, 24].

Generalizing to nonnegative matrix multiplication by the column row method is intuitive. Assume the input matrices consist of $\{0,1\}$-valued entries only. We successively generate the n outer products and run the Frequent algorithm on the resulting stream associating entries with items. There are several problems to resolve: First, we want to multiply arbitrary nonnegative matrices, thus our algorithm has to handle weighted updates. Second, we have to consider $\Theta(n^3)$ occurrences of weighted items in the stream. Third, we cannot apply any more the amortized argument for the running time analysis since a group of $b - 1$ heavy items might be followed by many lighter items causing expensive updates of the summary and it is not obvious how to extend the deterministic approach from [16, 24] guaranteeing

```
function ComputeSummary
Require: matrices $A, B \in \mathbb{R_+}^{n \times n}$, summary $\mathcal{S}$ for $b$ entries
 1: for $i \in [n]$ do
 2:     Denote by $R := A_{*,i} \cdot B_{i,*}$ the outer product of the $i$th column of $A$ and $i$th row of $B$
 3:     Find the weight $w^R_{b+1}$ of the entry of rank $b+1$ in $R$
 4:     Let $\mathcal{L}$ be the $b$ entries in $R$ with rank less than $b+1$, i.e. the largest $b$ entries
 5:     Decrease the weight of each entry in $\mathcal{L}$ by $w^R_{b+1}$
 6:     for each entry $e$ occurring in $\mathcal{S}$ and $\mathcal{L}$ do
 7:         add $e$'s weight in $\mathcal{L}$ to $e$'s weight in $\mathcal{S}$
 8:         remove $e$ from $\mathcal{L}$
 9:     Find the weight $w^{\mathcal{S}\cup\mathcal{L}}_{b+1}$ of the entry of rank $b+1$ in $\mathcal{S} \cup \mathcal{L}$, if any
10:     Update $\mathcal{S}$ to contain the largest $b$ entries in $\mathcal{S} \cup \mathcal{L}$ and decrease their weight by $w^{\mathcal{S}\cup\mathcal{L}}_{b+1}$
```

```
function EstimateEntry
Require: Entry $(i, j)$
 1: if $(i, j)$ is in the summary $\mathcal{S}$ then
 2:     return the weight of $(i, j)$ in $\mathcal{S}$
 3: else
 4:     return 0
```

Figure 1 A high-level pseudocode description of the algorithm. In ComputeSummary we iterate over the n outer products and to each one of them apply Lemma 1 such that only the b heaviest entries remain. We update the summary with the entries output by the outer product. After processing the input matrices we can estimate the weight of an individual entry by checking the summary.

constant time updates in the worst case.

The first issue is easily resolved by the following

▶ **Lemma 1.** Let $\mathbf{f}$ be the frequency vector of an insert only stream S over a domain $[n]$. After successively decrementing t times the weight of at least b distinct items by $\Delta_i > 0$, $1 \leq i \leq t$, such that at each step $f_i \geq 0$ for all $0 \leq i \leq n-1$, it holds $f_k > 0$ for all b-heavy hitters, $k \in [n]$, for all $t \in \mathbb{N}$.

Proof. Since $f_i \geq 0$ for all $i \in [n]$ holds, the total decrease is bounded by $\|\mathbf{f}\|_1$. A decrement of Δ_i in the weight of a given item is witnessed by the same decrement in the weights of at least $b-1$ different items. Thus, we have $b\sum_{i=1}^{t} \Delta_i \leq \|\mathbf{f}\|_1$ which bounds the possible decrease in the weight of a heavy hitter to $\|\mathbf{f}\|_1/b$. ◀

In the next section we show that the specific structure of an outer product allows us to design efficient algorithms resolving the last two issues.

3.2 The algorithm

We assume that $A \in \mathbb{R}_+^{n\times n}$ is stored in column-major order and $B \in \mathbb{R}_+^{n\times n}$ in row-major order. We show how to modify the column row method in order to obtain an additive approximation of each entry in AB in terms of the entrywise 1-norm of AB.

Essentially, we run the Frequent algorithm for the stream of n outer products: we keep a summary $\mathcal{S}$ of b distinct items and for each outer product we want to update the summary with the incoming weighted entries over the domain $[n] \times [n]$. The main difference is that for $b = o(n^2)$ we can use the specific structure of an outer product and update the summary in $o(n^2)$ steps. In COMPUTESUMMARY in Figure 1 for each of the n outer products we simulate the successive application of Lemma 1 until at most b entries with weight larger than 0 remain in the outer product. We then update $\mathcal{S}$ with the remaining entries.

Correctness.

▶ Lemma 2. Let w be the weight of an entry (i, j) in the product $C = AB$. After termination of COMPUTESUMMARY for the estimated weight $\overline{w}$ of w returned by ESTIMATEENTRY, $i, j \in [n]$, holds $\max(w - \|C\|_{E_1}/b, 0) \leq \overline{w} \leq w$.

Proof. The product AB equals $\sum_{i=0}^{n-1} a_i \cdot b_i$ for the columns $a_0, \ldots, a_{n-1}$ of A and the rows $b_0, \ldots, b_{n-1}$ of B. We consider each outer product as n^2 updates for different entries over the domain $[n] \times [n]$ in an insert only stream with positive real weights. We show how the algorithm updates the summary for a single outer product R. First, in line 3 the algorithm finds the entry of rank $b + 1$ in R. In line 4 we decrease the weight of the b largest entries by w^R_{b+1} which yields the same result as the following iterative procedure: While there are at least $b + 1$ nonzero entries in R, find the entry with smallest weight $w_{\min}$ in R and decrease the weight of all non-zero entries by $w_{\min}$. Equivalence holds because we always decrease the weight of an entry with the smallest weight and thus the decrease of the largest b entries weights can never exceed w^R_{b+1}. Also, the decrease can not be smaller than w^R_{b+1} since otherwise we would have more than b non-zero entries in the outer product. Thus, we always decrease by the same amount the weight of at least $b + 1$ different entries which by Lemma 1 guarantees the claimed approximation error. In lines 6–10 we apply essentially the same procedure again for the nonzero entries in the outer product and the entries in the summary. The remaining at most b nonzero entries constitute the updated summary. ◀

Running time. In the following lemmas we present efficient deterministic algorithms for the subroutines used in COMPUTESUMMARY. We concentrate how the algorithm updates the summary for a single outer product. Before presenting our approach, we give the main building blocks that will be used to achieve an efficient solution.

▶ Lemma 3. Given two sorted vectors $u, v \in {\mathbb{R}_+}^n$ we can find the entry of rank b in the outer product uv in time and space $O(n)$.

Proof. First note that we can ignore all $u_i = 0$ and $v_i = 0$ since they will result in a row, respectively column, in the outer product with all entries having weight 0, and clearly we do not need to update the summary with such entries. We reduce the problem to selection of the element of rank b in a Cartesian sum $X + Y = \{x + y : x \in X, y \in Y\}$ for sorted sets of real numbers X and Y. Setting $U = \{\log u_i : u_i \in u\}$ and $V = \{\log v_i : v_i \in v\}$ and searching in the Cartesian sum $U + V$ for the element of rank b corresponds to searching for the entry of rank b in the outer product uv, this follows from monotonicity of the $\log : \mathbb{R}_+ \backslash \{0\} \rightarrow \mathbb{R}$ function. The best known deterministic algorithm for selection in a Cartesian sum of two sorted sets [19] runs in time and space $O(n)$. ◀

▶ Lemma 4. Given vectors $u, v \in {\mathbb{R}_+}^n$, with elements sorted in descending order, we can output an implicit representation of the largest b elements from the outer product uv in a data structure $\mathcal{L}$ in time and space $O(n)$.

Proof. Assume we have found the entry of rank b as outlined in Lemma 3, let this element be c. Let i, j be two pointers for u and v respectively. Initialize $i = 0, j = n - 1$. Assume

i is fixed. We compare c to $u_i v_j$. While it is larger or equal, we move left v's pointer by decreasing j by 1. At the end we add the pair (i, j) to $\mathcal{L}$, denoting that the entries in ith row of uv bigger than c, and thus of rank less than b, are all (i, ℓ) for $\ell \leq j$. Then we go to the next row in uv by incrementing i and repeat the above while-loop starting with the current value of j. When $i = n$ or $j = 0$ we know that all entries smaller than c have been found. Correctness is immediate since the product $u_i v_j$ is monotonically increasing with i and decreasing with j, and thus for each row of the outer product we record the position of the entries smaller than c in $\mathcal{L}$. Both i and j are always incremented or respectively decremented, thus the running time is linear in n. We need to explicitly store only u, v and $\mathcal{L}$, this gives the claimed space usage. ◀

Next we present an efficient approach for updating the summary for a given outer product after finding the entries of rank at most b.

▶ **Lemma 5.** For a given outer product uv, $u, v \in {\mathbb{R}_+}^n$ we update the summary $\mathcal{S}$ in time $O(b + \mathrm{sort}(n))$.

Proof. We first sort the vectors u and v in decreasing order according to the values u_i and v_i, respectively. Let us call the sorted vectors u^s and v^s. Each entry in u^s and v^s will be of the form (val, pos) such that $u_{pos} = val$ and $v_{pos} = val$, respectively, i.e. pos will record the position of a given value in u and v. We define the entry (i, j) in the outer product $u^s v^s$ as a $(val_u val_v, (pos_u, pos_v))$ such that $u_i^s = (val_u, pos_u)$ and $u_j^s = (val_v, pos_v)$. Comparing the entries on the $val_u val_v$ values, we can assume that we compute the outer product of two sorted vectors.

Assume we have computed the data structure $\mathcal{L}$ implicitly representing the largest b entries in $u^s v^s$, as shown in Lemma 4. Now we show how to update the summary with the entries in $\mathcal{L}$ in time $O(b + \mathrm{sort}(n))$. We introduce a *position total order* on entries such that $(i_1, j_1) < (i_2, j_2)$ iff $i_1 n + j_1 < i_2 n + j_2$, $i, j \in [n]$. We will keep the entries in $\mathcal{S}$ in the summary sorted according to this order. Assume we can output the b heaviest entries from a given outer product sorted according to the position total order in $\mathcal{L}$. Then in a merge-like scan through $\mathcal{S}$ and $\mathcal{L}$ we update the entries in $\mathcal{S} \cap \mathcal{L}$, remove those from $\mathcal{L}$ and obtain a sorted data structure containing the entries from $\mathcal{S}$ and $\mathcal{L}$ in $O(b)$ steps. The entry of rank $b + 1$ in the set $\mathcal{L} \cup \mathcal{S}$, which has size at most $2b$, can be found in $O(b)$ by [4]. Thus, if the entries in $\mathcal{L}$ are sorted according to the position total order, updating the summary will run in $O(b)$ steps.

We output the b heaviest entries sorted according to the position total order by the following algorithm. Let $\mathcal{L}$ be implicitly given as a sorted column vector u^s and a sorted row vector v^s as described above, and $\ell \leq n$ integer pairs (q, r_q) denoting that in the qth row in the outer product $u^s v^s$ the first $r_q > 0$ entries have rank not more than b. Clearly, r_q will monotonically decrease as q increases. We start with $q = \ell$, namely the shortest interval, sort the r_q entries according to the position total order. We then decrease q by 1 and sort the next r_{q-1} entries according to the position total order. However, we observe that due to monotonicity $r_{q-1} \geq r_q$ and all elements from v^s appearing in the qth row of $u^s v^s$ also appear in the $(q-1)$th row. Thus, we can sort only the new $r_{q-1} - r_q$ elements and then merge the result with the already sorted r_q elements. We continue like this until the elements in each row of the outer product have been sorted. Then we sort the elements in the column vector u^s according to their position, keeping a pointer to the corresponding sorted subinterval of v^s for each entry in u^s. From this we build the set $\mathcal{L}$ with entries sorted according the position total order. By setting $r_{\ell+1} = 0$ the running time for the ℓ mergings and sortings amounts to $\sum_{i=0}^{\ell}(r_i + \mathrm{Sort}(r_i - r_{i+1}))$. We can bound this sum by $O(b + \mathrm{Sort}(n))$ since $\sum_{i=0}^{\ell} r_i = b$,

$\sum_{i=0}^{\ell}(r_i - r_{i+1}) \le n$ and $\sum_{i=0}^{n-1} f(x_i) \le f(\sum_{i=0}^{n-1} x_i)$ for a monotonically increasing convex function f and numbers x_i, $i \in [n]$, in its domain. ◀

3.3 Analysis of the approximation guarantee

The only remaining component is how to efficiently answer queries to the summary after processing all outer products. We use a static dictionary with constant look-up time. Observing that the entries are from a universe of size n^2, the best known result by Ružić [29] provides a construction in time $O(b \log^2 \log b)$ and space $O(b)$. Note that $b < n^2$, therefore as a first result we observe that Lemmas 2 and 5 immediately yield the following

▶ Lemma 6. Given $n \times n$-matrices A, B with non-negative real entries, there exists a deterministic algorithm approximating the weight of each entry in the product C of A and B within an additive error of $\|C\|_{E_1}/b$. The algorithm runs in time $O(nb + n\text{Sort}(n))$ and space $O(b + n)$ in one pass over the input matrices.

It was first observed by Bose et al. [5] that the Frequent algorithm guarantees tighter estimates for items with weight significantly larger than N/b in a stream of length N and summary of size b. Berinde et al. [3] develop a general framework for the analysis of so called *heavy-tolerant* counter based algorithms and show that Frequent falls in this class.

▶ Lemma 7. (Bose et al, [5]) For an entry (i, j) in $C = AB$ with weight $\alpha\|C\|_{E1}$, $\alpha b > 1$, after termination of ComputeSummary it holds $\widehat{C_{ij}} \ge C_{ij} - (1 - \alpha)\|C\|_{E1}/(b - 1)$ where $\widehat{C_{ij}}$ is the approximation of C_{ij} returned by EstimateEntry(i, j).

▶ Lemma 8. (Berinde et al, [3]) $\|C\|_{E^k 1}/(b - k)$ is an upper bound on the underestimation of any C_{ij} returned by EstimateEntry(i, j) for any $k \le b$.

The above lemmas are important since they yield approximation guarantees depending on the residual k-norm of the matrix product, thus for skewed matrix products the approximation is much better than the one provided by Lemma 6.

Sparse recovery. The approximation of the matrix product $C = AB$ in [18, 28, 30] is analyzed in terms of the Frobenius norm of the difference of C and the obtained approximation $\widehat{C}$, i.e $\|C - \widehat{C}\|_F$. By simply creating a sparse matrix with all non-zero estimations in the summary we obtain an approximation of C: the so called k-sparse recovery of a frequency vector $\mathbf{f}$ aims at finding a vector $\widehat{\mathbf{f}}$ with at most k non-zero entries such that the p-norm $\|\mathbf{f} - \widehat{\mathbf{f}}\|_p$ is minimized.

As shown by Berinde et al. [3] the class of heavy-tolerant counter algorithms yields the best known bounds for the sparse recovery in the p-norm. The following Theorem 1 follows from Lemma 5 and their main result.

▶ Theorem 1. Let A, B be nonnegative $n \times n$ real matrices and $C = AB$ their product. There exists a one-pass approximation deterministic algorithm returning a matrix $\widehat{C}$ such that $\|C - \widehat{C}\|_{Ep} \le (1 + \varepsilon)^{\frac{1}{p}}(\varepsilon/k)^{1-\frac{1}{p}}\|C\|_{E^k 1}$. The algorithm runs in time $O(n \cdot \text{Sort}(n) + (nk)/\varepsilon)$ and uses space $O(n + k/\varepsilon)$ for any $0 < \varepsilon < 1$ and $k \ge 1$.

Clearly, for $k/\varepsilon = o(n^2)$ the algorithm runs in subcubic time and subquadratic memory. In the next paragraph we show that for skewed output matrices EstimateEntry can provably detect the most significant entries even for modest summary sizes.

Zipfian distributions. In the full version of the paper we discuss the practical motivation for the assumption that the entries in the product adhere to a Zipfian distribution. The results stated below not only give a better understanding of the approximation yielded by the algorithm, but also allow direct comparison to related work.

▶ Lemma 9. (Berinde et al, [3]) If the entries weights in the product matrix follow a Zipfian distribution with parameter $z > 1$, then EstimateEntry with a summary of size b

1. approximates the weight of all entries with rank $i \leq b$ with additive error of $(1 - \frac{1}{\zeta(z)i^z})\frac{\|C\|_{E1}}{b-1}$.
2. estimates the weight of all entries with additive error of $\varepsilon\|C\|_{E1}$ for $b = O((\frac{1}{\varepsilon})^{\frac{1}{z}})$.
3. returns the largest k entries in the matrix product for $b = O(k)$.
4. returns the largest k entries in a correct order for $b = \Omega(k(\frac{k}{z})^{\frac{1}{z}})$.

3.4 Comparison to previous work

The randomized algorithm by Cohen and Lewis [13] for computing the product of nonnegative matrices yields an unbiased estimator of each entry and a concentration around the expected entry weight with high probability. However, their algorithm requires a random walk in a bipartite graph of size $\Theta(n^2)$ space and is thus not space efficient. It is difficult to compare the bounds returned by EstimateEntry to the bounds obtained in [18, 30], but it is natural to compare the guarantee of our estimates to the ones shown by Pagh [28].

The approximation error of the matrix estimation $\widehat{C}$ in [28], $\|C - \widehat{C}\|_F$, is bounded by $(n\|C\|_F)/\sqrt{b}$ with high probability. The running time is $O(n^2 \log n + b \log b \log n)$ and space usage is $O(n + b \log n)$. Our deterministic algorithm achieves an error guarantee of $(1 + \varepsilon)(\varepsilon/k)^{1-\frac{1}{p}}\|C\|_{E^k 1}$ for the approximation $\|C - \widehat{C}\|_{Ep}$ for any $p > 0$. For a direct comparison we set $p = 2$, $k = 0$ and $b = \lceil 1/\varepsilon \rceil$ and obtain an approximation error of $\|C\|_{E1}/\sqrt{b}$ which is at most $(n\|C\|_F)/\sqrt{b}$ by Cauchy-Schwarz inequality. The time and space complexity of our algorithm is a polylogarithmic factor better. Note also that the approximation guarantee does not depend on the dimension n as in [28].

For individual entries we achieve an error bounded by $\min_{k \in [b]} \|C\|_{E^k 1}/(b - k)$ while [28] shows that the error of the obtained estimates is bounded by $\|C\|_{E^{b/\kappa} 1}/\sqrt{b}$ for a suitably chosen constant $\kappa > 2$.

Assuming Zipfian distribution with $z > 1$ the approximation error of the Frobenius norm of the matrix product in [28] is bounded by $O(nb^{-z}\|C\|_{E1})$ with high probability. By setting $k = 0$ our deterministic algorithm achieves $O(\|C\|_{E1}/\sqrt{b})$ for the Frobenius norm approximation error. For an $\varepsilon\|C\|_{E1}$-approximation of individual entries both [28] and our algorithm need a data structure, a sketch or a summary, of size $O((1/\varepsilon)^{\frac{1}{z}})$ but [28] needs to run $O(\log n)$ copies of the algorithm in parallel in order to guarantee that the estimates are correct with high probability. In the full version of the paper we show that the approximation guarantee of our algorithm is better than the one in [28] for some real data sets exhibiting higher skew. However, Pagh's algorithm achieves better bounds for lighter skew when $1/2 < z < 1$ and more important it is not restricted to nonnegative input matrices.

4 An algorithm for arbitrary real-valued matrices

In this section we show how to efficiently extend the deterministic streaming algorithm sketched in [15, 27] to matrix multiplication. The algorithm in [15, 27] works for streams in the non-strict turnstile model where updates are of the form (i, v) for an item i and $v \in \mathbb{R}$. We only give an overview of how it works, a thorough description will appear in the full version of the paper.

A majority item in a stream is an item whose absolute total weight is more than half of the absolute sum of total weights of all other items in the stream. In [15] the authors present an elegant group testing algorithm for finding a majority item in a stream in the non-strict turnstile model. The algorithm works by keeping a counter for each bit set to

1 in the binary representation of the items and a global counter for the total weight of all items processed so far. A new item is processed by updating the corresponding bit counters and the global counter. Once the stream is processed, a candidate for the majority item is constructed from the bit counters and the global counter. In a second pass we verify whether the candidate is indeed a majority item. Assuming the items are from the domain $[m]$, we need $O(\log m)$ counters. In the following we will call this algorithm BinaryMajority. A generalization of the algorithm to find the b items with nonzero weight after processing the stream is presented in Theorem 14 in [27]. Let P be a set of suitably chosen consecutive prime numbers and $|P|$ denote its cardinality. Each item $i \in [n]$ is distributed to p distinct groups, depending on the value $i \bmod p$, for each prime $p \in P$. In each such group we run BinaryMajority. The crucial observation is that for sufficiently large set of primes P, and sufficiently big primes in P, each of the b nonzero items will be isolated in at least one group and therefore BinaryMajority will detect it.

Generalizing to matrix multiplication again builds upon the column row method. We treat the matrix product as a stream of n updates to the n^2 entries by each outer product. We assume that $n = 2^\ell$ for some $\ell > 0$, otherwise we add less than n zero entries to each row and column vector such that the assumption holds. We number the n^2 entries of the matrix product as $0, 1, \ldots, n^2 - 1$ such that the entry in the position (i, j) is assigned a number $i2^\ell + j$, $0 \leq i, j \leq n - 1$. Now observe that the number of an entry in the outer product consists of 2ℓ bits and the term j determines only the least significant ℓ bits while the term $i2^\ell$ determines the most significant ℓ bits. In the sequence $0, 1, \ldots, n^2 - 1$ the elements having 1 in the kth position, $0 \leq k \leq 2\ell - 1$, are the ones in positions $\{2^k + i2^{k+1}, \ldots, (i+1)2^{k+1} - 1\}$, $0 \leq i \leq 2^{2\ell-k-1}$. Thus, given a column vector a of A and a row vector b of B the entries in the outer product ab, whose numbers have the kth bit equal to 1, are uniquely determined by the position of the contribution from either a or b. This means that in order to obtain the total contribution from all entries with the ℓth bit equal to 1, we simply need to nullify half of the entries in either a or b, and compute the sum over all entries weights in the outer product ab. The latter can be efficiently done by computing the sum of all entries weights in each vector and then multiplying the results. Thus, a majority candidate can be constructed in one pass over the input matrices, $O(n^2 \log n)$ steps and linear space.

For distributing the n^2 entries in ab to different groups, we apply the technique presented by Pagh [28]. Let p be a given prime number. We treat the column and row vectors a and b as polynomials of degree $p - 1$: $p_a = \sum_{i=0}^{n-1} a_i x^{i \cdot n \bmod p}$ and $p_b = \sum_{j=0}^{n-1} b_j x^{j \bmod p}$. p_a and p_b are then multiplied by Fast Fourier Transformation in time $O(p \log p)$. The resulting polynomial has degree $2(p - 1)$, thus we add the coefficients that have the same exponent modulo p. The coefficient in front of x^k, $k \in [p]$, is now exactly the total sum of the entries weights equal to k modulo p. The approach can be combined with BinaryMajority, which yields the following

▶ Theorem 2. Let A, B be real $n \times n$ matrices and $C = AB$ their product. If the absolute weight of each of the b entries with largest absolute weight is bigger than $\|C\|_{E^b 1}$, then there exists a deterministic algorithm computing the b heaviest entries exactly in time $O(n^2 + nb^2 \log^3 n \log^2 b)$ and space $O(n + b^2 \log^3 n \log b)$ in two passes over the input.

▶ **Corollary 1.** *Let A, B be real $n \times n$ matrices and $C = AB$ their product. If C has at most b nonzero entries then there exists a deterministic algorithm computing C exactly in time $O(n^2 + nb^2 \log^3 n \log^2 b)$ and space $O(n + b^2 \log^3 n \log b)$ in two passes over the input.*

4.1 Zipfian distribution

For the case when the absolute values of the entries in the outer product adhere to Zipfian distribution with parameter $z > 1$ we obtain the following

▶ Theorem 3. Let the absolute values of the entries weights in a matrix product adhere to Zipfian distribution. Then for user-defined $s > 0$ and $k > 0$ there exists a deterministic algorithm detecting the ks heaviest entries in the product in time $O(s(n^2 + nk^{\frac{2z}{z-1}} \log^3 n \log^2 k))$ and space $O(n + ks + k^{\frac{2z}{z-1}} \log^3 n \log k)$ in $2s$ passes over the input matrices.

4.2 Comparison to previous work

The algorithm seems to be only of theoretical interest. Note that its complexity is much worse than the one achieved by Pagh's randomized one-pass algorithm [28]: the b non-zero entries can be found in time $O(n^2 + nb \log n)$ and space $O(n + b \log n)$ with error probability $O(1/\mathrm{poly}(n))$. Nevertheless since the best known space lower bound for finding b non-zero elements by a deterministic algorithm is $O(b \log n)$ there seems to be potential for improvement. For example Ganguly and Majumder [20] present improvements of the deterministic algorithm from [27] but their techniques are not suitable for our problem.

To the best of our knowledge this is the first deterministic algorithm for computing matrix products in time $O(n^{2+\varepsilon})$ for the case when the product contains at most $O(\sqrt{n})$ non-zero entries. The algorithm by Iwen and Spencer achieves this for an arguably more interesting class of matrix products, namely those with n^{β}, $\beta \leq 0.29462$, nonzero entries in each row, but the algorithm relies on fast rectangular matrix multiplication and its simple version runs in time $O(n^{2+\beta})$.

Acknowledgements. I would like to thank my supervisor Rasmus Pagh and the anonymous reviewers for valuable comments and suggestions.

References

1 N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci*, 58(1):137–147, 1999.

2 R. R. Amossen and R. Pagh. Faster join-projects and sparse matrix multiplications. *ICDT 2009*, 121–126.

3 R. Berinde, P. Indyk, G. Cormode, M. J. Strauss: Space-optimal heavy hitters with strong error bounds. *ACM Trans. Database Syst.* 35(4): 26 (2010)

4 M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, R. E. Tarjan. Time Bounds for Selection. *J. Comput. Syst. Sci. 7(4)*: 448–461 (1973)

5 P. Bose, E. Kranakis, P. Morin, Y. Tang. Bounds for Frequency Estimation of Packet Streams. *SIROCCO 2003*: 33–42

6 R. Boyer and S. Moore A Fast Majority Vote Algorithm *U. Texas Tech report*, 1982

7 S. Brin, R. Motwani, C. Silverstein. Beyond Market Baskets: Generalizing Association Rules to Correlations. *SIGMOD 1997*: 265–276

8 A. Buluç. Linear Algebraic Primitives for Parallel Computing on Large Graphs. *PhD thesis*, University of California, Santa Barbara.

9 P. Burgisser, M. Clausen, and M. A. Shokrollahi. Algebraic complexity theory. *Springer-Verlag*, 1997

10 A. Campagna and R. Pagh. Finding associations and computing similarity via biased pair sampling. *Knowl. Inf. Syst.* 31(3): 505–526 (2012)

11 M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci*, 312(1):3–15, 2004

12 E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, C. Yang. Finding Interesting Associations without Support Pruning. *IEEE Trans. Knowl. Data Eng.* 13(1): 64–78 (2001)

13 E. Cohen and D. D. Lewis. Approximating matrix multiplication for pattern recognition tasks. *Journal of Algorithms*, 30(2):211–252, 1999

14 D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990

15 G. Cormode and S. Muthukrishnan. What's hot and what's not: Tracking most frequent items dynamically. *ACM Transactions on Database Systems*, 30(1):249–278, 2005.

16 E. D. Demaine, A. López-Ortiz, J. I. Munro. Frequency Estimation of Internet Packet Streams with Limited Space. *ESA 2002*: 348–360

17 S. V. Dongen. Graph Clustering by Flow Simulation. *PhD thesis*, University of Utrecht, 2000

18 P. Drineas, R. Kannan, and M. W. Mahoney. Fast Monte Carlo algorithms for matrices I: Approximating matrix multiplication. *SIAM Journal on Computing*, 36(1):132–157, 2006

19 G. N. Frederickson, D. B. Johnson. The Complexity of Selection and Ranking in X+Y and Matrices with Sorted Columns. *J. Comput. Syst. Sci. 24(2)*: 197–208 (1982)

20 S. Ganguly and A. Majumder. Deterministic k-set structure. *PODS 2006*: 280–289

21 Y. Han. Deterministic sorting in $O(n \log \log n)$ time and linear space. *J. Algorithms 50(1)*: 96–105 (2004)

22 J. Han, M. Kamber. Data Mining: Concepts and Techniques *Morgan Kaufmann* 2000

23 M. A. Iwen and C. V. Spencer. A note on compressed sensing and the complexity of matrix multiplication. *Inf. Process. Lett*, 109(10):468–471, 2009

24 R. M. Karp, S. Shenker, C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst. 28*: 51–55 (2003)

25 A. Lingas. A fast output-sensitive algorithm for boolean matrix multiplication. *ESA 2009*, 408–419.

26 J. Misra, D. Gries: Finding Repeated Elements. *Sci. Comput. Program.* 2(2): 143–152 (1982)

27 S. Muthukrishnan. Data Streams: Algorithms and Applications. *Foundations and Trends in Theoretical Computer Science*, Vol. 1, Issue 2, 2005

28 R. Pagh. Compressed Matrix Multiplication. *Proceedings of ACM Innovations in Theoretical Computer Science (ITCS)*, 2012

29 M. Ružić. Constructing Efficient Dictionaries in Close to Sorting Time. *ICALP (1) 2008*: 84–95

30 T. Sarlós. Improved Approximation Algorithms for Large Matrices via Random Projections. *FOCS 2006*: 143–152

31 C.-P. Schnorr, C. R. Subramanian. Almost Optimal (on the average) Combinatorial Algorithms for Boolean Matrix Product Witnesses, Computing the Diameter. *RANDOM 1998*: 218–231

32 A. J. Stothers. On the complexity of matrix multiplication. *Ph.D. thesis*, University of Edinburgh, 2010

33 V. Strassen. Gaussian Elimination is not Optimal. *Numer. Math.* 13, 354–356, 1969

34 R. Yuster and U. Zwick. Fast sparse matrix multiplication. *ACM Transactions on Algorithms*, 1(1):2–13, 2005.

35 V. Vassilevska Williams. Multiplying matrices faster than Coppersmith-Winograd. *STOC 2012*, 887–898

36 G. Zipf. Human Behavior and The Principle of Least Effort. Addison-Wesley, 1949

The Simulated Greedy Algorithm for Several Submodular Matroid Secretary Problems

Tengyu Ma[1], Bo Tang[2], and Yajun Wang[3]

1 **Princeton University***
`tengyu@cs.princeton.edu`
2 **University of Liverpool***
`tangbonk1@gmail.com`
3 **Microsoft Research Asia**
`yajunw@microsoft.com`

Abstract

We study the matroid secretary problems with submodular valuation functions. In these problems, the elements arrive in random order. When one element arrives, we have to make an immediate and irrevocable decision on whether to accept it or not. The set of accepted elements must form an *independent set* in a predefined matroid. Our objective is to maximize the value of the accepted elements. In this paper, we focus on the case that the valuation function is a non-negative and monotonically non-decreasing submodular function.

We introduce a general algorithm for such *submodular matroid secretary problems*. In particular, we obtain constant competitive algorithms for the cases of laminar matroids and transversal matroids. Our algorithms can be further applied to any independent set system defined by the intersection of a *constant* number of laminar matroids, while still achieving constant competitive ratios. Notice that laminar matroids generalize uniform matroids and partition matroids.

On the other hand, when the underlying valuation function is linear, our algorithm achieves a competitive ratio of 9.6 for laminar matroids, which significantly improves the previous result.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases secretary problem, submodular function, matroid, online algorithm

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.478

1 Introduction

In the classical secretary problem [8, 11, 12], one interviewer is interviewing n candidates for a secretary position. The candidates arrive in an online fashion and the interviewer has to decide whether or not to hire the current candidate when he/she arrives. The goal is to hire the best secretary. It has been shown that when the candidates are arriving in random order, there exists an algorithm that hires the best candidate with probability $1/e$, where e is the base of the natural logarithm.

Recently, Babaioff et al. [3] formulated the matroid secretary problem. Instead of hiring one candidate (element), in the matroid secretary problem, we seek to select a set of elements which form an independent set in a matroid. Again, the elements arrive in random order and the weights of the elements are revealed when they arrive. When one element arrives, we have to make an immediate and irrevocable decision on whether to accept this element or not. The important constraint is that the set of accepted elements must form an independent

* This work was done when the authors were visiting Microsoft Research Asia.

set in the predefined matroid. The objective is to maximize the total weights of the selected elements. Notice that the decision on accepting a particular element will impact our ability in accepting future elements.

In the matroid secretary problem, the value of a set of elements is the summation of the weights on these elements, i.e., the valuation function is linear. In some applications, however, it is more natural to measure the quality of a set by a valuation function, which is not necessarily linear. One set of functions widely used in the optimization community are the *submodular* functions. Such functions are characterized as functions with diminishing returns. We give the formal definition in Section 2.

For example, consider the following scenario. An advertiser is targeting a few platforms to reach a good coverage of audience. However, the coverage from different platforms may overlap with each other. In this case, the performance of a particular set of platforms can only be modeled as a submodular function. Assume the advertiser has to negotiate with the platforms one by one in an online fashion and has a hard budget limit on targeting at most k platforms. This is exactly the matroid secretary problem with a submodular valuation function on a uniform matroid.

We can also consider multiple arriving advertisers, while assuming platforms are available offline. One can impose constraints both on the advertisers and platforms, e.g., each advertiser can afford k platforms, and each platform can support at most ℓ advertisers. This scenario can be modeled as an intersection of two partition matroids, with a submodular valuation function, where the objective is to maximize the value of an overall online assignment.

In this paper, we extend the matroid secretary problem to the case with submodular valuation functions. In other words, the weights are not directly associated with elements. Instead, there exists an oracle to query the value of any subset of the elements we have seen. Our objective is to accept a set of elements which are independent in a given matroid with maximum value with respect to a submodular valuation function. We refer such problems as *submodular matroid secretary problems*. We refer the original matroid secretary problems, i.e., those with linear valuation functions, as *linear matroid secretary problems*.

We use the competitive analysis to measure the performance of our algorithms following the matroid secretary problem literature. More formally, let U be the set of elements and $\mathcal{M}$ be a matroid defined on U. Before the process starts, an adversary assigns a submodular valuation function $f(\cdot) : 2^{|U|} \to \mathcal{R}^+ \cup \{0\}$, which maps any subset of U to a non-negative real number. After that, there is a random permutation applied to the elements to decide their arriving order to our online algorithm. Our algorithm can only query $f(\cdot)$ using elements that have been seen. In other words, the algorithm does not know $f(\cdot)$ before any element arrives.

Let $\mathrm{OPT}_f(\mathcal{M}) = \max_{S\in\mathcal{M}} f(S)$ be the value of the optimal independent set. The objective of the submodular matroid secretary problem is to find an algorithm Alg which maximizes the following ratio:

$$\inf_f \frac{\mathbb{E}_{\mathcal{P},\mathcal{A}}[f(\mathrm{Alg}_f(\mathcal{P},\mathcal{A}))]}{\mathrm{OPT}_f(\mathcal{M})}, \tag{1}$$

where $\mathrm{Alg}_f(\mathcal{P},\mathcal{A})$ is the solution generated by the algorithm given permutation $\mathcal{P}$ and the internal randomness $\mathcal{A}$ of the algorithm with valuation function $f(\cdot)$. The expectation is taken over all permutations and the internal randomness of the algorithm. We call the algorithm is C-competitive, i.e., with competitive ratio C, if the ratio in Eqn.(1) is at least $1/C$.

Our contributions. In this paper, we study the submodular matroid secretary problem with submodular valuation functions that are *non-negative* and *monotonically non-decreasing*.

Our contribution is two-fold. First, we develop a general *simulated greedy* algorithm, which is inspired by the algorithm for the linear matroid secretary problem with transversal matroids in [6, 17]. Our algorithm is constant competitive for the submodular matroid secretary problem with laminar matroids and transversal matroids. Our analysis can be extended to the case that the independent set is defined as the intersection of a *constant* number of laminar matroids. Notice that laminar matroids generalize uniform matroids and partition matroids. When applying to the linear matroid secretary problem on laminar matroids, our algorithm improves the competitive ratio from $\frac{16000}{3}$ [15] to 9.6. Our algorithm is also much simpler than the one in [15].

Second, our technique in analyzing submodular functions could be of independent interest. Consider our simulated greedy algorithm for the uniform matroid case with cardinality μ. We maintain two sets M and N, which are initially empty. In each time, we will select an element $e \in U \setminus (M \cup N)$ such that $f_M(e)$ is maximized until $|M| = \mu$, where $f(\cdot)$ is the valuation function. With probability p, e is placed into M. Otherwise, i.e., with probability $1-p$, e is placed into N. We develop machinery to show that $\mathbb{E}[f(N)] = \Theta(\mathbb{E}[f(M)])$, despite the fact that the elements are greedily selected with optimal marginal values against M. This fact is not intuitive though very important in our analysis. See our result in Section 4 in more details.

Related work. The secretary problem has been studied decades ago. It is first published in [12] and has been folklore even earlier [10]. Several results have appeared to generalize the classical secretary problem, while assuming that the elements arrive in random order. For example, Kleinberg [16] gave a $1 + O(1/\sqrt{k})$-competitive algorithm for selecting at most k elements to maximize the sum of the weights. Babaioff et al. [2] provided a constant competitive algorithm for the Knapsack secretary problem, in which each element has a weight and a size, and the objective is to accept a set of elements whose total size is at most a given integer such that the total weight is maximized.

Babaioff et al. [3] systematically introduced the *matroid secretary problem.* The objective is to maximize the total weight of the selected elements S, which form an independent set in a given matroid. They gave an $O(\log r)$-competitive algorithm for a general matroid, i.e., the expected total weight of the elements in S is $O(1/\log r)$ of the optimal solution, where r is the rank of the matroid. The competitive ratio has been recently improved to $O(\sqrt{\log r})$ by Chakraborty et al. [5]. However, the conjecture that the matroid secretary problem with a general matroid allows a constant competitive algorithm is still widely open, while constant competitive algorithms have been found for various matroids: uniform/partition matroids [2, 16], truncated partition matroids [3], graphical matroids [1, 17], transversal matroids [6, 17], laminar matroids [15], and regular and decomposable matroids [7]. For general matroids, Soto [19] developed a constant-competitive algorithm in *random assignment model*, i.e., the weights of the elements are assigned uniformly at random. This result can be extended to the case where the elements arrive in an adversarial order [13].

Gupta et al. [14] studied the *non-monotone* submodular matroid maximization problem for both offline and online (secretary) versions. For the online (secretary) version, they provided a $O(\log r)$-competitive algorithm for general matroids and a constant competitive algorithm for uniform matroids (algorithms achieving constant competitive ratios are obtained independently by Bateni et al.[4]) and partition matroids. Feldman et al. [9] developed a simpler algorithm with a better competitive ratio for partition matroids for *monotonically non-decreasing* submodular functions.

Structure. In Section 2, we present some preliminaries and our algorithm. We then analyze a simple stochastic process in Section 3, which serves as a building block for later analysis. In Section 4, we analyze the algorithm for the cases of laminar matroids and the intersection of constant number of laminar matroids. Due to space limitation, some of the proofs and the analysis for the transversal matroid case are deferred to the full version [18].

2 Preliminaries

2.1 Matroids

In the matroid secretary problem, the set of accepted elements must form an independent set defined by a given matroid.

▶ **Definition 1** (Matroids). Let $U \neq \emptyset$ be the ground set and $\mathcal{I}$ be a set of subsets of U. The system $\mathcal{M} = (U, \mathcal{I})$ is a matroid with independent sets $\mathcal{I}$ if:

1. If $A \subseteq B \subseteq U$ and $B \in \mathcal{I}$, then $A \in \mathcal{I}$.
2. If $A, B \in \mathcal{I}$ and $|A| < |B|$, there exists an element $x \in B \setminus A$ such that $A \cup \{x\} \in \mathcal{I}$.

In this paper, we work with the following two matroids.

▶ **Definition 2** (Laminar matroids). Let $U \neq \emptyset$ be the ground set. Let $\mathcal{F} = \{B_1, \ldots, B_\ell\}$ be a family of subsets over U. $\mathcal{F}$ is a laminar family, if for any B_i, B_j such that $|B_i| \leq |B_j|$, either $B_i \cap B_j = \emptyset$ or $B_i \subseteq B_j$. Each set $B_i \in \mathcal{F}$ is associated with capacity $\mu(B_i)$. The laminar family $\mathcal{F}$ and $\mu(\cdot)$ define a matroid $\mathcal{M} = (U, \mathcal{I})$, such that any set $T \subseteq U$ is independent if for all $1 \leq i \leq \ell$, $|T \cap B_i| \leq \mu(B_i)$.

In particular, each B_i defines a capacity constraint on the independent sets and a set is independent if it satisfies all such constraints. For simplicity, we assume all B_is are distinct and $\mu(B_i) < \mu(B_j)$ if $B_i \subset B_j$. Otherwise, the capacity constraint in B_i is redundant.

▶ **Definition 3** (Transversal matroids). Let $G = (L, R, E)$ be an undirected bipartite graph with left nodes L, right nodes R and edges E. In the transversal matroid defined by G, the ground set is L and a set of left nodes $S \subseteq L$ is independent if there exists a matching in G such that the set of left nodes in the matching is S.

2.2 Submodular functions

In this paper, we assume the quality of the solution is measured by a submodular function. Notice that throughout this paper, we only work with *non-negative* and *monotonically non-decreasing* submodular functions.

▶ **Definition 4.** Let U be the ground set. Let $f(\cdot) : 2^{|U|} \to \mathcal{R}$ be a function mapping any subset of U to a real number. $f(\cdot)$ is a submodular function if:

$$\forall S, T \subseteq U,\ \ f(S) + f(T) \geq f(S \cup T) + f(S \cap T).$$

For simplicity, for any set $S \subseteq U$, we define its marginal function value $f_S(\cdot)$ as follows. For any $T \subseteq U$, $f_S(T) = f(S \cup T) - f(S)$. For singletons, we also write $f_S(e) = f_S(\{e\})$. It is not difficult to see that $f_S(\cdot)$ is submodular if $f(\cdot)$ is submodular.

2.3 The simulated greedy algorithm

Our general algorithm is based on the greedy algorithm, as in Algorithm 1.

```
Algorithm 1: GREEDY
  Input: Set $H \subseteq U$ of matroid $(U, \mathcal{I})$ and function $f(\cdot)$
  Output: A set of elements $T \subseteq H$ and $T \in \mathcal{I}$
  $T \leftarrow \emptyset$;
  while $\exists\, e^* = \text{argmax}_{e \in H} \{f_T(e) \mid M \cup \{e\} \in \mathcal{I}\}$ do
      $T \leftarrow T \cup \{e\}$; $H \leftarrow H \setminus \{e\}$;
  end
  return $T$;
```

```
Algorithm 2: ONLINE
  Input: Matroid $(U, \mathcal{I})$ and function $f(\cdot)$
  Output: The set of selected elements ALG
  $M, N, \text{ALG} \leftarrow \emptyset$;
  $m \leftarrow Binom(|U|, p)$;
  Observe the first $m$ elements denoted by $H$;
  $M \leftarrow \text{GREEDY}(H)$;
  for any subsequent element $e$ do
      if $\text{GREEDY}(H \cup \{e\}) \neq \text{GREEDY}(H)$ then
          $N \leftarrow N \cup \{e\}$;
          if $\text{ALG} \cup \{e\} \in \mathcal{I}$ then
              Accept $e$ and $\text{ALG} \leftarrow \text{ALG} \cup \{e\}$;
          end
      end
  end
```

```
Algorithm 3: SIMULATE
  Input: Matroid $(U, \mathcal{I})$ and function $f(\cdot)$
  Output: The set of selected elements $S$
  $H, M, N, S \leftarrow \emptyset$;
  for each element $e$ do
      Flip a coin with probability $p$ of head;
      if head, $H \leftarrow H \cup \{e\}$;
  end
  while $\exists\, e^* = \text{argmax}_{e \in U \setminus \{M \cup N\}} \{f_M(e) \mid M \cup \{e\} \in \mathcal{I}\}$ do
      if $e \in H$ then $M \leftarrow M \cup \{e\}$;
      else $N \leftarrow N \cup \{e\}$;
  end
  Prune $N$ to produce a set of elements $S \in \mathcal{I}$;
```

Our *simulated greedy* algorithm ONLINE works as follows. (We will discuss the name of *simulated greedy* in a minute.) We observe the first m elements H without any selection, where m is sampled from Binomial distribution $Binom(n, p)$ for some chosen probability p. Then we compute the greedy solution $\text{GREEDY}(H)$. After that, for any subsequent element e, we test that whether the greedy solution will change if e is added to H hypothetically. If so, we mark e as a candidate and place it in N. Furthermore, if $\text{ALG} \cup \{e\}$ is independent in $\mathcal{I}$ for candidate e and current ALG, we accept e into ALG. (Both N and ALG are initially empty.) The final ALG will be the output of our algorithm. Observe that maintaining set N is not necessary because N only collects elements that has passed the greedy check and might be accepted potentially. However, we keep the notation in the algorithm because it corresponds to the same N in SIMULATE, which is heavily used throughout the analysis.

As we mentioned earlier, ONLINE is a generalization of the algorithms in [6, 17]. In particular, it has been observed that a *simulated* random algorithm in Algorithm 3 can be used in analyzing the performance of ONLINE. (We name ONLINE as a *simulated greedy* algorithm because of the corresponding greedy algorithm which simulates the online version.)

More specifically, SIMULATE works as follows. We maintain two sets M and N which are initially empty. In each step, we select an element $e \in U \setminus (M \cup N)$ such that $f_M(e)$ is maximized and $M \cup \{e\} \in \mathcal{I}$. (If no such element exists, SIMULATE terminates.) Then we toss a biased random coin with probability p to be head, which is the same probability in sampling m in ONLINE. If the coin is head, e is placed into M. Otherwise, e is placed into

N. Since N may not be an independent set in $\mathcal{I}$ after SIMULATE terminates, we prune N to produce $S \subset N$ such that $S \in \mathcal{I}$. The actual pruning step might be different in different application settings.

SIMULATE is useful in analyzing the performance of ONLINE with random arriving elements, because, as the naming suggests, both M and N have the same joint distribution in the two algorithms. This connection is extensively discussed in [6, 17]. For completeness, we provide a proof in the full version [18]. We will guarantee that S in SIMULATE is stochastically dominated by ALG in ONLINE. Since we assume $f(\cdot)$ is non-decreasing, in analyzing the performance of ONLINE, we can focus on S in SIMULATE.

▶ **Lemma 5.** *The sets of elements of H, M and N by* SIMULATE *have the same joint distribution as the H, M and N generated by* ONLINE *with a random permutation of the elements in U.*

3 A simple stochastic process

In this section, we study a simple stochastic process which serves as a building block of our analysis. We will apply this process to either the entire ground set U or some subsets of the elements in U. Therefore, although we use the same notation for M and N in this section, they can be viewed as the intersections between the set of elements that are under consideration and the actual global M and N generated by the algorithm.

The simple stochastic process is defined by an underlying Bernoulli process, with an infinite sequence of independent and identical random variables $X_t \in \{0, 1\}$ for $t \geq 1$. Each variable X_t is a Bernoulli random variable with probability p to be 1.

Our stochastic process is parametrized by a constant $\mu \geq 1$. We maintain two sets M and N, which are initially empty, as follows. Starting from $t = 1$, if $X_t = 1$, we place t into M; otherwise, t is placed in N. The process immediately terminates after $|M| = \mu$.

We associate a non-negative weight w_t to every time stamp t. In particular w_t is a mapping from the previous $t-1$ random variables to a non-negative real number. (w_1 is constant by definition. If the process has been terminated before time t, we set $w_t = 0$.) For any set $T \subseteq \mathbb{N}$, we define the weight as,

$$w(T) = \sum_{t \in T} w_t(X_1, X_2, \ldots, X_{t-1}). \tag{2}$$

Define $w(\emptyset) = 0$. The following proposition shows that the total weights of M and N are close to each other.

▶ Proposition 6. $\mathbb{E}[w(M)] = \frac{p}{1-p}\mathbb{E}[w(N)]$.

Notice that after the process terminates, we have $|M| = \mu$. On the other hand, the size of N might be very large. Our analysis will be based on Ns that are with size at most μ. We produce an independent set S from N by a pruning process as follows.

Pruning. More formally, to address the issue of too large Ns, we define $S = N$ if $|N| \leq \mu$ and $S = \emptyset$ otherwise. Clearly, we have $S \subseteq N$ and $w(S) \leq w(N)$.

We want to show that $w(S)$ is close to $w(N)$ in expectation. However , it is not possible for arbitrary set of $\{w_t\}$. In what follows, we impose a "decreasing weight" condition on $\{w_t\}$, which always holds in our applications. This condition is crucial in building the connection between $w(S)$ and $w(N)$.

▶ **Definition 7** (Decreasing weight mappings). The set of mappings $\{w_t\}$ forms a sequence of decreasing weight mappings if for any $i < j$ and $x_1, x_2, \ldots, x_{i-1}, x_i, \ldots, x_{j-1}$ we have:

$$w_i(x_1, \ldots, x_{i-1}) \geq w_j(x_1, \ldots, x_{i-1}, \ldots, x_{j-1}).$$

▶ Proposition 8. Let $\beta = 2e(1-p)$. If $\{w_t\}$ forms a sequence of decreasing weight mappings, we have

$$\mathbb{E}[w(N)] - \mathbb{E}[w(S)] \leq \frac{(\mu + 1 - \mu\beta)\beta^{\mu}}{(1-\beta)^2} \cdot \mathbb{E}[w(S)] \leq \frac{(\mu + 1 - \mu\beta)\beta^{\mu}}{(1-\beta)^2} \cdot \mathbb{E}[w(N)]$$

If $\mu = 1$, it can be improved to

$$\mathbb{E}[w(N)] - \mathbb{E}[w(S)] \leq \frac{1-p^2}{p^2}\mathbb{E}[w(S)] \leq \frac{1-p^2}{p^2}\mathbb{E}[w(N)].$$

We briefly discuss the intuition behind this statement. Our objective is to show that the weight pruned from N to S is small. The random process indicates that the probability for having a large N is exponentially decreasing on its size, e.g., by the Chernoff bound. Therefore, the probability mass of N that is pruned is small. In terms of weight, on the other hand, those larger Ns do have greater weights.

The condition of the decreasing weight mappings comes to rescue. In particular, in this case, the weight of N grows roughly “linear” to its size. As the probability decreases exponentially with the size of N, the total weight pruned can still be bounded as the summation of a geometric sequence for those large Ns. The complete proof can be found in [18].

4 Laminar Matroid

In this section, we study the performance of our simulated greedy algorithm SIMULATE for the submodular matroid secretary problem with a laminar matroid. We first show that the entire process of SIMULATE can be casted as a simple stochastic process as discussed in the previous section. After that, we inspect the pruning stage in details. In particular, for each B_i in the laminar matroid, we study a simple stochastic process restricted on the elements in B_i. The loss on the entire pruning steps can be divided into losses on the B_is, which can be bounded by Proposition 8.

Let μ be the rank of the laminar matroid. Essentially, SIMULATE will select (at most) μ elements. We cast the SIMULATE process to the simple stochastic process with μ as follows.

In the t-th round, when the first $t-1$ random coins are tossed, the current element e in the greedy order is uniquely defined, as well as the current M and N. We define the weight $w_t = f_{M_e}(e)$ where M_e is the current elements in M.

Remark. We make two remarks regarding the connection between the two stochastic processes. First, the original simple stochastic process terminates when $|M| = \mu$. SIMULATE might terminate earlier because of the limit on the number of elements. In such cases, we assume the availability of an *infinite* number of dummy elements, with zero weights, which will eventually fill up M. In particular, when any of these dummy element arrives at time t, $w_t = 0$ with respect to the previous random outcomes. Notice that these dummy elements will enlarge the size of N without increasing the weights of N and S. So all conclusions we draw in last section still hold. Second, M (as well as N and S) in the simple stochastic process consists of time stamps, while in all processes we study later M consists of real elements. Nevertheless, for every real element $e \in M$, we define $w(e) = w_t$ where t is the time e appears in the greedy order of SIMULATE. Both w_t and $w(e)$ are random variables. We have $w(M) = \sum_{e \in M} w(e)$.

We extend the $w(\cdot)$ to elements besides those in M. In particular, $w(e) = f_{M_e}(e)$ for $e \in M \cup N$, i.e., e appears in the greedy order of SIMULATE, where M_e is the current set of elements in M when e appears. If $e \notin M \cup N$, set $w(e) = 0$. Notice that $w(M) = f(M)$

by definition. Furthermore, each element in the offline optimal solution has probability p in H, i.e., a head coin is associated with it. By submodularity of $f(\cdot)$, the expected value of the optimal solution in H is at least $p \cdot \mathrm{OPT}$. On the other hand, the greedy algorithm is a 2-approximation with a matroid constraint when the valuation function is monotone and submodular. Together with Proposition 6, we have

▶ **Lemma 9.**

$$\mathbb{E}[f(M)] = \mathbb{E}[w(M)] = \frac{p}{1-p}\mathbb{E}[w(N)] \geq \frac{p}{2} \cdot \mathrm{OPT}$$

Pruning. Notice that although M is independent, N might not be independent. We obtain S by pruning N as follows.

$$S = N \setminus \left(\bigcup_{B \in \mathcal{F}} \mathbf{1}_{|N \cap B| > \mu(B)} \cdot (N \cap B) \right), \tag{3}$$

where $\mathbf{1}_{cond} \cdot (N \cap B) = N \cap B$ if *cond* is true and empty otherwise. In other words, if one constraint B_i is violated in N, we remove all elements in B_i from N. Clearly, S is independent. Furthermore, since ALG will be the greedy independent set of N for a random order, it is straightforward to show that $S \subseteq \mathrm{ALG}$.

Therefore, it is sufficient to bound $\mathbb{E}[f(S)]$. To do that, we first provide a lower bound for $\mathbb{E}[w(S)]$. After that, we bound $\mathbb{E}[f(S)]$ in terms of $\mathbb{E}[w(S)]$.

Roadmap. Here we briefly outline our strategy in getting the two pieces of results. To measure $\mathbb{E}[w(S)]$, we estimate the weight loss due to the pruning in Eqn.(3). For each constraint B_i, we cast the stochastic process in SIMULATE in processing elements in B_i into a simple stochastic process with $\mu(B_i)$. By invoking Proposition 8, the weight loss $w(N \cap B_i) - w(S \cap B_i)$ is $2^{O(\mu(B_i))} \cdot w(N \cap B_i)$, which is charged to all elements in B_i proportionally to $\mathbf{1}_{e \in N} w(e)$ for all $e \in B_i$. The catch here is, for each element $e \in U$, the set of $\{B_i\}$ containing e has a strictly increasing $\{\mu(B_i)\}$ sequence. Therefore, the charges on e form a geometric sequence which in total will not exceed a constant fraction of $\mathbf{1}_{e \in N} \cdot w(e)$. Since $w(N) = \sum_{e \in N} w(e)$, the total weight loss is a constant fraction.

The second piece of ingredient is to make a connection between $\mathbb{E}[f(S)]$ and $\mathbb{E}[w(S)]$. For simplicity, let us consider $\mathbb{E}[f(N)]$ and $\mathbb{E}[f(M)]$ instead to convey the idea. Recall that $w(N) = \sum_{e \in N} f_{M_e}(e)$, where M_e is the set of elements in M when e arrives. Therefore, it is not intuitive why $\mathbb{E}[f(N)]$ should be large in the first place. To elaborate, we consider function $F = f(M) + 2f(N) - f(M \cup N)$ during the execution of the algorithm, which is a lower bound of $2f(N)$. We can view $f(M) + f(N) - f(M \cup N)$ as the *intersection* between M and N, e.g., if $f(\cdot)$ is modeling a set cover. During the execution of the algorithm, when e arrives, we have two cases: (1) $f_{M_e}(e) \approx f_{N_e}(e)$, where M_e and N_e are the current set of M and N respectively. F will grow nicely proportional to $f_{M_e}(e)$ in this case. (2) $f_{M_e}(e) \gg f_{N_e}(e)$. Notice e is placed into M with probability p, in which case F grows proportional to $f_{M_e}(e)$ as well. This is because $f_{M_e \cup N_e}(e) \leq f_{N_e}(e) \ll f_{M_e}(e)$ due to the submodularity of $f(\cdot)$. Therefore, F grows in both cases in expectation, which gives a lower bound for $\mathbb{E}[f(N)]$ with respect to $\mathbb{E}[f(M)]$. The analysis in bounded $f(S)$ is more complicated. Though the underlying idea is identical. We formally implement these two ideas in Lemma 10 and Lemma 11.

▶ **Lemma 10.** *Let $\beta = 2e(1-p)$. We have*

$$\mathbb{E}[w(S)] \geq (1 - \frac{2\beta}{(1-\beta)^3})\mathbb{E}[w(N)].$$

Proof. Since for a fixed set of random outcomes, $w(\cdot)$ is a linear function. By Eqn.(3), we have that

$$\mathbb{E}[w(N)] \leq \mathbb{E}[w(S)] + \sum_{B \in \mathcal{F}} \mathbb{E}[w(\mathbf{1}_{|N \cap B| > \mu(B)} \cdot (N \cap B))].$$

Now we focus on the term $\mathbb{E}[w(\mathbf{1}_{|N \cap B| > \mu(B)} \cdot (N \cap B))]$ and the simulated greedy algorithm on elements in B, i.e., a particular constraint in $\mathcal{F}$. We isolate B in the process by rearranging the randomness as follows. First, for each element in $U \setminus B$, we assign an independent random coin to it, i.e., if this element appears in the algorithm, its random coin will be tossed. For a fixed outcome of all random coins outside of B, the simulated greedy algorithm is a *simple stochastic process* for the elements in B. The only difference, however, is the process may terminate before $|M \cap B| = \mu(B)$. This can be easily remedied by appending dummy elements as before. Recall that $\beta = 2e(1-p)$. By Proposition 8, we have:

$$\mathbb{E}[\mathbf{1}_{|N \cap B| > \mu(B)} \cdot w(N \cap B)] \leq \frac{(\mu(B) + 1 - \mu(B)\beta)\beta^{\mu(B)}}{(1-\beta)^2} \cdot \mathbb{E}[w(N \cap B)]. \tag{4}$$

It follows that

$$\begin{aligned}
&\mathbb{E}[w(N)] \\
&\leq \mathbb{E}[w(S)] + \sum_{B \in \mathcal{F}} \frac{(\mu(B) + 1 - \mu(B)\beta)\beta^{\mu(B)}}{(1-\beta)^2} \cdot \mathbb{E}[w(N \cap B)] \\
&= \mathbb{E}[w(S)] + \frac{1}{(1-\beta)^2} \sum_{B \in \mathcal{F}} \sum_{e \in U} \mathbb{E}[(\mu(B) + 1 - \mu(B)\beta)\beta^{\mu(B)} \cdot w(e)\mathbf{1}_{e \in B} \cdot \mathbf{1}_{e \in N}] \\
&= \mathbb{E}[w(S)] + \frac{1}{(1-\beta)^2} \sum_{e \in U} \mathbb{E}\left[w(e)\mathbf{1}_{e \in N} \left(\sum_{B \in \mathcal{F}} (\mu(B) + 1 - \mu(B)\beta)\beta^{\mu(B)} \cdot \mathbf{1}_{e \in B}\right)\right] \\
&\leq \mathbb{E}[w(S)] + \frac{1}{(1-\beta)^2} \sum_{e \in U} \mathbb{E}[w(e)\mathbf{1}_{e \in N}] \left(\sum_{i \geq 1} (i + 1 - i\beta)\beta^i\right) \qquad (5) \\
&= \mathbb{E}[w(S)] + \frac{2\beta}{(1-\beta)^3} \mathbb{E}[w(N)]
\end{aligned}$$

Eqn.(5) uses the fact that the set of constrains $\{B_i\}$ containing an element e has a strictly increasing sequence of $\{\mu(B_i)\}$. ◀

We then bound $\mathbb{E}[f(S)]$ as follows. For an element e, let N_e be the set of elements in N when e appears in SIMULATE. We define $g(e) = f_{N_e}(e)$ if $e \in M \cup N$ and $g(e) = 0$ otherwise. [1]

▶ **Lemma 11.** *For any $t > 0$, let $\theta = 1 + \frac{(1-p)t}{p}$. We have*

$$\mathbb{E}[f(S)] \geq (\frac{1}{\theta} - \frac{(1-\beta)^3}{t((1-\beta)^3 - 2\beta)})\mathbb{E}[w(S)]$$

[1] We define $g(e)$ based on N_e instead of S_e, i.e., the current set of elements in S, because S_e is still a random set even all the randomness before e's arrival is fixed.

Proof. Let $g(S) = \sum_{e \in S} g(e)$. Since $S \subseteq N$, we have $f(S) \geq g(S)$ by the submodularity of $f(\cdot)$. We inspect the function $F(S, M, N) = t \cdot g(S) + f(M) - f(M \cup N)$. By the monotonicity of f, $f(S) \geq g(S) \geq F(S, M, N)/t$.

Define $\Delta_e = F(S'_e, M'_e, N'_e) - F(S_e, M_e, N_e)$ where M'_e (resp. N'_e and S'_e) is the set M (resp. N and S) after we process element e. If $e \notin M \cup N$, define $\Delta_e = 0$. Therefore, $F(S, M, N) = \sum_{e \in U} \Delta_e$. Let $\mathcal{R}_e$ be the sub-σ-algebra encoding all randomness up to the time e is picked in SIMULATE. Notice that M_e and N_e are $\mathcal{R}_e$ measurable. We have $\Pr[e \in M \mid \mathcal{R}_e] = p$ and $\Pr[e \in N \mid \mathcal{R}_e] = 1 - p$.

$$\begin{aligned}
\mathbb{E}[\Delta_e \mid \mathcal{R}_e] &= t \cdot (\mathbb{E}[g(S') - g(S) \mid \mathcal{R}_e]) + (\mathbb{E}[f(M') - f(M) \mid \mathcal{R}_e]) \\
&\quad - (\mathbb{E}[f(M' \cup N') - f(M \cup N) \mid \mathcal{R}_e]) \\
&= t \cdot \Pr[e \in S \mid \mathcal{R}_e] f_{N_e}(e) + \Pr[e \in M \mid \mathcal{R}_e] f_{M_e}(e) - f_{M_e \cup N_e}(e)
\end{aligned}$$

Then we bound $\mathbb{E}[\Delta_e \mid \mathcal{R}_e]$ by case analysis. Notice that $\Pr[e \in M \mid \mathcal{R}_e] + \Pr[e \in N \mid \mathcal{R}_e] = 1$ and $\Pr[e \in N \mid \mathcal{R}_e] \geq \Pr[e \in S \mid \mathcal{R}_e]$.
Case 1: $f_{M_e}(e) \geq \theta \cdot f_{N_e}(e)$.

$$\begin{aligned}
\mathbb{E}[\Delta_e \mid \mathcal{R}_e] &\geq \Pr[e \in M \mid \mathcal{R}_e](f_{M_e}(e) - f_{M_e \cup N_e}(e)) - \Pr[e \in N \mid \mathcal{R}_e] f_{M_e \cup N_e}(e) \\
&\geq \frac{p}{1-p}(1 - \frac{1}{\theta}) \Pr[e \in S \mid \mathcal{R}_e] f_{M_e}(e) - \Pr[e \in N \mid \mathcal{R}_e] f_{M_e}(e)
\end{aligned}$$

Case 2: $f_{M_e}(e) < \theta \cdot f_{N_e}(e)$.

$$\begin{aligned}
\mathbb{E}[\Delta_e \mid \mathcal{R}_e] &\geq t \cdot \Pr[e \in S \mid \mathcal{R}_e] f_{N_e}(e) - \Pr[e \in N \mid \mathcal{R}_e] f_{M_e \cup N_e}(e) \\
&\geq \frac{t}{\theta} \Pr[e \in S \mid \mathcal{R}_e] f_{M_e}(e) - \Pr[e \in N \mid \mathcal{R}_e] f_{M_e}(e)
\end{aligned}$$

By definition of θ, we have $\frac{p}{1-p}(1 - \frac{1}{\theta}) = t/\theta$. So

$$\mathbb{E}[\Delta_e \mid \mathcal{R}_e] \geq \frac{t}{\theta} \Pr[e \in S \mid \mathcal{R}_e] f_{M_e}(e) - \Pr[e \in N \mid \mathcal{R}_e] f_{M_e}(e)$$

Therefore

$$\begin{aligned}
t \cdot \mathbb{E}[f(S)] \geq \mathbb{E}[F(S, M, N)] &= \sum_e \mathbb{E}_{\mathcal{R}_e}[\mathbb{E}[\Delta_e \mid \mathcal{R}_e]] \\
&\geq \sum_{e \in U} \mathbb{E}_{\mathcal{R}_e}[\frac{t}{\theta} \Pr[e \in S \mid \mathcal{R}_e] f_{M_e}(e) - \Pr[e \in N \mid \mathcal{R}_e] f_{M_e}(e)] \\
&= \frac{t}{\theta} \mathbb{E}[w(S)] - \mathbb{E}[w(N)] \qquad (6) \\
&\geq \frac{t}{\theta} \mathbb{E}[w(S)] - \frac{(1-\beta)^3}{(1-\beta)^3 - 2\beta} \mathbb{E}[w(S)]
\end{aligned}$$

The last inequality is by Lemma 10. So $\mathbb{E}[f(S)] \geq (\frac{1}{\theta} - \frac{(1-\beta)^3}{t((1-\beta)^3 - 2\beta)}) \mathbb{E}[w(S)]$ ◀

Combining all the results together, we have an algorithm with competitive ratio at most 211 with $p = 0.9794$ and $t = 10.1415$.

▶ **Theorem 12.** *There is an online algorithm with competitive ratio at most* 211 *for the submodular matroid secretary problem with laminar matroids.*

5 Conclusion

In this paper, we develop a general algorithm for the submodular matroid secretary problems. In particular, we obtain constant competitive algorithms for laminar matroids and transversal matroids. Our algorithm can also handle the intersection of a constant number of laminar matroids, which makes our algorithm more applicable. We state the results for transversal matroids and intersection of matroids, and defer their proofs in the full version [18], where we also analyze the algorithm for the linear matroid secretary problem with laminar matroids.

▶ **Theorem 13.** *There is an online algorithm with competitive ratio at most 95 for the submodular matroid secretary problem with transversal matroids.*

▶ **Theorem 14.** *For any constant k, there is an online algorithm with competitive ratio at most $\frac{1000k(k+1)}{9}$ for the submodular matroid secretary problem with the intersection of k laminar matroids.*

▶ **Theorem 15.** *Algorithm 2 is a 9.6-competitive algorithm for the linear matroid secretary problem with laminar matroids.*

However, our algorithm does not work on general matroid case. Consider the following simple example on graphical matroids. There is a single heavy edge (u, v) in the graph. There is a large number of nodes $K = \{u_1, u_2, \dots, u_n\}$ and edges $\{(u, u_i), (u_i, v) \mid u_i \in K\}$. The weight on each such edge is very small. It is easy to verify that the probability that our algorithm will accept (u, v) is exponentially small on n. Nevertheless, our algorithm can handle graphical matroids using the same decomposition technique [1], i.e., by reducing the problem to a partition matroid, which is randomly selected from two constructed partition matroids. On the other hand, it would be interesting to characterize the independent set constraints for which our algorithm framework is constant competitive.

In the distinction between the submodular case and linear case in matroid secretary problem, we still cannot adapt the recent $O(\sqrt{\log r})$ competitive algorithm in [5] as well as the constant competitive algorithm for the random assignment model in [19] previously on the linear case. It would be interesting to close this gap. Finally, it is still widely open whether the matroid secretary problem permits constant competitive algorithms for general matroids.

References

1 Moshe Babaioff, Michael Dinitz, Anupam Gupta, Nicole Immorlica, and Kunal Talwar. Secretary problems: weights and discounts. In *SODA*, pages 1245–1254, 2009.

2 Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. A knapsack secretary problem with applications. In *APPROX/RANDOM*, pages 16–28, 2007.

3 Moshe Babaioff, Nicole Immorlica, and Robert Kleinberg. Matroids, secretary problems, and online mechanisms. In *SODA*, pages 434–443, 2007.

4 Mohammad Hossein Bateni, MohammadTaghi Hajiaghayi, and Morteza Zadimoghaddam. The submodular secretary problem and its extensions. In *APPROX*, pages 39–52, 2010.

5 Sourav Chakraborty and Oded Lachish. Improved competitive ratio for the matroid secretary problem. In *SODA*, pages 1702–1712, 2012.

6 Nedialko B. Dimitrov and C. Greg Plaxton. Competitive weighted matching in transversal matroids. In *ICALP*, pages 397–408, 2008.

7 Michael Dinitz and Guy Kortsarz. Matroid secretary for regular and decomposable matroids. *CoRR*, abs/1207.5146, 2012.

8 E. B. Dynkin. Optimal choice of the stopping moment of a markov process. *Dokl.Akad.Nauk SSSR*, 150:238–240, 1963.
9 Moran Feldman, Joseph (Seffi) Naor, and Roy Schwartz. Improved competitive ratios for submodular secretary problems (extended abstract). In *APPROX*, pages 218–229, 2011.
10 T. S. Ferguson. Who solved the secretary problem? *Statistical science*, pages 282–289, 1989.
11 PR Freeman. The secretary problem and its extensions: A review. *International Statistical Review*, pages 189–206, 1983.
12 M. Gardner. Mathematical games column. *Scientific American*, 35, 1960.
13 Shayan Oveis Gharan and Jan Vondrák. On variants of the matroid secretary problem. In *ESA*, pages 335–346, 2011.
14 A. Gupta, A. Roth, G. Schoenebeck, and K. Talwar. Constrained non-monotone submodular maximization: Offline and secretary algorithms. In *WINE*, pages 246–257, 2010.
15 Sungjin Im and Yajun Wang. Secretary problems: Laminar matroid and interval scheduling. In *SODA*, 2011.
16 Robert Kleinberg. A multiple-choice secretary algorithm with applications to online auctions. In *SODA*, pages 630–631, Philadelphia, PA, USA, 2005.
17 Nitish Korula and Martin Pál. Algorithms for secretary problems on graphs and hypergraphs. In *ICALP (2)*, pages 508–520, 2009.
18 T. Ma, B. Tang, and Y. Wang. The simulated greedy algorithm for several submodular matroid secretary problems. *arXiv preprint arXiv:1107.2188*, 2011.
19 Jose Soto. Matroid secretary problem in the random assignment model. In *SODA*, pages 1275–1284, 2011.

Hardness of Conjugacy, Embedding and Factorization of multidimensional Subshifts of Finite Type*

Emmanuel Jeandel[1] **and Pascal Vanier**[2]

1 **LORIA**
Campus Scientifique - BP 239
54506 Vandoeuvre-les-Nancy
France
`emmanuel.jeandel@loria.fr`

2 **Einstein Institute of Mathematics**
Hebrew University
Givat Ram
Jerusalem 91904
Israel
`pascal.vanier@lif.univ-mrs.fr`

Abstract

Subshifts of finite type are sets of colorings of the plane defined by local constraints. They can be seen as a discretization of continuous dynamical systems. We investigate here the hardness of deciding factorization, conjugacy and embedding of subshifts of finite type (SFTs) in dimension $d > 1$. In particular, we prove that the factorization problem is Σ_3^0-complete and that the conjugacy and embedding problems are Σ_1^0-complete in the arithmetical hierarchy.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Subshifts, Computability, Factorization, Embedding, Conjugacy

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.490

A d-dimensional Subshift of Finite Type (SFT) is the set of colorings of $\mathbb{Z}^d$ by a finite set of colors in which a finite set of forbidden patterns never appear. One can also see them as tilings of $\mathbb{Z}^d$, and in dimension 2 they are equivalent to the usual notion of tilings introduced by Wang [13]. SFTs are a way to discretize continuous dynamical systems: if X is a compact space and $\phi : X \to X$ a continuous map, we can partition X in a finite number of parts $\Sigma = \{1, \ldots, n\}$ and transform the orbit of a point $x \in X$ into a sequence $(x_n)_{n\in\mathbb{N}^*}$, where x_i denotes the part of X in which $\phi^i(x)$ lies.

Conjugacy is the right notion of isomorphism between subshifts, and plays a major role in their study: when two subshifts are conjugate they code each other and hence have the same dynamical properties. Conjugacy is an equivalence relation and allows to separate SFTs into equivalence classes. Deciding whether two SFTs are conjugate is called the classification problem. It is a long standing open problem in dimension one [4], although has been proved decidable in the particular case of one-sided SFTs on $\mathbb{N}$, see [14]. It has been known for a long time that in higher dimensions the problem is undecidable when given two SFTs, since it can be reduced to the emptyness problem which is Σ_1^0-complete [1]. However, we prove here a slightly stronger result: even by fixing the class in advance, it is still undecidable to decide whether some given SFT belongs to it:

* This work was sponsored by the ANR grant ANR-09-BLAN-0164 and the ISF grant number 1409/11.

▶ **Theorem 1.** *For any fixed X, given Y as an input, it is Σ_1^0-complete to decide if X and Y are conjugate.*

An interesting open question for higher dimension that would probably help solve the one dimensional problem would be *is conjugacy of subshifts decidable when provided an oracle answering whether or not a pattern is extensible ?*. A positive answer to this question would solve the one dimensional case, even if the SFTs are considered on $\mathbb{N}^2$ instead of $\mathbb{Z}^2$.

Factorization is the notion of surjective morphism adapted to SFTs: when X factors on Y, then Y is a recoding of X, possibly with information loss: the dynamic of Y is "simpler" than X's,*i.e.* it can be deduced from X's. The problem of knowing if some SFT is a factor of another one has also been much studied. In dimension one, it is only partly solved for the case when the entropies of the two SFTs X, Y verify $h(X) > h(Y)$, see [3]. Factor maps have also been studied with the hope of finding universal SFTs: SFTs that can factor on any other and thus contain the dynamics of all of them. However it has been shown that such SFTs do not exist, see [2, 5]. We prove here that it is harder to know if an SFT is a factor of another than to know if it is conjugate to it.

▶ **Theorem 2.** *Given two SFTs X, Y as inputs, it is Σ_3^0-complete to decide if X factors onto Y.*

The last problem we will tackle is the embedding problem, that is to say: when can an SFT be injected into some other SFT? If an SFT X can be injected into another SFT Y, that means that there is an SFT $Z \subseteq Y$ such that X and Z are conjugate. In dimension 1, this problem is also partly solved when the two SFTs X, Y are irreducible and their entropies verify $h(X) > h(Y)$ [8]. We prove here that the problem is Σ_1^0-complete:

▶ **Theorem 3.** *Given two SFTs X, Y as inputs, it is Σ_1^0-complete to decide if X embeds into Y.*

The paper is organised as follows: first we give the necessary definitions and fix the notation is section 1, after what we give the proofs of Theorems 1, 2 and 3 in Sections 2, 3 and 4 respectively.

1 Preliminary definitions

1.1 Subshifts of finite type

We give here some standard definitions and facts about multidimensional subshifts, one may consult Lind [10] or Lind/Marcus [9] for more details.

Let Σ be a finite alphabet, its elements are called *symbols*, the d-dimensional full shift on Σ is the set $\Sigma^{\mathbb{Z}^d}$ of all maps (colorings) from $\mathbb{Z}^d$ to the Σ (the colors). For $v \in \mathbb{Z}^d$, the shift functions $\sigma_v : \Sigma^{\mathbb{Z}^d} \to \Sigma^{\mathbb{Z}^d}$, are defined locally by $\sigma_v(c_x) = c_{x+v}$. The full shift equipped with the distance $d(x, y) = 2^{-\min\{\|v\| | v \in \mathbb{Z}^d, x_v \neq y_v\}}$ is a compact metric space on which the shift functions act as homeomorphisms. An element of $\Sigma^{\mathbb{Z}^d}$ is called a *configuration*.

Every closed shift-invariant (invariant by application of any σ_v) subset X of $\Sigma^{\mathbb{Z}^d}$ is called a *subshift*. An element of a subshift is called a *point* of this subshift.

Alternatively, subshifts can be defined with the help of forbidden patterns. A *pattern* is a function $p : P \to \Sigma$, where P, the *support*, is a finite subset of $\mathbb{Z}^d$. Let $\mathcal{F}$ be a collection of *forbidden* patterns, the subset X_F of $\Sigma^{\mathbb{Z}^d}$ containing the configurations having nowhere a pattern of F. More formally, $X_{\mathcal{F}}$ is defined by

$$X_{\mathcal{F}} = \left\{ x \in \Sigma^{\mathbb{Z}^d} \middle| \forall z \in \mathbb{Z}^d, \forall p \in F, x_{|z+P} \neq p \right\}.$$

In particular, a subshift is said to be a *subshift of finite type* (SFT) when the collection of forbidden patterns is finite. Usually, the patterns used are *blocks* or *r-blocks*, that is they are defined over a finite subset P of $\mathbb{Z}^d$ of the form $B_r = \llbracket -r, r \rrbracket^d$, r is called its *radius*. We may assume that all patterns of $\mathcal{F}$ are defined with blocks of the same radius r, and say the family $\mathcal{F}$ has radius r. We note r_X the radius of the SFT X, the smallest r for which there is a family $\mathcal{F}$ of radius r defining X.

Given a subshift X, a pattern p is said to be *extensible* if there exists $x \in X$ in which p appears, p is also said to be extensible to x. We also say that a pattern p_1 is extensible to a pattern p_2 if p_1 appears in p_2. A block or pattern is said to be *admissible* if it does not contain any forbidden pattern. Note that every extensible pattern is admissible but that the converse is not necessarily true. As a matter of fact, for SFTs, it is undecidable (in Π_1^0 to be precise) in general to know whether a pattern is extensible while it is always decidable efficiently (polytime) to know if a pattern is admissible.

As we said before, SFTs are compact spaces, this gives a link between admissible and extensible: if a pattern appears in an increasing sequence of admissible patterns, then it appears in a valid configuration and is thus extensible. More generally, if we have an increasing sequence of admissible pattern, then we can extract from it a sequence converging to some point of the SFT.

Note that instead of using the formalism of SFTs for the constructions we could have used the formalism of Wang tiles, in which numerous results have been proved. In particular the undecidability of knowing whether an SFT is empty. Since we will use a construction based on Wang tiles, we review their definitions.

Wang tiles are unit squares with colored edges which may not be flipped or rotated. A *tileset* T is a finite set of Wang tiles. A *coloring of the plane* is a mapping $c : \mathbb{Z}^2 \to T$ assigning a Wang tile to each point of the plane. If all adjacent tiles of a coloring of the plane have matching edges, it is called a tiling.

The set of tilings of a Wang tileset is a SFT on the alphabet formed by the tiles. Conversely, any SFT is isomorphic to a Wang tileset. From a recursivity point of view, one can say that SFTs and Wang tilesets are equivalent. In this paper, we will be using both terminologies indiscriminately.

1.2 Conjugacy, Embedding and Factorization

In the rest of the paper, we will use the notation Σ_X for the alphabet of the subshift X.

Let $X \subseteq \Sigma_X^{\mathbb{Z}^2}$ and $Y \subseteq \Sigma_Y^{\mathbb{Z}^2}$ be two subshifts a function $F : X \to Y$ is a block code if there exists a finite set $V = \{v_1, \ldots, v_k\} \subset \mathbb{Z}^2$, the *window*, and a local map $f : \Sigma_X^{|V|} \to \Sigma_Y$, such that for any point $x \in X$ and $y = F(x)$, for all $z \in \mathbb{Z}^d$, $y_z = f(x_{z+v_1}, \ldots, x_{z+v_k})$. That is to say F is defined locally. Without loss of generality, we may suppose that the window is an r-block, r being then called the radius of F and $(2r+1)$ its diameter, we note r_F the radius of F.

A *factorization* or *factor map* is a surjective block code $F : X \to Y$. When the function is injective instead of being surjective, it is called an *embedding*, and we say that X embeds into Y.

When the map F is bijective and invertible and its inverse is also a block code, the subshifts X and Y are said to be *conjugate*. In the rest of the paper, we will note with the same symbol the local and global functions, the context making clear which one is being used.

The entropy of a subshift X is defined as

$$h(X) = \lim_{n\to\infty} \frac{\log E_n(X)}{n^d}$$

where $E_n(X)$ is the number of extensible patterns of X of support $[\![0,n]\!]^d$ where d is the dimension. The entropy is a conjugacy invariant, that is to say, if X and Y are conjugate, then $h(X) = h(Y)$. It is in particular easy to see thanks to the entropy that the full shift on n symbols is not conjugate to the full shift with n' symbols when $n \neq n'$.

1.3 Arithmetical Hierarchy and computability

We give now some background in computability theory and in particular about the arithmetical hierarchy. More details can be found in Rogers [12].

In computability, the arithmetical hierarchy is a classification of sets according to their logical characterization. A set $A \subseteq \mathbb{N}$ is Σ^0_n if there exists a total computable predicate R such that $x \in A \Leftrightarrow \exists\overline{y_1}, \forall\overline{y_2}, \dots, Q\overline{y_n} R(x, \overline{y_1}, \dots, \overline{y_n})$, where Q is a $\forall$ or an $\exists$ depending on the parity of n. A set A is Π^0_n if there exists a total computable predicate R such that $x \in A \Leftrightarrow \forall\overline{y_1}, \exists\overline{y_2}, \dots, Q\overline{y_n} R(x, \overline{y_1}, \dots, \overline{y_n})$, where Q is a $\forall$ or an $\exists$ depending on the parity of n. Equivalently, a set is Σ^0_n iff its complement is Π^0_n.

We say a set A is many-one reducible to a set B, $A \leq_m B$ if there exists a computable function f such that for any x, $f(x) \in A \Leftrightarrow x \in B$. Given an enumeration of Turing Machines M_i with oracle X, the Turing *jump* X' of a set X is the set of integers i such that M_i halts on input i. We note $X^{(0)} = X$ and $X^{(n+1)} = (X^{(n)})'$. In particular $0'$ is the set of halting Turing machines.

A set A is Σ^0_n-hard (resp. Π^0_n) iff for any Σ^0_n (resp. Π^0_n) set B, $B \leq_m A$. The problem $0^{(n)}$ is Σ^0_n-complete. Furthermore, it is Σ^0_n-complete if it is in Σ^0_n. The sets in Σ^0_1 are also called recursively enumerable and the sets in Π^0_1 are called the co-recursively enumerable or effectively closed sets.

2 Conjugacy

We prove here the Σ^0_1-completeness of the conjugacy problem in dimension $d \geq 2$, even for a fixed SFT. We first prove the following lemma, which is the first step to show that conjugacy is Σ^0_1 and also proves that equality is Σ^0_1.

▶ **Lemma 4.** *Given F, X, Y as an input, deciding if $F(X) \subseteq Y$ is Σ^0_1.*

Proof. It is clear that $F(X) \subseteq Y$ if and only if $F(X)$ does not contain any configuration where a forbidden patterns of Y appears. We now show that this is equivalent to the following Σ^0_1 statement: *there exists a radius $r > \max(r_F + r_Y, r_X)$ such that for any admissible r-block M of X, $F(M)$ does not contain any forbidden pattern in its center.*

We prove the result by contraposition, in both directions. Suppose there is a configuration $x \in X$ such that $F(x)$ contains a forbidden pattern. Then for any radius $r > \max(r_F + r_Y, r_X)$, there exists an extensible, hence admissible, pattern M of size r such that $F(M)$ contains a forbidden pattern in its center.

Conversely, if for any radius $r > \max(r_F + r_Y, r_X)$, there exists an admissible pattern M of X of size r such that $F(M)$ contains a forbidden pattern in its center, then by compactness, there exists a configuration $x \in X$ such that $F(x)$ contains a forbidden pattern in its center. ◀

▶ **Corollary 5.** *Given two SFTs X, Y as an input, it is Σ_1^0 to decide if $X = Y$.*

▶ **Theorem 6.** *Given two SFTs X, Y as an input, it is Σ_1^0 to decide whether X and Y are conjugate.*

Proof. To decide whether two SFTs X and Y are conjugate, we have to check whether there exists two local functions $F : \Sigma_X^{B_{r_F}} \to \Sigma_Y$ and $G : \Sigma_Y^{B_{r_G}} \to \Sigma_X$ such that the global functions associated verify $F_{|X} \circ G_{|Y} = id_{|Y}$ and $G_{|Y} \circ F_{|X} = id_{|X}$. These functions being local, we can guess them with a first order existential quantifier. We prove that X and Y are conjugate if and only if the following Σ_1^0 statement is true :

There exist F, G and $k > \max(r_X + r_Y) + r_F + r_G$ such that $F(X) \subseteq Y$ and $G(Y) \subseteq X$ and :

- *for all k-block b, if b is admissible for X, then $G \circ F(b)_0 = b_0$*
- *for all k-block b, if b is admissible for Y, then $F \circ G(b)_0 = b_0$*

We only prove the statement for $G \circ F$ the other one being identical. The proof is by contraposition in both directions :

- Let $x \in X$ be a point such that $G \circ F(x) \neq x$, we may suppose that the difference is in 0 by shifting. For all k, there exists an extensible pattern b of size k such that $G \circ F(x)_0 \neq b_0$.
- Conversely, if there exists a sequence b_k of admissible k-blocks such that $G \circ F(b_k)_0 \neq (b_k)_0$, then by compactness we can extract a subsequence converging to some point $x \in X$ which by construction is different from its image by $G \circ F$ in 0.

As we have seen in Lemma 4 that checking whether $F(X) \subseteq Y$ is Σ_1^0, we have the desired result. ◀

▶ **Theorem 7.** *For any X, given Y as an input, it is Σ_1^0-hard to decide if X and Y are conjugate (resp. equal).*

Proof. We reduce the problem from $0'$, the halting problem. Given a Turing machine M we construct a SFT Y_M such that Y_M is conjugate to X iff M halts.

Let R_M be Robinson's SFT [11] encoding computations of M: R_M is empty iff M halts[1].

Now take the full shift on one more symbol than X, note it F. Let Y_M be now the disjoint union of X and $R_M \times F$.

If M halts, $Y_M = X$ and hence is conjugate to X. In the other direction, suppose M does not halt, then $R_M \times F$ has entropy strictly greater than that of X and hence Y_M is not conjugate to X. ◀

▶ **Corollary 8.** *Given two SFTs X, Y as an input, it is Σ_1^0-hard to decide if $X = Y$.*

3 Factorization

We start with two small examples to see why factorization is more complex than conjugacy. Here the examples are the simplest ones possible: we fix the SFT to which we factor in a very simple way, thus making the factor map known in advance.

[1] Robinson's SFT is in dimension 2 of course, for higher dimensions, we take the rules that the symbol in $x \pm e_i$ equals the symbol in x, for $i > 2$.

▶ **Theorem 9.** *Let Y be the SFT containing exactly one configuration, a uniform configuration. Given X as an input, it is Π^0_1-complete to know whether X factors onto Y.*

Proof. In this case the factor map is forced: it has to send everything to the only symbol of Σ_Y. And the problem is hence equivalent to knowing whether a SFT is *not* empty, which is Π^0_1-complete. ◀

▶ **Theorem 10.** *Let Y be the empty SFT. Given X as an input, it is Σ^0_1-complete to know whether X factors onto Y.*

Proof. Here any factor map is suitable, the problem is equivalent to knowing whether X is empty, which is Σ^0_1-complete. ◀

We study now the hardness of factorization in the general case, that is to say when two SFTs are given as inputs and we want to know whether one is a factor of the other. We prove here with Theorems 11 and 15 the Σ^0_3-completeness of the factorization problem.

3.1 Factorization is in Σ^0_3

▶ **Theorem 11.** *Given two SFTs X, Y as an input, deciding whether X factors onto Y is in Σ^0_3.*

Proof. The shift X factors onto Y iff there exists a factor map F, a local function, such that $F(X) = Y$. This is the first existential quantifier.The result follows from the next lemma and Lemma 4. ◀

▶ **Lemma 12.** *Given two SFTs X, Y and a local map F as an input, deciding if $Y \subseteq F(X)$ is Π^0_2.*

Proof. We prove here that the statement $Y \subseteq F(X)$, that is to say, *for every point $y \in Y$, there exists a point $x \in X$ such that $F(x) = y$*, is equivalent to the following Π^0_2 statement: *for any admissible pattern m of Y, if m is extensible, then $F^{-1}(m)$ contains an admissible pattern.* This statement is Π^0_2 since checking that m is *not* extensible is Σ^0_1, that is to say: there exists a radius r such that all r-blocks containing m are not admissible.

We now prove the equivalence. Suppose that $Y \subseteq F(X)$, then any extensible pattern m of Y appears in a configuration $y \in Y$ which has a preimage $x \in X$. A preimage of m being extensible, it is also admissible. This proves the first direction.

Conversely, suppose all extensible patterns m of Y have an admissible preimage. Let y be a point of Y, then we have an increasing sequence m_i of extensible patterns converging to y. All of them have at least one admissible preimage m'_i. By compactness, we can extract from this sequence a converging subsequence, note x its limit. By construction x is a point of X and a preimage of y. ◀

3.2 Factorization is Σ^0_3-hard

To prove the hardness, we use the base construction that we introduced in [6]: we note it T. This construction introduces a new way to put Turing machine computations in SFTs, in particular, the base construction has exactly one point (up to shift) in which computations may be encoded. We call this point *configuration α*, its schematic view is shown in Figure 2a. The computation is encoded in the inner grid which is sparse. Each crossing between a horizontal line and a vertical one forms a *cell*. The constraints are carried along the vertical and horizontal lines, so that we may view the encoding of the Turing machine as a tiling on the grid. For each time step, the tape of the Turing machine is encoded in the NW-SE diagonals and the size of the diagonal steadily increases in size when going north-east. At each growth of the diagonal size, it gains two cells.

Configuration α is made of two layers: one producing the horizontal lines and the other the vertical ones. The layer producing the vertical lines is shown in Figure 1, the vertical lines are the black vertical lines. The configuration producing the horizontal lines is its exact symmetric along the south-west/north-east diagonal. The key property of these layers is that when a corner tile (the tile in the lower left corner of the first square) appears, then the point is necessarily of this form.

In the original construction, corner tiles of the horizontal and vertical layers could only be superimposed to each other. We just change this so that instead, the corner tile of the vertical layer has to be at position $(1, -1)$ relative to the corner tile of the horizontal one. This change does not impact any of the properties of T, but simplifies a bit the proof of Lemma 14.

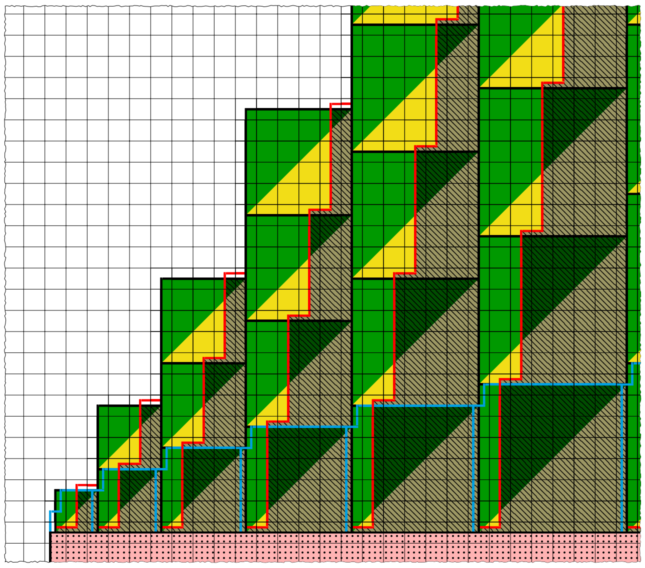


Figure 1 The vertical layer of point α, the meaningful point of X_T. The corner tile may be seen on the first non all-white column: it is the lower left corner of the square.

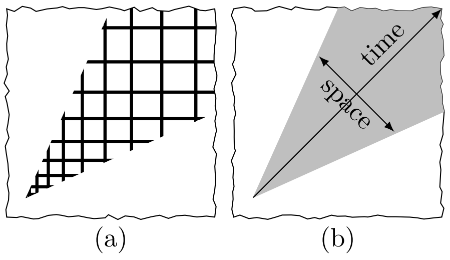


Figure 2 (a) The skeleton of configuration α. (b) How the computation is superimposed to α.

Our reduction will use two SFTs based on this construction, both of them will be feature a different tiling on its grid. We will say that an SFT which is basically T with a tiling on its grid as having T-structure.

▶ **Definition 13** (T-structure)**.** We say an SFT X has *T-structure* if it is a copy of T to which we superimposed new symbols only on the symbols representing the horizontal/vertical lines and their crossings.

Note that an SFT may have T-structure while having no α-configuration: for instance if you put a computation of a Turing machine that produces an error whenever it halts.

The next lemma states a very intuitive result, that will be used later, namely that if an SFT with T-structure factors to another one, then the structure of each point is preserved by factorization. Furthermore, it shows that the factor map can only send a cell to its corresponding one, that is to say cell of the preimage has to be in the window of the image.

▶ **Lemma 14.** *Let X, Y be two SFTs with T-structure, such that X factors onto Y. Let r be the radius of the factor map, then any α-configuration of Y is factored on by an α-configuration of X shifted by v, with $\|v\|_\infty \leq r$.*

Proof. By [6, Lemma 1], we know that non-α-configurations have at most one vertical line and one horizontal line. And therefore that they have two uniform (same symbols everywhere) quarter-planes and four uniform eighth-planes, as seen on Figure 3. The two north east eighth-planes are not uniform in α. Thus they cannot be factored on α.

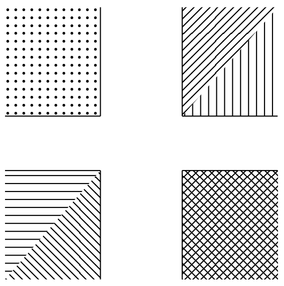


Figure 3 Uniform quarter- and eighth-planes in non-α-configurations.

It remains to prove the second part: that in the factoring process the α-structure is at most shifted by the radius of the factorization. We do that by contradiction, suppose that an α-configuration x of X is mapped to an α-configuration y of Y and shifts it by $v = (v_x, v_y)$, with $\|v\|_\infty > r$. Without loss of generality we may suppose that $v_x > r$ and $v_y > 0$ and that the vertical and horizontal corner tiles of the preimage are at positions $(0, 1)$ and $(1, 0)$ respectively. We are now going to show that this is not possible.

On the horizontal layer, for all $k \in \mathbb{N}^*$ there is a square with lower left corner at $(2k^2 + k, 2k^2 + k)$, see Figure 1. Inside this square, there are two $(k-1) \times (k-1)$ uniform smaller squares, see Figure 4. This being also true for the vertical layer, these squares remain uniform when they are superimposed. Now take k such that $k > (\|v\|_\infty + 2r + 1)$. By hypothesis, there is a vertical line symbol t at $z_p = (2k^2 + 2k + 1, 2k^2 + k)$ on x, and thus at $z_i = (2k^2 + 2k + 1 + v_x, 2k^2 + k + v_y)$ on y. We know $x_{|z_i+B_r}$ has image t, and by what precedes that $x_{|z_i+B_r} = x_{|z_i+(1,0)+B_r}$ since they are both uniform, therefore, there should be two t symbols next to eachother in y at z_i and $z_i + (1, 0)$. This is impossible.

◀

▶ **Theorem 15.** *Given two SFTs X, Y as an input, deciding whether X factors onto Y is Σ_3^0-hard.*

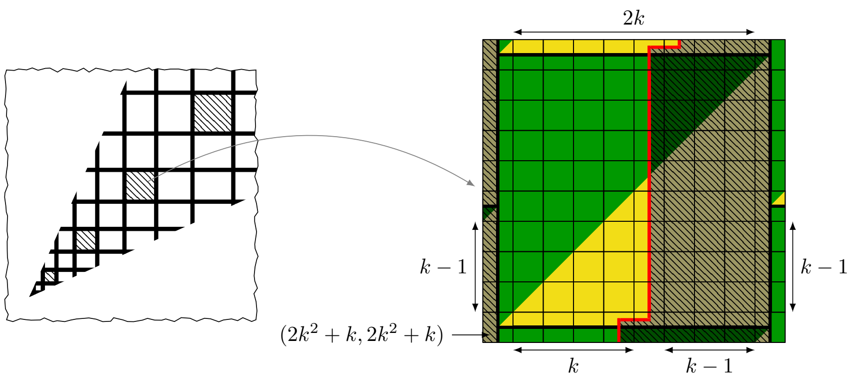


Figure 4 For every $k \in \mathbb{N}^*$, the square starting at position $(2k^2+k, 2k^2+k)$ is of the form on the right on the component producing the vertical lines (and is the symmetric along the diagonal for the one producing the horizontal lines). We can see that there are two uniform $(k-1)\times(k-1)$ squares at $(2k^2+2k+2, 2k^2+k+1)$ and $(2k^2+k+1, 2k^2+2k+2)$ respectively.

For this proof, we will reduce from the problem **COFINITE**, which is known to be Σ_3^0-complete, see Kozen [7]. **COFINITE** is the set of Turing machines which run infinitely only on a finite set of inputs.

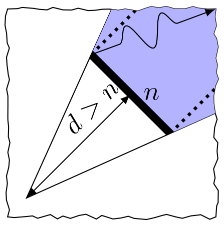


Figure 5 Computation on input n in the SFT Z, the number of white diagonals d preceeding the computation is strigtly greater than the input n.

Proof. Given a Turing machine M, we construct two SFTs X_M and Y_M such that X_M factors on Y_M iff the set of inputs on which M does not halt is finite. We first introduce an SFT Z_M on which both will be based. It will have T structure. Above the T base, we allow the cells of the grid to be either white or blue according to the following rules:

- All cells on a NW-SE diagonal are of the same color.
- A blue diagonal may follow (along direction SW-NE) a white diagonal, but not the contrary.
- A transition from white diagonal to blue may only appear when the grid grows.

We now allow computation on blue cells only. Only the diagonals after the growth of the grid may contain computations. The Turing machine M is launched on the input formed by the size of the first blue line (in number of cells). We forbid the machine to halt.

So for each n on which M does not halt, there is a configuration with white cells until the first blue diagonal appears, then computation occurs inside the blue cone, see Figure 5 for a schematic view. If M halts on n, then there is no tiling where the first blue line codes n. By compactness, there is of course a configuration with only white diagonals. If M is total, then the only α-configuration in Z_M is the one with only white diagonals.

Now from Z_M, we can give X_M and Y_M:

- X_M: Let Z'_M be a copy of Z_M to which we add two decorations 0 and 1 on the blue cells only, and all blue cells in a configuration must have the same decoration. Now X_M is Z'_M to which we add a third color, red, that may only appear alone, instead of white and blue. No computation is superimposed on red.
- Y_M is a copy of Z_M where we decorated only the horizontal corner tile with two symbols 0 and 1.

We now check that X_M factors onto Y_M iff M does not halt on a finite set of inputs:

$\Rightarrow$ Suppose M does not halt on a finite set of inputs: there exists N such that M halts on every input greater than N. The following factor map F works:

- F is the identity on Z_M. Note that the additional copy of T is also sent to the component Z_M.
- F has a radius big enough so that when its window is centered on the corner tile, it would cover the beginning of the computation on input N.
- An α-configuration x of X_M is sent on the same α-configuration y in Y_M. For the decorations, when there is a computation on x, the factor map can see it and gives the same decoration to the corner tile of y. When there is no computation, the factor map doesn't see a computation zone and gives decoration 0 to the corner tile. The configuration with only white diagonals and decoration 1 of Y_M is factored on by the α-configuration colored in red contained in X_M.

Note that this also works when M is total.

$\Leftarrow$ Conversely, suppose M does not halt on an infinite set of inputs, and that there exists a factor map F with radius r: Lemma 14 states that all α-configurations of Y_M are factored on by α-configurations of X_M. Now, there is an infinite number of α-configurations with corner tile decorated with 0 (resp. 1) in Y_M, they all must be factored on by some α-configuration of X_M. Still by Lemma 14, the corner tile of the preimage must be in the window of the corner tile of the image. However, there can only be a finite number of configurations in which the symbols in this window differ. So the α-configurations of X_M factor to a finite number of α-configurations of Y_M with one of the decorations. This is impossible.

Note that the construction of X_M and Y_M from the description of M is computable and uniform. The reduction is thus many-one. ◀

4 Embedding

We prove now Theorem 3 stating that the embedding problem is Σ^0_1-complete. We start with an analogue of Lemma 14 :

▶ **Lemma 16.** *Let X, Y be two SFTs with T-structure, such that X embeds into Y. Let r be the radius of the embedding, then any α-configuration of X is mapped to an α-configuration of Y shifted by v, with $\|v\|_\infty \leq r$.*

Proof. First note that the uniform points of X must be mapped to uniform points of Y. So all different uniform points, and thus all uniform patterns of support B_r, have different images. Now an α-configuration of X has arbitrarily large uniform areas, as seen in Lemma 14, see also Figure 4. These uniform areas alternate, so their image also alternates when they are sufficiently large. The only configurations that have growingly large alternating uniform areas are α-configurations. So α-configurations of X are mapped to α-configurations of Y. The proof that these mappings do not shift the T-structure by more than r is exactly the same as in Lemma 14. ◀

▶ **Lemma 17.** *Let X and Y be two SFTs, it is Σ_1^0 to check whether X embeds into Y.*

Proof. To decide whether X embeds into Y, we have to check if there exists an injective local function $F : X \to Y$. Such a function being local, it can be guessed with a first order existential quantifier. To check that it is an embedding, we have to check that $F(X) \subseteq Y$ and that for all $x_1, x_2 \in X$, $x_1 \neq x_2 \Rightarrow F(x_1) \neq F(x_2)$. We know from Lemma 4 that checking $F(X) \subseteq Y$ is Σ_1^0. We now show that the second part is also Σ_1^0 by showing that the two following statements are equivalent.

- There exist $x_1, x_2 \in X$ such that $x_1 \neq x_2$ and $F(x_1) = F(x_2)$.
- For all $r > \max(r_F, r_X)$, there exist two admissible r-blocks M_1, M_2 such that $(M_1)_0 \neq (M_2)_0$ and $F(M_1) = F(M_2)$.

It is clear that the second statement is Π_1^0 and that the first statement is the negation of the definition of injectivity. Now to the proof :

- Suppose there exist two different points $x_1, x_2 \in X$ such that $F(x_1) \neq F(x_2)$, we may assume x_1 and x_2 differ in 0 by shifting. For all $r > \max(r_F, r_X)$, the central r-blocks M_1, M_2 of x_1, x_2 are admissible and differ in 0
- Suppose now that for all $r > \max(r_F, r_X)$ there exist two admissible r-blocks M_1^r, M_2^r differing in 0 and such that $F(M_1^r) = F(M_2^r)$. By the pigeonhole principle, there is an infinity of M_1^r which have the same symbol in 0 and thus of M_2^r without this symbol in 0. Take these subsequences of M_1^r and M_2^r, by compactness we can extract converging subsequences from them which converge to two points $x_1, x_2 \in X$ with different symbols in 0. These two points have the same image, by construction.

◀

▶ **Lemma 18.** *Given two SFTs X, Y as an input, deciding whether X embeds into Y is Σ_1^0-hard.*

We will use a reduction from the halting problem, the set of Turing machines that halt on a blank input, and a construction based on a T-structure, as before.

Proof. Given a Turing machine M, we construct two SFTs X_M and Y_M such that X_M embeds into Y_M iff the Turing machine M halts. Both SFTs have as a base an SFT Z_M with a T-structure, in which we encode computations of M. Let us describe Z_M : Z_M is only T on which we directly encode the computation of M, it may eventually reach a halting state in which case the remaining space is given a new color, say blue. So our SFT Z_M can take two different forms : if the machine M halts, then a blue zone appears, if it does not halt, then this zone does not appear.

- Now X_M is Z_M for which we add a decoration to the corner tile, 0 or 1, so there are two different grid points in any case, whether the machine M halts or not.
- Y_M is Z_M for which we add a decoration to the halting state only (it appears at most once), there are two different grid points only when the machine M halts.

Let us check now that X_M embeds into Y_M if and only if M halts.

$\Rightarrow$ When the machine M halts, X_M embeds into Y_M : the radius of the embedding r is the distance between the halting state and the corner, the decoration of the corner is just translated to the halting state. All the rest remains unchanged. Note that there are less non α-configurations in X_M than in Y_M : these are the configurations containing an infinite cross of black lines with a halting state on top. They have different decorations in Y_M but not in X_M.

$\Leftarrow$ When the machine M does not halt, there are two different α-configurations in X_M up to shift, while there is only one in Y_M, so there are two that must have the same image.

◀

References

1 Robert Berger. *The Undecidability of the Domino Problem.* PhD thesis, Harvard University, 1964.
2 Laurent Boyer and Guillaume Theyssier. On factor universality in symbolic spaces. In *MFCS*, pages 209–220, 2010.
3 Mike Boyle. Lower entropy factors of sofic systems. *Ergodic Theory and Dynamical Systems*, 3:541–551, 1983.
4 Mike Boyle. Open Problems in Symbolic Dynamics. *Contemporary Mathematics*, 469:69–118, 2008.
5 Michael Hochman. A note on universality in multidimensional symbolic dynamics. *Discrete and Continuous Dynamical Systems S*, 2(2), 2009.
6 Emmanuel Jeandel and Pascal Vanier. Π_1^0 sets and tilings. In *Theory and Applications of Models of Computation (TAMC)*, volume 6648 of *Lecture Notes in Computer Science*, pages 230–239, 2011.
7 Dexter Kozen. *Theory of Computation.* Springer, New York, 2006.
8 Wolfgang Krieger. On the subsystems of topological markov chains. *Ergodic Theory and Dynamical Systems*, 2(02):195–202, 1982.
9 Douglas Lind and Brian Marcus. *An introduction to symbolic dynamics and coding.* Cambridge University Press, New York, NY, USA, 1995.
10 Douglas A. Lind. Multi-Dimensional Symbolic Dynamics. In Susan G. Williams, editor, *Symbolic Dynamics and its Applications*, number 60 in Proceedings of Symposia in Applied Mathematics, pages 61–79. American Mathematical Society, 2004.
11 Raphael M. Robinson. Undecidability and Nonperiodicity for Tilings of the Plane. *Inventiones Math.*, 12, 1971.
12 Hartley Rogers, Jr. *Theory of recursive functions and effective computability.* MIT Press, Cambridge, MA, USA, 1987.
13 Hao Wang. Proving theorems by Pattern Recognition II. *Bell Systems technical journal*, 40:1–41, 1961.
14 R. F. Williams. Classification of subshifts of finite type. *Annals of Mathematics*, 98:120–153, 1973.

The finiteness of a group generated by a 2-letter invertible-reversible Mealy automaton is decidable

Ines Klimann*

Univ Paris Diderot, Sorbonne Paris Cité, LIAFA, UMR 7089 CNRS, F-75013 Paris, France
klimann@liafa.univ-paris-diderot.fr

Abstract

We prove that a semigroup generated by a reversible two-state Mealy automaton is either finite or free of rank 2. This fact leads to the decidability of finiteness for groups generated by two-state or two-letter invertible-reversible Mealy automata and to the decidability of freeness for semigroups generated by two-state invertible-reversible Mealy automata.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases Mealy automata, automaton semigroups, decidability of finiteness, decidability of freeness, Nerode equivalence

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.502

1 Introduction

Automaton (semi)groups — short for semigroups generated by Mealy automata and groups generated by invertible Mealy automata — were formally introduced half a century ago (for details, see [14, 9] and references therein). Over the years, important results have started revealing their full potential, by contributing to important conjectures in group theory, as Milnor problem (first example of a group of intermediate growth) or Burnside problem (example of a very simple Mealy automaton generating an infinite torsion group).

In a way, semigroups can be classified according to their growth function: at one end stand finite semigroups and at the other one free semigroups. Several sufficient or necessary criteria for finiteness of automaton semigroups exist [2, 15, 9, 16, 17, 22, 4, 8, 21], but deciding finiteness of such semigroups is still an open problem. As to freeness, it has been and it is still a challenge: only some particular invertible Mealy automata, possibly parametrized, have been shown to generate free groups [23, 11, 19, 24, 25]; and some Cayley automaton semigroups have been shown to be free [22].

In this paper, we link both issues for semigroups generated by reversible two-state Mealy automata: we prove that such semigroups are either finite or free, in this latter case the states of the generating Mealy automaton being free generators of the semigroup, answering a conjecture stated in [15]. On the basis of this dichotomy between finite and free semigroups, we prove that finiteness and freeness of the semigroup are decidable if the generating reversible two-state Mealy automaton is also invertible. Decidability of finiteness extends by duality to groups generated by two-letter invertible-reversible Mealy automata. The problems of deciding finiteness or freeness of automaton semigroups was raised by Grigorchuk, Nekrashevych, and Sushchanskii [14, Problem 7.2.1(b)].

* The author is partially supported by ANR Project MealyM ANR-JCJC-12-JS02-012-01.

Specializing to two letters or states may seem to be a strong restriction, but most of the significant examples in literature have faced this restriction: the first example of a finitely generated group of intermediate growth, the Grigorchuk group [13, 14], is generated by a two-letter Mealy automaton while the very smallest Mealy automaton with intermediate growth [6] has two letters and two states; the lamplighter group [12] is generated by a two-letter and two-state Mealy automaton; the Aleshin automaton [3, 24] gives the simplest example of a free automaton group and has two letters. The article [7] is entirely devoted to the study of groups generated by 3-state 2-letter invertible Mealy automata.

This paper is organized as follows. In Section 2 we define Mealy automata and automaton (semi)groups. Basic tools to manipulate them are introduced in Section 3. Section 4 is devoted to the dichotomy between free and finite semigroups. The decidability results are proved in Section 5. The cornerstone of our proofs and constructions is the very classical Nerode equivalence used to minimize automata.

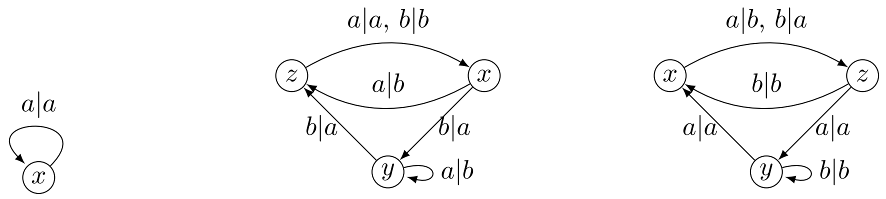
2 (Semi)groups generated by Mealy automata

2.1 Mealy automata

If one forgets initial and final states, a *(finite, deterministic, and complete) automaton* $\mathcal{A}$ is a triple $\big(A, \Sigma, \delta = (\delta_i : A \to A)_{i\in\Sigma}\big)$, where the *stateset* A and the *alphabet* Σ are non-empty finite sets, and where the δ_i are functions.

A *Mealy automaton* is a quadruple $\big(A, \Sigma, \delta = (\delta_i : A \to A)_{i\in\Sigma}, \rho = (\rho_x : \Sigma \to \Sigma)_{x\in A}\big)$, such that both (A, Σ, δ) and (Σ, A, ρ) are automata. In other terms, a Mealy automaton is a letter-to-letter transducer with the same input and output alphabet.

The graphical representation of a Mealy automaton is standard, see Figure 1.



(a) The trivial aut. **(b)** The Aleshin automaton. **(c)** The Baby-Aleshin aut.

Figure 1 Examples of Mealy automata: the Aleshin automaton generates the rank 3 free group [3, 24], the Baby-Aleshin automaton generates the free product $\mathbb{Z}_2^{*3} = \mathbb{Z}_2 * \mathbb{Z}_2 * \mathbb{Z}_2$ [19].

A Mealy automaton $\mathcal{A} = (A, \Sigma, \delta, \rho)$ is *invertible* if the functions ρ_x are permutations of Σ and *reversible* if the functions δ_i are permutations of A.

In a Mealy automaton $\mathcal{A} = (A, \Sigma, \delta, \rho)$, the sets A and Σ play dual roles. So we may consider the *dual (Mealy) automaton* defined by $\mathfrak{d}(\mathcal{A}) = (\Sigma, A, \rho, \delta)$. Obviously, a Mealy automaton is reversible if and only if its dual is invertible.

Considering the underlying graph of a Mealy automaton, it makes sense to look at the connected components of a Mealy automaton. Note that a connected component of a reversible Mealy automaton is always strongly connected: its $(\delta_i : A \to A)_{i\in\Sigma}$ are permutations of a finite set and in particular they are surjective.

2.2 Automaton (semi)groups

Let $\mathcal{A} = (A, \Sigma, \delta, \rho)$ be a Mealy automaton. We view $\mathcal{A}$ as an automaton with an input and an output tape, thus defining mappings from input words over Σ to output words over Σ.

Formally, for $x \in A$, the map $\rho_x : \Sigma^* \to \Sigma^*$, extending $\rho_x : \Sigma \to \Sigma$, is defined by:

$$\forall i \in \Sigma,\ \forall \mathbf{s} \in \Sigma^*, \qquad \rho_x(i\mathbf{s}) = \rho_x(i)\rho_{\delta_i(x)}(\mathbf{s}) \ .$$

By convention, the image of the empty word is itself. The mapping ρ_x is length-preserving and prefix-preserving. We say that ρ_x is the *production function* associated with $(\mathcal{A}, x)$ or more briefly, if there is no ambiguity, the *production function* of x. For $\mathbf{x} = x_1 \cdots x_n \in A^n$ with $n > 0$, set $\rho_{\mathbf{x}} : \Sigma^* \to \Sigma^*, \rho_{\mathbf{x}} = \rho_{x_n} \circ \cdots \circ \rho_{x_1}$.

Denote dually by $\delta_i : A^* \to A^*, i \in \Sigma$, the production functions associated with the dual automaton $\mathfrak{d}(\mathcal{A})$. For $\mathbf{s} = s_1 \cdots s_n \in \Sigma^n$ with $n > 0$, set $\delta_{\mathbf{s}} : A^* \to A^*,\ \delta_{\mathbf{s}} = \delta_{s_n} \circ \cdots \circ \delta_{s_1}$.

The semigroup of mappings from Σ^* to Σ^* generated by $\rho_x, x \in A$, is called the *semigroup generated by* $\mathcal{A}$ and is denoted by $\langle \mathcal{A} \rangle_+$. When $\mathcal{A}$ is invertible, its production functions are permutations on words of the same length and thus we may consider the group of mappings from Σ^* to Σ^* generated by $\rho_x, x \in A$; it is called the *group generated by* $\mathcal{A}$ and is denoted by $\langle \mathcal{A} \rangle$.

An invertible Mealy automaton generates a finite group if and only if it generates a finite semigroup [2]. A Mealy automaton generates a finite semigroup if and only if so does its dual [19, 20, 2].

3 Basic tools

In this section, we present basic tools to manipulate Mealy automata: Nerode equivalence and minimization of automata (§ 3.1) are classic constructions from automata theory, $\mathfrak{md}$-reduction and $\mathfrak{md}$-triviality (§ 3.2) have been introduced in [2] to give a sufficient condition for finiteness, portraits of automorphisms on a regular rooted tree (§ 3.3) come from geometric group theory and tensor closures (§ 3.4) are newly introduced in order to better control the structure of a Mealy automaton.

Let $\mathcal{A} = (A, \Sigma, \delta, \rho)$ be a Mealy automaton. A convenient and natural operation is to raise $\mathcal{A}$ to the power n, for some $n > 0$: its *n-th power* is the Mealy automaton

$$\mathcal{A}^n = \big(\ A^n, \Sigma, (\delta_i : A^n \to A^n)_{i \in \Sigma}, (\rho_{\mathbf{u}} : \Sigma \to \Sigma)_{\mathbf{u} \in A^n}\ \big) \ .$$

Note that the powers of a reversible Mealy automaton are reversible.

3.1 Nerode equivalence and minimization of a Mealy automaton

Throughout this subsection, $\mathcal{A} = (A, \Sigma, \delta, \rho)$ denotes a Mealy automaton.

The *Nerode equivalence* $\equiv$ *on* A is the limit of the sequence of increasingly finer equivalences $(\equiv_k)$ recursively defined by:

$$\begin{aligned} \forall x, y \in A, \qquad\qquad & x \equiv_0 y \iff \rho_x = \rho_y \ , \\ \forall k \geqslant 0,\ & x \equiv_{k+1} y \iff \big(x \equiv_k y \quad \wedge \quad \forall i \in \Sigma,\ \delta_i(x) \equiv_k \delta_i(y)\big) \ . \end{aligned}$$

Since the set A is finite, this sequence is ultimately constant; moreover if two consecutive equivalences are equal, the sequence remains constant from this point on. The limit is therefore computable. For every element x in A, we denote by $[x]$ (*resp.* $[x]_k$) the class of x w.r.t. the Nerode equivalence (*resp.* the $\equiv_k$ equivalence), called the *Nerode class* (*resp.* the *k-class*) of x. Extending to the n-th power of $\mathcal{A}$, we denote by $[\mathbf{x}]$ the Nerode class in A^n of $\mathbf{x} \in A^n$.

The *minimization* of $\mathcal{A}$ is the Mealy automaton $\mathfrak{m}(\mathcal{A}) = (A/{\equiv}, \Sigma, \tilde{\delta}, \tilde{\rho})$, where for every (x, i) in $A \times \Sigma$, $\tilde{\delta}_i([x]) = [\delta_i(x)]$ and $\tilde{\rho}_{[x]} = \rho_x$. This definition is consistent with the standard minimization of "deterministic finite automata" where instead of considering the mappings $(\rho_x : \Sigma \to \Sigma)_x$, the computation is initiated by the separation between terminal and non-terminal states. Using the Hopcroft algorithm, the time complexity of minimization is $\mathcal{O}(\Sigma A \log A)$, see [1] – E being used here instead of $\#E$, for a set E, to simplify notations.

Two states of a Mealy automaton belong to the same Nerode class if and only if they represent the same element in the generated semigroup, i.e. if and only if they have the same production function $\Sigma^* \to \Sigma^*$. Two words on A of the same length n are *equivalent* if they belong to the same Nerode class in A^n. By extension, any two words on A are *equivalent* if they have the same production function. The set of all words equivalent to $\mathbf{x} \in A^*$, regardless of their length, is denoted by $[\![\mathbf{x}]\!]$.

Two states of a Mealy automaton belong to the same k-class if and only if the restrictions of their production functions to $\Sigma^k \to \Sigma^k$ are equal.

The following remarks will be useful for the rest of the paper:

▶ **Remark 1.** *Let n be an integer. If each word of A^n is equivalent to a strictly shorter word, then the semigroup $\langle \mathcal{A} \rangle_+$ is finite, its set of elements being $\{\rho_{\mathbf{u}}, \mathbf{u} \in A^{\leq n-1}\}$.*

▶ **Remark 2.** *If two words of A^* are equivalent, so are their images under the action of each element of $\langle \mathfrak{d}(\mathcal{A}) \rangle_+$.*

3.2 $\mathfrak{md}$-reduction and $\mathfrak{md}$-triviality

The $\mathfrak{md}$-reduction and the $\mathfrak{md}$-triviality were introduced in [2] to give a sufficient but not necessary condition of finiteness. We show in Section 5 that, in the case of a two-state or two-letter invertible-reversible Mealy automaton, this condition is actually necessary.

A pair of dual Mealy automata is *reduced* if both automata are minimal. The $\mathfrak{md}$-*reduction* of a Mealy automaton consists in minimizing the automaton or its dual until the resulting pair of dual Mealy automata is reduced. It is well-defined: if both a Mealy automaton and its dual automaton are non-minimal, the reduction is confluent [2].

The trivial Mealy automaton (see Figure 1(a)) generates the trivial (semi)group. If the $\mathfrak{md}$-reduction of a Mealy automaton $\mathcal{A}$ leads to the trivial Mealy automaton, $\mathcal{A}$ is said to be $\mathfrak{md}$-*trivial*. It is decidable whether a Mealy automaton is $\mathfrak{md}$-trivial. An $\mathfrak{md}$-trivial Mealy automaton generates a finite semigroup, but in general the converse is false [2].

A priori the sequence of minimization-dualization can be arbitrarily long: the minimization of a Mealy automaton with a minimal dual can make the dual automaton non-minimal. Nevertheless, if the automaton has two states, the $\mathfrak{md}$-reduction can be shortened to $\mathfrak{mdmd}$. Hence, in this particular case, the time complexity of the $\mathfrak{md}$-reduction is $\mathcal{O}(\Sigma \log \Sigma)$.

3.3 Portrait of a word

Throughout this subsection, $\mathcal{A} = (A, \Sigma, \delta, \rho)$ denotes an invertible Mealy automaton.

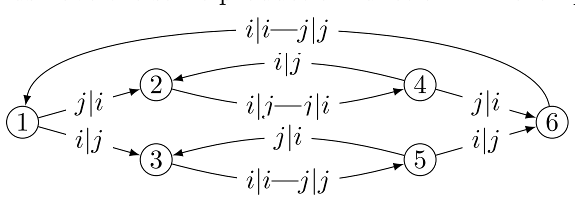
The set Σ^* can naturally be thought of as a regular rooted tree; its root is the empty word and two words are connected if and only if they are of the form $\mathbf{s}$ and $\mathbf{s}i$, with $\mathbf{s} \in \Sigma^*$, $i \in \Sigma$. The set Σ^n is the *nth level* of Σ^*. A *branch* of the tree Σ^* is a sequence of words $(\mathbf{s}_k)_{k \in \mathbb{N}}$ such that, for each $k \in \mathbb{N}$, $\mathbf{s}_k$ is of length k and is a prefix of $\mathbf{s}_{k+1}$.

An *automorphism* of Σ^* is a bijective map $\Sigma^* \to \Sigma^*$ preserving the root and the adjacency of the vertices. Each state x of the automaton $\mathcal{A}$ acts on the regular rooted tree Σ^* by the production rule ρ_x. The constructions of this subsection are directly inspired by this view (see [19] and references therein for more details on automorphisms acting on regular rooted trees). Denote by $Aut(\Sigma^*)$ the set of automorphisms of Σ^*.

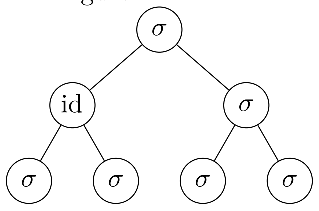
Let g be an automorphism on the regular rooted tree Σ^*. For any word $\mathbf{s} \in \Sigma^*$, there exists a unique automorphism $g_{|\mathbf{s}} : \Sigma^* \to \Sigma^*$ called a *section* of g and defined, for all word $\mathbf{t} \in \Sigma^*$, by $g(\mathbf{st}) = g(\mathbf{s})g_{|\mathbf{s}}(\mathbf{t})$, see [19] for more details. The *portrait* of g is the tree Σ^* in which each vertex $\mathbf{s} \in \Sigma^*$ is labeled by $g_{|\mathbf{s}} : \Sigma \to \Sigma$. It is denoted by $\mathfrak{p}_\infty(g)$. The permutation of Σ associated to the empty word is the *root permutation* of g. A level (*resp.* branch) of a portrait is the labeled level (*resp.* branch) of the tree.

For a given integer k, the *k-portrait* of g is the restriction of $\mathfrak{p}_\infty(g)$ to levels 0 to $k-1$ and is denoted by $\mathfrak{p}_k(g)$, it represents the action of g on the partial regular rooted tree $\Sigma^{\leq k}$.

Let $\mathbf{u} \in A^*$. The *portrait* (or *∞-portrait* — *resp.* the *k-portrait*) of $\mathbf{u}$ is the portrait (*resp.* the k-portrait) of $\rho_\mathbf{u}$: each vertex $\mathbf{s} \in \Sigma^*$ is labeled by $\rho_{\delta_\mathbf{s}(\mathbf{u})} : \Sigma \to \Sigma$. It is denoted by $\mathfrak{p}_\infty[\![\mathbf{u}]\!]$ (*resp.* $\mathfrak{p}_k[\![\mathbf{u}]\!]$). This notation is completely justified by the fact that two equivalent words have the same production function. An example is given in Figure 2.



(a) An invertible Mealy automaton,



(b) one of its portraits: $\mathfrak{p}_3[\![1]\!]$.

Figure 2 Some portrait of a two-letter Mealy automaton; id $= \mathrm{id}_\Sigma$ and σ permutes i and j.

The map from $Aut(\Sigma^*)$ to the set of portraits induces a monoid structure on the set of portraits. The neutral element of the product of portraits is the *identity portrait*: $\mathcal{I}_\infty = \mathfrak{p}_\infty(\mathrm{id}_{\Sigma^*})$. The *portraits of the automaton* $\mathcal{A}$ are the portraits of the elements of $\langle\mathcal{A}\rangle_+$. The product of two k-portraits of $\mathcal{A}$ can be expressed in terms of words: $\mathfrak{p}_k[\![\mathbf{u}]\!]\mathfrak{p}_k[\![\mathbf{v}]\!] = \mathfrak{p}_k[\![\mathbf{uv}]\!]$. It provides a monoid structure to the set of k-portraits of $\mathcal{A}$, whose neutral element is the *identity k-portrait* $\mathcal{I}_k = \mathfrak{p}_k(\mathrm{id}_{\Sigma^*})$.

A level of a portrait is *homogeneous* if all its vertices have the same label; a portrait is *homogeneous* if all its levels are homogeneous: the portrait $\mathfrak{p}_3[\![1]\!]$ of Figure 2(b) has homogeneous levels 0 and 2, but is not homogeneous. For any integer $k \geq 1$, the k-portrait $\mathfrak{p}_k(g)$ is *almost homogeneous* if $\mathfrak{p}_{k-1}(g)$ and all the $\left(\mathfrak{p}_{k-1}(g_{|i})\right)_{i\in\Sigma}$ are homogeneous.

An almost homogeneous $(k+1)$-portrait $\mathcal{K}$ is built in the following way from a homogeneous k-portrait $\mathcal{J}$ and a sequence $\tau = (\tau_i)_{i\in\Sigma}$ of permutations of Σ: the restriction of $\mathcal{K}$ to levels 0 to $k-1$ is $\mathcal{J}$ and the leaves of the subtree of the root corresponding to the letter $i \in \Sigma$ have all label τ_i. This portrait is denoted by $\mathcal{J}\lfloor\tau\rfloor$, see Figure 3.

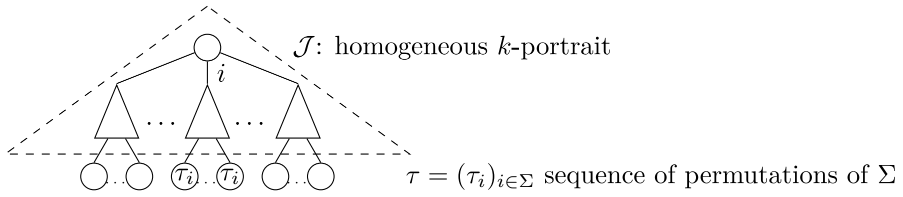

$\mathcal{J}$: homogeneous k-portrait
i
τ_i
τ_i
$\tau = (\tau_i)_{i\in\Sigma}$ sequence of permutations of Σ

Figure 3 The almost homogeneous $(k+1)$-portrait $\mathcal{J}\lfloor\tau\rfloor$, $\tau = (\tau_i)_{i\in\Sigma}$.

▶ **Remark 3.** *The product of two homogeneous k-portraits is a homogeneous k-portrait. Furthermore, if $\Sigma = \{i, j\}$:*

- *the square of a homogeneous k-portrait is the identity k-portrait $\mathcal{I}_k$;*
- *the square of an almost homogeneous k-portrait whose root permutation is the identity on Σ is the identity k-portrait;*

- *the square of an almost homogeneous k-portrait $\mathcal{J}\lfloor\tau_i, \tau_j\rfloor$ whose root permutation is the permutation of i and j is the identity k-portrait if and only if $\tau_i = \tau_j$.*

3.4 Tensor closure

When a Mealy automaton generates a finite semigroup, we may augment the alphabet on which it acts to gain a better control over its structure.

Let $\mathcal{A} = (A, \Sigma, \delta, \rho)$ be a Mealy automaton which generates a finite semigroup. Its *tensor closure* is the Mealy automaton $\mathfrak{c}(\mathcal{A}) = (A, \Xi, \bar{\delta}, \bar{\rho})$, where $\Xi = \{[\![\mathbf{s}]\!] \mid \mathbf{s} \in \Sigma^*\} = \langle \mathfrak{d}(\mathcal{A})\rangle_+$ and $\bar{\delta}$ and $\bar{\rho}$ are the natural extensions of δ and ρ:

$$\forall x \in A, \forall \mathbf{s} \in \Sigma^*,\ \bar{\delta}_{[\![\mathbf{s}]\!]}(x) = \delta_{\mathbf{s}}(x) \text{ and } \bar{\rho}_x([\![\mathbf{s}]\!]) = [\![\rho_x(\mathbf{s})]\!]\ .$$

A Mealy automaton is *tensor closed* if it is isomorphic to its tensor closure. Its dual is then minimal.

The following remark justifies the introduction of the tensor closures:

▶ **Remark 4.** *Let $\mathcal{A}$ be a two-state Mealy automaton which generates a finite semigroup. Then the automaton $\mathfrak{c}(\mathcal{A})$ generates a finite semigroup. If $\mathfrak{c}(\mathcal{A})$ is $\mathfrak{md}$-trivial, then so is $\mathcal{A}$.*

The first result is obtained by looking at the respective dual automata which generates the same semigroup. The second result is immediate since a two-state Mealy automaton $\mathcal{A}$ is $\mathfrak{md}$-trivial if and only if $\mathfrak{mdmd}(\mathcal{A})$ is trivial and the alphabet of $\mathfrak{dmd}(\mathcal{A})$ can be injected into the alphabet of $\mathfrak{c}(\mathcal{A})$.

▶ **Lemma 5.** *Let $\mathcal{A} = (A, \Xi, \delta, \rho)$ be a two-state invertible-reversible tensor closed Mealy automaton. The connected components of the powers of $\mathcal{A}$ are complete graphs.*

Proof. Let k be an integer. The connected components of $\mathcal{A}^k$ are strongly connected by reversibility. Hence any two words $\mathbf{u}$ and $\mathbf{v}$ in the same connected component are connected by a path with input label in Ξ^*. The automaton $\mathcal{A}$ being tensor closed, any word over Ξ is equivalent to a one-length word over Ξ and so the connected component of $\mathbf{u}$ and $\mathbf{v}$ is a complete graph: any two states are connected by a transition. ◀

4 The semigroup is either free or finite

Recall that a semigroup S is *free* if there exists a subset X of S such that every element of S can be written uniquely as a word over X, its *rank* is then the cardinality of X.

▶ Remark. On the other hand, a group G is free if there exists a subset X of G such that every element of G can be written uniquely as an irreducible word over $X \sqcup X^{-1}$. An invertible automaton can generate a free semigroup and a non-free group; for example, the dual of Aleshin automaton (see Figure 1(b)) generates a free semigroup, by Theorems 6 and 19, but not a free group: $ba^{-1}ba^{-1} = 1$.

▶ **Theorem 6.** *Let $\mathcal{A}$ be a reversible two-state Mealy automaton. If $\mathcal{A}$ admits a disconnected power, then it generates a finite semigroup, otherwise it generates a free semigroup of rank 2 with the states of $\mathcal{A}$ being free generators.*

Theorem 6 is a corollary of Proposition 10 and the case $p = 2$ in Proposition 14 below.

Let us look at the connected components of the powers of a Mealy automaton $\mathcal{A}$. For $m > 0$, $\mathbf{u}, \mathbf{v} \in A^m$, and $x, y \in A$, if there exists a path from $\mathbf{u}x$ to $\mathbf{v}y$ in $\mathcal{A}^{m+1}$, then there is a path from $\mathbf{u}$ to $\mathbf{v}$ in $\mathcal{A}^m$. Hence if $\mathcal{A}^n$ is disconnected, so are the $\mathcal{A}^k$, for all $k > n$. Thus there exists at most one integer n such that $\mathcal{A}^n$ is connected and $\mathcal{A}^{n+1}$ is disconnected. This integer is called the *connection degree* of $\mathcal{A}$. By convention, if $\mathcal{A}$ is disconnected, its connection degree is 0, and it has an infinite connection degree if no power of $\mathcal{A}$ is disconnected. For a Mealy automaton, having infinite connection degree coincides with the very classical notion of level transitivity (or spherical transitivity) for its dual [19, 14].

Note that the Baby Aleshin automaton (see Figure 1(c)) is reversible, has a connection degree of 2, three states, and generates an infinite non-free semigroup (its generators have order 2). So Theorem 6 and Proposition 10 do not extend to bigger stateset. However, we conjecture that Proposition 14 extends to any stateset for invertible automata.

4.1 Finite connection degree

In this section, we prove that a reversible two-state Mealy automaton has a finite connection degree if and only if it generates a finite semigroup. This result is already known [7, Lemma 3], but we present here a new proof; its main idea is to bound the sizes of the connected components of the powers of $\mathcal{A}$ once the connection degree has passed.

▶ **Lemma 7.** *Let $\mathcal{A} = (A, \Sigma, \delta, \rho)$ be a reversible Mealy automaton with at least two states, which generates a semigroup with torsion elements. Then its connection degree is finite.*

Proof. Since $\langle \mathcal{A} \rangle_+$ has torsion elements, there exist a word $\mathbf{u} \in A^+$ and two integers $n \geq 0$ and $k > 0$ such that $\mathbf{u}^n$ and $\mathbf{u}^{n+k}$ are equivalent: $\rho_{\mathbf{u}^n} = \rho_{\mathbf{u}^{n+k}}$.

Let $\mathbf{s} \in \Sigma^*$, we have: $\delta_{\mathbf{s}}(\mathbf{u}^{n+2k}) = \delta_{\mathbf{s}}(\mathbf{u}^n)\delta_{\rho_{\mathbf{u}^n}(\mathbf{s})}(\mathbf{u}^k)\delta_{\rho_{\mathbf{u}^{n+k}}(\mathbf{s})}(\mathbf{u}^k) =$
$\delta_{\mathbf{s}}(\mathbf{u}^n)\big(\delta_{\rho_{\mathbf{u}^n}(\mathbf{s})}(\mathbf{u}^k)\big)^2$. Hence all the states of the connected component of $\mathbf{u}^{n+2k}$ have form $\mathbf{v}\mathbf{w}^2$ and $\mathcal{A}^{(n+2k)|\mathbf{u}|}$ is disconnected. ◀

In the reminder of this subsection, $\mathcal{A} = (A, \Sigma, \delta, \rho)$ denotes a reversible two-state Mealy automaton ($A = \{x, y\}$) with finite connection degree n. If $z \in A$ is a state of $\mathcal{A}$, $\bar{z} \in A$ denotes the other state: $z \neq \bar{z}$.

▶ **Lemma 8.** *Let $\mathcal{C}$ be a connected component of $\mathcal{A}^m$ for some m, and let $\mathbf{u} \in A^m$ be a state of $\mathcal{C}$. The connected component (in $\mathcal{A}^{m+1}$) of $\mathbf{u}x$ has size $\#\mathcal{C}$ if it does not contain $\mathbf{u}y$, and $2\#\mathcal{C}$ if it does contain $\mathbf{u}y$.*

Proof. Let $\mathcal{D}$ be the connected component of $\mathbf{u}x$: $\mathbf{v} \in A^m$ is a state of $\mathcal{C}$ if and only if there exists $z \in A$ such that $\mathbf{v}z$ is a state of $\mathcal{D}$, hence: $N \leq \#\mathcal{D} \leq 2N$. Let $\mathbf{v}$ be a state of $\mathcal{C}$ and $z \in A$: $\mathbf{u}x$ and $\mathbf{v}z$ are in the same connected component if and only if so are $\mathbf{u}y$ and $\mathbf{v}\bar{z}$. The result follows. ◀

Recall that n is the connection degree of $\mathcal{A}$.

▶ **Lemma 9.** *For each $m \geq n$, the connected components of $\mathcal{A}^m$ have size exactly 2^n.*

Proof. By induction on $m \geq n$. For $m \in \{n, n+1\}$, the property is true (using Lemma 8 for $m = n + 1$).

Assume $m > n + 1$. Suppose that the connected components of $\mathcal{A}^{m-1}$ and $\mathcal{A}^m$ have size 2^n. Then let $\mathcal{C}$ be a connected component of $\mathcal{A}^{m+1}$ and $\mathbf{u} = u_1 \cdots u_{m+1}$ a state of $\mathcal{C}$. The word $\mathbf{u}^\bullet = u_1 \cdots u_m$ belongs to a connected component $\mathcal{D}$ of $\mathcal{A}^m$, of size 2^n by the induction hypothesis. Hence $\mathcal{C}$ has size 2^n or 2^{n+1} according to Lemma 8.

Suppose that $\mathcal{C}$ has size 2^{n+1}: it means by Lemma 8 that both $\mathbf{u}$ and $\mathbf{u}^\bullet\overline{u_{m+1}}$ belong to $\mathcal{C}$. It follows that $u_2 \cdots u_m u_{m+1}$ and $u_2 \cdots u_m \overline{u_{m+1}}$ belong to the same connected component $\mathcal{E}$ of $\mathcal{A}^m$, of size 2^n by the induction hypothesis. Hence Lemma 8 ensures the existence of a connected component of $\mathcal{A}^{m-1}$ of size 2^{n-1}, contradicting the induction hypothesis. ◀

▶ **Proposition 10.** *The connection degree of a reversible two-state Mealy automaton is finite if and only if it generates a finite semigroup.*

Proof. Let $\mathcal{A} = (A, \Sigma, \delta, \rho)$ be a reversible two-state Mealy automaton. If the connection degree of $\mathcal{A}$ is 0, $\langle \mathfrak{d}(\mathcal{A}) \rangle_+$ is the trivial semigroup and $\langle \mathcal{A} \rangle_+$ is finite [2].

Otherwise, let $n \geq 1$ be the connection degree of $\mathcal{A}$: by Lemma 9, for $m \geq n$, the connected components of $\mathcal{A}^m$ have size 2^n. These connected components are reversible Mealy automata on the alphabet Σ. Up to state numbering, there are only a finite number of such automata and thus there exist $p < q$ such that $\mathfrak{m}(\mathcal{A}^p) = \mathfrak{m}(\mathcal{A}^q)$. It follows by Remark 1 that $\langle \mathcal{A} \rangle_+$ is finite.

The reciprocal property is a particular case of Lemma 7. ◀

4.2 Infinite connection degree

Here we prove that if a reversible p-state Mealy automaton, p prime, has infinite connection degree, then it generates a free semigroup, the states of the automaton being free generators. The idea is to bound the sizes of the Nerode classes in the powers of $\mathcal{A}$.

For the next three lemmas, let $\mathcal{A} = (A, \Sigma, \delta, \rho)$ be a reversible p-state Mealy automaton, p prime, with infinite connection degree ($A = \{x_1, \ldots, x_p\}$). By Lemma 7, $\mathcal{A}$ generates an infinite semigroup.

▶ **Lemma 11.** *There cannot exist two equivalent words of different length in A^*.*

Proof. For each m, $\mathcal{A}^m$ is connected, and so any two words of length m are mapped one onto the other by an element of $\langle \mathfrak{d}(\mathcal{A}) \rangle_+$.

Let $\mathbf{u}$ and $\mathbf{v}$ be two equivalent words of different lengths, say $|\mathbf{u}| < |\mathbf{v}|$. Every word of length $|\mathbf{v}|$ is then equivalent to a word of length $|\mathbf{u}|$: if $\mathbf{w}$ is of length $|\mathbf{v}|$, then $\mathbf{w} = \delta_{\mathbf{t}}(\mathbf{v})$ for some $\mathbf{t} \in \Sigma^*$, and, by Remark 2, $\mathbf{w}$ is equivalent to $\delta_{\mathbf{t}}(\mathbf{u})$ of length $|\mathbf{u}|$. By Remark 1, the semigroup $\langle \mathcal{A} \rangle_+$ is finite, which is impossible. ◀

▶ **Lemma 12.** *All the Nerode classes of a given power A^m have the same size, which happens to be a power of p.*

Proof. Let $\mathbf{u} \in A^m$: $[\mathbf{u}] \subseteq A^m$ by definition. If $[\mathbf{u}] = A^m$, the result is clear. Otherwise, let $\mathbf{v} \in A^m - [\mathbf{u}]$. Since $\mathcal{A}^m$ is connected, $\mathbf{u}$ is mapped onto $\mathbf{v}$ by an element of $\langle \mathfrak{d}(\mathcal{A}) \rangle_+$; that is there exists $\mathbf{r} \in \Sigma^*$ such that $\mathbf{v} = \delta_{\mathbf{r}}(\mathbf{u})$.

By Remark 2, any word equivalent to $\mathbf{u}$ is mapped by $\delta_{\mathbf{r}}$ onto a word equivalent to $\mathbf{v}$. Since the automaton $\mathcal{A}^m$ is reversible, $\delta_{\mathbf{r}}$ is a permutation of A^m, hence we find $\#[\mathbf{u}] = \#[\mathbf{v}]$.

The stateset of $\mathcal{A}^m$ has size a power of p, where p is a prime number, and so has any Nerode equivalence class. ◀

▶ **Lemma 13.** *There cannot exist two equivalent words of the same length in A^*.*

Proof. Let $\mathbf{u}$ and $\mathbf{v}$ be two different equivalent words of the same length $n+1$. Let us prove by induction on $m > n$ that $\mathfrak{m}(\mathcal{A}^m)$ has at most p^n states.

The automaton $\mathcal{A}^{n+1}$ has p^{n+1} states. The words $\mathbf{u}$ and $\mathbf{v}$ are in the same Nerode class: by Lemma 12, all Nerode classes of A^{n+1} have at least p elements and $\mathfrak{m}(\mathcal{A}^{n+1})$ has at most p^n states.

Suppose that $\mathfrak{m}(\mathcal{A}^m)$ has at most p^n states. Then, since all Nerode classes have the same size by Lemma 12, the induction hypothesis implies that they have at least p^{m-n} elements. Let us look at $[x_1^m]$: it contains

$$x_1^m, \mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_{p^{m-n}-1}\ ,$$

which are pairwise distinct. Among these words, there is at least one whose suffix in x_1 is the shortest, say $\mathbf{u}_1$ without loss of generality: $p^{m-n} > 1$ and x_1^m has the longest possible suffix in x_1. Hence $[x_1^{m+1}]$ contains the following pairwise distinct $p^{m-n}+1$ words

$$x_1^{m+1}, \mathbf{u}_1 x_1, \mathbf{u}_2 x_1, \ldots, \mathbf{u}_{p^{m-n}-1} x_1, x_1 \mathbf{u}_1 \,.$$

By Lemma 12, $\#[x_1^{m+1}]$ is a power of p, so $\#[x_1^{m+1}] \geq p^{m+1-n}$. As all Nerode classes of A^{m+1} have the same cardinality, we can conclude that $\mathfrak{m}(\mathcal{A}^{m+1})$ has at most $p^{m+1}/p^{m+1-n} = p^n$ elements, ending the induction.

Consequently, since there is only a finite number of different Mealy automata with up to p^n states, there exist $k < \ell$ such that $\mathfrak{m}(\mathcal{A}^k)$ and $\mathfrak{m}(\mathcal{A}^\ell)$ are equal up to state numbering. By Remark 1, the semigroup $\langle \mathcal{A} \rangle_+$ is finite, which is impossible. ◀

As a corollary of Lemmas 7, 11 and 13 we can state the following proposition.

▶ **Proposition 14.** *Let $\mathcal{A}$ be a reversible p-state Mealy automaton, p prime. If the automaton $\mathcal{A}$ has infinite connection degree, then it generates a free semigroup of rank p with the states of $\mathcal{A}$ being free generators of the semigroup. The converse holds for $p = 2$.*

5 Decidability of finiteness and of freeness

This section is devoted to the decidability of finiteness and of freeness for semigroups generated by two-state invertible-reversible Mealy automata by linking Theorem 6 and the possible $\mathfrak{m}\mathfrak{d}$-triviality of such an automaton.

▶ **Lemma 15.** *Let $\mathcal{A} = (A, \Sigma, \delta, \rho)$ be a two-state invertible-reversible automaton of finite connection degree n. Two elements of Σ^* which have the same action on a word of A^n are equivalent.*

Proof. It is sufficient to prove that id_{A^*} is the only element of $\langle \mathfrak{d}(\mathcal{A}) \rangle_+$ which fixes a word of A^n.

If $n = 0$, $\langle \mathfrak{d}(\mathcal{A}) \rangle_+$ is the trivial semigroup and the result is true. Otherwise, let $\mathbf{u} \in A^n$ and $\mathbf{s} \in \Sigma^*$ such that $\mathbf{u}$ is stable by $\delta_{\mathbf{s}}$: $\delta_{\mathbf{s}}(\mathbf{u}) = \mathbf{u}$.

By Lemma 8, $\mathcal{A}^{n+1}$ has two connected components: $\mathbf{u}x$ belongs to one of them and $\mathbf{u}y$ to the other one. Looking forward, a connected component $\mathcal{C}$ of $\mathcal{A}^m$, for $m \geq n$, originates two connected components of $\mathcal{A}^{m+1}$: $\{\mathbf{v}z_{\mathbf{v}} \mid \mathbf{v} \in \mathcal{C}, z_{\mathbf{v}} \in A\}$ and $\{\mathbf{v}\overline{z_{\mathbf{v}}} \mid \mathbf{v} \in \mathcal{C}\}$. And all connected component of $\mathcal{A}^{m+1}$ are built this way. Hence if two different words of the same length $m > n$ have the same prefix of length n, they belong to different connected components of $\mathcal{A}^m$.

Let $\mathbf{t} \in \Sigma^*$ satisfy $\rho_{\mathbf{u}}(\mathbf{s}) = \mathbf{t}$, and let $\mathbf{v}, \mathbf{w} \in A^*$ such that $\mathbf{t}$ maps $\mathbf{v}$ onto $\mathbf{w}$: $\delta_{\mathbf{t}}(\mathbf{v}) = \mathbf{w}$. The words $\mathbf{uv}$ and $\mathbf{uw}$ belong to the same connected component:

$$\delta_{\mathbf{s}}(\mathbf{uv}) = \delta_{\mathbf{s}}(\mathbf{u})\delta_{\rho_{\mathbf{u}}(\mathbf{s})}(\mathbf{v}) = \mathbf{u}\delta_{\mathbf{t}}(\mathbf{v}) = \mathbf{uw} \,,$$

and have a common prefix of length n, so they are equal. Hence: $\delta_{\mathbf{t}} = \mathrm{id}_{A^*}$. As $\mathfrak{d}(\mathcal{A})$ is reversible, $\mathbf{t}$ is mapped onto $\mathbf{s}$ by an element of $\langle \mathcal{A} \rangle_+$ and $\delta_{\mathbf{s}} = \mathrm{id}_{A^*}$. ◀

We have a similar (but weaker) result on shorter words for tensor closed Mealy automata. **In the next three lemmas of this section, $\mathcal{A} = (A, \Xi, \delta, \rho)$ denotes a tensor closed two-state invertible-reversible automaton of finite connection degree n: $A = \{x, y\}$.** By Lemma 5, $\mathcal{A}^n$ is complete as a graph. Furthermore, a transition has a unique label: if a transition had several labels, they would coincide on a word of A^n and by Lemma 15 they actually would be the same letter of Ξ.

▶ **Lemma 16.** *Let k be an integer, $1 \leq k \leq n$. Two elements of Ξ^* which map a given word of A^k into the same word have the same action on A^k.*

Proof. Each word of Ξ^* is equivalent to a letter of Ξ, hence it is sufficient to prove the result for letters.

The Mealy automaton $\mathcal{A}^n$ has 2^n states, is complete as a graph and each transition has a unique label, so $\#\Xi = 2^n$. By hypothesis, Ξ is the set of elements of $\langle \mathfrak{d}(\mathcal{A})\rangle_+$, so $\#\langle \mathfrak{d}(\mathcal{A})\rangle_+ = 2^n$.

Let us consider the minimization of $\mathfrak{d}(\mathcal{A})$, using the sequence of increasingly finer equivalences $(\equiv_k)$ introduced in Section 3.1. Each n-class of Ξ is a singleton by Lemma 15, hence the sequence $(\equiv_k)$ remains constant at least from n on. So the Nerode equivalence produces 2^n equivalence classes formed uniquely by singletons, by partitioning the stateset of $\mathfrak{d}(\mathcal{A})$ of cardinality 2^n in n steps, each step cutting each class of the previous one into at most two subsets as $\#A = 2$. Hence the equivalence $\equiv_k$ cuts each $(k-1)$-class into two sets of the same cardinality: $\forall k, 0 \leq k \leq n, \forall s \in \Xi$, $\#[s]_k = \#[s]_{k-1}/2 = 2^{n-k}$.

Let k, $1 \leq k \leq n$, $\mathbf{u} \in A^k$, and $s \in \Xi$. We have:

$$[s]_k \subseteq \{t \in \Xi \mid t(\mathbf{u}) = s(\mathbf{u})\}\ . \tag{1}$$

The left set in Equation (1) has cardinality 2^{n-k}, it is the set of elements of Ξ which coincide with s on A^k. Since two elements of Ξ whose actions coincide on a word of A^n are equivalent, the right set of Equation (1) has cardinality at most $\#A^{n-k} = 2^{n-k}$, and so the two sets of Equation (1) are equal, leading to the result. ◀

One consequence of Lemma 16 is that an element of Ξ^* which fixes a word of length k on A fixes completely A^k.

Denote by id the identity of A and by σ the permutation of x and y. We can translate Lemma 16 in terms of portraits of $\mathfrak{d}(\mathcal{A})$: whenever two k-portraits of $\mathfrak{d}(\mathcal{A})$ have an identical branch, they are equal. In particular, $\mathcal{I}_k$ being a portrait of $\mathfrak{d}(\mathcal{A})$, if a whole branch of a k-portrait of $\mathfrak{d}(\mathcal{A})$ is labeled by id, this portrait is $\mathcal{I}_k$. Hence if in a k-portrait of $\mathfrak{d}(\mathcal{A})$, all vertices at level less than $k-1$ are labeled by id, this portrait is either $\mathcal{I}_k$ or $\mathcal{I}_{k-1}\lfloor\sigma,\sigma\rfloor$. Note that for $k \leq n$, both $\mathcal{I}_k$ and $\mathcal{I}_{k-1}\lfloor\sigma,\sigma\rfloor$ are portraits of $\mathfrak{d}(\mathcal{A})$.

By Lemma 15, any element of $\langle \mathfrak{d}(\mathcal{A})\rangle_+$ whose n-portrait is $\mathcal{I}_n$ acts trivially on A^*.

What are the possible portraits of $\mathfrak{d}(\mathcal{A})$? Since $\mathcal{A}^n$ is connected and $\mathcal{A}$ is tensor closed, it is immediate that each finite sequence $(\pi_i)_{1\leq i\leq n} \in \{\mathrm{id},\sigma\}^n$ labels a branch of an n-portrait of $\mathfrak{d}(\mathcal{A})$: in $\mathcal{A}^n$, there is a transition with input $s \in \Xi$ from x^n to $\pi_1(x)\cdots\pi_n(x)$ and the leftmost branch of $\mathfrak{p}_n[\![s]\!]$ is labeled by π.

▶ **Lemma 17.** *The portraits of $\mathfrak{d}(\mathcal{A})$ are homogeneous.*

Proof. Let us prove the result for $k \leq n$, by induction on $k \geq 1$. A 1-portrait has a unique element, its root, and so is homogeneous.

Suppose that the ℓ-portraits of $\mathfrak{d}(\mathcal{A})$ are all homogeneous, for $\ell \leq k < n$. Let us consider a letter $s \in \Xi$ and $\mathcal{S} = \mathfrak{p}_{k+1}[\![s]\!]$: it is almost homogeneous by the induction hypothesis. More precisely: $\mathcal{S} = \mathfrak{p}_k[\![s]\!]\lfloor\tau_1,\tau_2\rfloor$ for τ_1,τ_2, some permutations of A.

First case: δ_s permutes x and y. We consider the following $(n+1)$-portrait $\mathcal{K}$:

- the restriction of $\mathcal{K}$ to levels 0 to $(n-k-1)$ is $\mathcal{I}_{n-k}$,
- in bottom-left of $\mathcal{I}_{n-k}$, we put $\mathfrak{p}_{k+1}[\![s]\!]$: the root of $\mathfrak{p}_{k+1}[\![s]\!]$ is the left child of the bottom-left leaf of $\mathcal{I}_{n-k}$ (it is possible since we can choose the left branch of a portrait, applying Lemma 16 and $\mathfrak{p}_{k+1}[\![s]\!]$ is actually a portrait of $\mathfrak{d}(\mathcal{A})$),

- it is completed to be a portrait of $\mathfrak{d}(\mathcal{A})$.

The leftmost branch of $\mathcal{K}^2$ starts with id^n. Hence by Lemma 15, $\mathcal{K}^2$ is the identity $(n+1)$-portrait, which implies $\tau_1 = \tau_2$ by Remark 3 and Lemma 15, that is $\mathcal{S}$ is homogeneous.

Second case: δ_s stabilizes A. Let $\mathcal{L}$ be the $(k+1)$-portrait whose root permutation is σ and all other vertices are labeled by id: it is a portrait of $\mathfrak{d}(\mathcal{A})$ since so are all homogeneous $(k+1)$-portraits with root permutation σ from first case. Then by multiplying $\mathcal{S}$ by $\mathcal{L}$, we obtain a non-homogeneous $(k+1)$-portrait with root permutation σ which has to be a portrait of $\mathfrak{d}(\mathcal{A})$. That is impossible.

The proof is similar for $k > n$, considering the portrait $\mathfrak{p}_k[\![s]\!]$. ◀

▶ **Lemma 18.** *The states of $\mathcal{A}$ are equivalent.*

Proof. By Lemma 17, all the portraits of $\mathfrak{d}(\mathcal{A})$ are homogeneous. For any letter $s \in \Xi$, since its portrait is homogeneous, $\rho_x(s)$ and $\rho_y(s)$ are equivalent. The automaton being tensor closed, they are equal, and so $\rho_x = \rho_y$. ◀

▶ **Theorem 19.** *Let $\mathcal{A}$ be a two-state invertible-reversible Mealy automaton. It generates a finite group if and only if it is $\mathfrak{md}$-trivial.*

Proof. By [2], if $\mathcal{A}$ is $\mathfrak{md}$-trivial, it generates a finite group.

Suppose that $\mathcal{A}$ generates a finite group and consider its tensor closure $\mathfrak{c}(\mathcal{A})$: $\mathfrak{c}(\mathcal{A})$ generates a finite group by Remark 4. The connection degree of $\mathfrak{c}(\mathcal{A})$ is finite by Proposition 10 and so $\mathfrak{c}(\mathcal{A})$ is $\mathfrak{md}$-trivial by Lemma 18. Hence $\mathcal{A}$ is $\mathfrak{md}$-trivial by Remark 4. ◀

The last theorem summarizes all the decidability results arising from this article.

▶ **Theorem 20.** *It is decidable whether a two-state invertible-reversible Mealy automaton with alphabet Σ generates a finite group, in time $\mathcal{O}(\Sigma \log \Sigma)$. It is decidable whether it generates a free semigroup, in time $\mathcal{O}(\Sigma \log \Sigma)$.*

It is decidable whether a two-letter invertible-reversible Mealy automaton with stateset A generates a finite group, in time $\mathcal{O}(A \log A)$.

Up to now, the only methods to conclude infiniteness of automaton groups were to prove the existence of an element of infinite order [18, FindElementOfInfiniteOrder][5, SIZE_FR], using Sidki's fundamental work [8, 21], or to test level transitivity [5, IsLevelTransitive]. All these methods give sufficient but not necessary conditions.

To illustrate the actual efficiency of the $\mathfrak{md}$-triviality as an algorithm to test finiteness, let us consider the 2-letter 6-state invertible-reversible Mealy automata. Bireversible Mealy automata are particular invertible-reversible Mealy automata and an invertible-reversible automaton generates a finite group only if it is bireversible [2]. Testing the $\mathfrak{md}$-triviality of the 3446 bireversible 2-letter 6-states Mealy automata takes 751ms[1], while applying FindElementOfInfiniteOrder, SIZE_FR or IsLevelTransitive to determine the infinity of the group generated by the particular bireversible 2-letter 6-state Mealy automaton of Figure 2(a) is unsuccessful after three weeks of computation.

Acknowledgments I would like to thank Jean Mairesse and Matthieu Picantin for numerous discussions around this topic.

[1] Timings obtained on an Intel Xeon computer with clock speed 2.13GHz; programs written in GAP [10].

References

1 A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

2 A. Akhavi, I. Klimann, S. Lombardy, J. Mairesse, and M. Picantin. On the finiteness problem for automaton (semi)groups. *Int. J. Algebra Comput.*, 22(6):26p., 2012.

3 S.V. Alešin. Finite automata and the Burnside problem for periodic groups. *Mat. Zametki*, 11:319–328, 1972.

4 A.S. Antonenko. On transition functions of Mealy automata of finite growth. *Matematychni Studii.*, 29(1):3–17, 2008.

5 L. Bartholdi. FR *Functionally recursive groups – a GAP package, v.1.2.4.2*, 2011.

6 L. Bartholdi, I.I. Reznykov, and V.I. Sushchanskiĭ. The smallest Mealy automaton of intermediate growth. *J. Algebra*, 295(2):387–414, 2006.

7 I. Bondarenko, R.I. Grigorchuk, R. Kravchenko, Y. Muntyan, V. Nekrashevych, D. Savchuk, and Z. Šunić. On classification of groups generated by 3-state automata over a 2-letter alphabet. *Algebra Discrete Math.*, 1:1–163, 2008.

8 I.V. Bondarenko, N.V. Bondarenko, S.N. Sidki, and F.R. Zapata. On the conjugacy problem for finite-state automorphisms of regular rooted trees. *Groups, Geometry, and Dynamics*, in press. arXiv:math.GR/1011.2227.

9 A.J. Cain. Automaton semigroups. *Theor. Comput. Sci.*, 410:5022–5038, 2009.

10 The GAP Group. *GAP – Groups, Algorithms, and Programming, v.4.4.12*, 2008.

11 Y. Glasner and Sh. Mozes. Automata and square complexes. *Geom. Dedicata*, 111(1):43–6, 2005.

12 R. Grigorchuk and A. Żuk. The lamplighter group as a group generated by a 2-state automaton, and its spectrum. *Geom. Dedicata*, 87:209–244, 2001.

13 R.I. Grigorchuk. On Burnside's problem on periodic groups. *Funktsional. Anal. i Prilozhen.*, 14(1):53–54, 1980.

14 R.I. Grigorchuk, V.V. Nekrashevich, and V.I. Sushchanskiĭ. Automata, dynamical systems, and groups. *Tr. Mat. Inst. Steklova*, 231:134–214, 2000.

15 I. Klimann, J. Mairesse, and M. Picantin. Implementing computations in automaton (semi)groups. In *Proc. 17th CIAA*, volume 7381 of *LNCS*, pages 240–252, 2012. DOI No: 10.1142/S021819671250052X.

16 V. Maltcev. Cayley automaton semigroups. *Int. J. Algebra Comput.*, 19(1):79–95, 2009.

17 A. Mintz. On the Cayley semigroup of a finite aperiodic semigroup. *Int. J. Algebra Comput.*, 19(6):723–746, 2009.

18 Y. Muntyan and D. Savchuk. automgrp *Automata Groups – a GAP package, v.1.1.4.1*, 2008.

19 V. Nekrashevych. *Self-similar groups*, volume 117 of *Mathematical Surveys and Monographs.* American Mathematical Society, Providence, RI, 2005.

20 D.M. Savchuk and Y. Vorobets. Automata generating free products of groups of order 2. *J. Algebra*, 336(1):53–66, 2011.

21 S.N. Sidki. Automorphisms of one-rooted trees: growth, circuit structure, and acyclicity. *J. Math. Sci. (New York)*, 100(1):1925–1943, 2000. Algebra, 12.

22 P.V. Silva and B. Steinberg. On a class of automata groups generalizing lamplighter groups. *Int. J. Algebra Comput.*, 15(5-6):1213–1234, 2005.

23 B. Steinberg, M. Vorobets, and Y. Vorobets. Automata over a binary alphabet generating free groups of even rank. *Int. J. Algebra Comput.*, 21(1-2):329–354, 2011.

24 M. Vorobets and Y. Vorobets. On a free group of transformations defined by an automaton. *Geom. Dedicata*, 124:237–249, 2007.

25 M. Vorobets and Y. Vorobets. On a series of finite automata defining free transformation groups. *Groups Geometry and Dynamics*, 4:377–405, 2010.

Mortality of Iterated Piecewise Affine Functions over the Integers: Decidability and Complexity

Amir M. Ben-Amram

The Academic College of Tel-Aviv Yaffo
`amirben@mta.ac.il`

Abstract

In the theory of discrete-time dynamical systems, one studies the limiting behaviour of processes defined by iterating a fixed function f over a given space. A much-studied case involves piecewise affine functions on $\mathbb{R}^n$. Blondel et al. (2001) studied the decidability of questions such as *mortality* for such functions with rational coefficients. *Mortality* means that every trajectory includes a 0; if the iteration is seen as a loop `while` $(x \neq 0)$ x `:=` $f(x)$, mortality means that the loop is guaranteed to terminate.

Blondel et al. proved that the problems are undecidable when the dimension n of the state space is at least two. They assume that the variables range over the rationals; this is an essential assumption. From a program analysis (and discrete Computability) viewpoint, it would be more interesting to consider integer-valued variables.

This paper establishes (un)decidability results for the *integer* setting. We show that also over integers, undecidability (moreover, Π^0_2 completeness) begins at two dimensions. We further investigate the effect of several restrictions on the iterated functions. Specifically, we consider bounding the size of the partition defining f, and restricting the coefficients of the linear components. In the decidable cases, we give complexity results. The complexity is PTIME for affine functions, but for piecewise-affine ones it is PSPACE-complete. The undecidability proofs use some variants of the Collatz problem, which may be of independent interest.

1998 ACM Subject Classification F.1.0 Computation by abstract devices – General

Keywords and phrases discrete-time dynamical systems, termination, Collatz problem

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.514

1 Introduction

The purpose of this paper is to study some computational problems regarding the asymptotic behaviour of discrete-time dynamical systems, in particular problems related to questions about the termination of simple programs. In the context of this paper, an (n-dimensional) *discrete-time dynamical system* is defined by $x_{t+1} = f(x_t)$, where $f : \mathbb{R}^n \to \mathbb{R}^n$. For a given initial point x_0, the sequence so generated is called *a trajectory.* A class of functions much studied in the Dynamical System literature is piecewise affine functions (Definition 3 at the end of this section). Some of the central problems regarding such systems involve their asymptotic behavior, among which are *global convergence* and *mortality*, defined next.

▶ **Definition 1.** Let f be an arbitrary map on $\mathbb{R}^n$; let 0 be the origin. We call f *globally convergent to zero* if for every initial point x_0, the trajectory $x_{t+1} = f(x_t)$ converges to 0 (the words "to zero" and sometimes "globally" may be omitted in the sequel). We call f *mortal* if for every initial point, the trajectory reaches the origin.

These problems were studied from a Computability viewpoint by Blondel et al. [3]. They considered piecewise affine functions f, where the coefficients are all rational (this is important since we are considering computability in the traditional, discrete sense).

▶ **Theorem 2.** [3] . *The following problems are undecidable for all $n \geq 2$: Given a piecewise affine function $f : \mathbb{R}^n \to \mathbb{R}^n$ (with rational coefficients), is it globally convergent? Is it mortal?*

Global convergence is decidable for $n = 1$ when the function is continuous.

Among the many decision problems studied for dynamical systems, mortality is the most appealing in its discrete setting since it is a restricted halting problem for a very simple type of program (more on the connection to program termination problems below). The global convergence problem as defined above is very closely related (in a discrete setting, a sequence x_t converges to 0 if and only if it is eventually constantly zero) and we will treat both problems throughout the following sections.

The main contribution of this paper is to establish for the integer setting results similar to Theorem 2. The proofs in [3] do not apply to this setting, since they are based on encoding the state of a computation (say, a Turing-machine tape) in the fractional digits of a number; unlimited precision is essential. To handle a discrete (or limited precision) setting, different proof techniques are necessary. As in the continuous case, we have decidability for one dimension and undecidability for two or more. The new undecidability result is stronger than the one in [3] in that it is achieved for a class of functions where there is a fixed bound on the number of regions in the partition defining f ([3] left this case open; but one can strengthen their result in this way, too). Furthermore, we prove Π^0_2 completeness, which is a sharper result than mere undecidability.

Next, we consider some other restrictions on the space of functions. Section 4 shows undecidability for functions of a very simple form, $f(x) = (x_{i_1} + b_1, x_{i_2} + b_2)$.

We should point out that when there is only one region (that is, f is affine) all problems considered in this paper are decidable in any dimension, and moreover in polynomial time; this follows from Linear-Algebraic techniques as used for linear discrete dynamical systems over the reals. However, even when we have one convex region in which the function is defined as an affine function, and is zero elsewhere, its analysis becomes challenging and, in fact, is still an open problem [5].

Section 5) considers the one-dimensional case, where the problems are decidable, and we concentrate on complexity results. While the affine case is PTIME, we show the one-dimensional piecewise-affine case to be PSPACE-complete.

In Section 6 we mention a couple of similar problems whose solutions follow easily, whereas the conclusion (Sect. 7) presents a selection of open problems.

To conclude this introduction, here are a few comments on the background to this research. The connection of Dynamical System Theory to the Theory of Computation is obvious—any traditional model of computation is a discrete-time dynamical system. However, most literature in Dynamical System Theory refers to a continuous state-space, whereas in the Theory of Computation, most models are discrete (but analog models *are* studied and cross-fertilization with Dynamical System Theory is evident; see for example [19]).

Taking classical (discrete) computability to continuous dynamics, Moore [17] discusses the significance of undecidability (in the Turing sense) to dynamical systems. He shows that a TM can be simulated by a piecewise-affine map on the plane—the method is quite similar to the one used by Blondel et al. He concludes that for such a map, the set of points on which the sequence converges (to a particular zone) is not recursive. Koiran, Cosnard, and Garzon showed that such simulations can be done already in two dimensions, but not in one [13]. Blondel and Tsiklitis [4] survey applications of discrete computability and complexity to dynamical systems, including the above-cited results.

The author's interest in the mortality problem and its variants is due to their interpretation as special cases of the halting problem, a.k.a. program termination (the latter term often refers to termination for any input, that is, a global property as those studied here). Decision procedures for the termination of *simple loops*, where a fixed (loop-free) computation is iterated until an end-condition is met, have gained much interest in program analysis and several heuristic approaches have been proposed (e.g., linear ranking functions [8, 18, 1]). These works concentrate on integer data, and on functions which are linear, piecewise linear, or defined by linear constraints (which is a wider class). In [20], Tiwari draws on inspiration from Dynamical System Theory to solve a termination problem for loops with an affine-linear update function—however, over the reals. Consequently, Braverman [5] tackled the problem for the rationals and integers. Passing from the real-number world to the integers is sometimes quite a challenge as the theory of integers has many surprises of its own. A notorious example is the solution of multivariate polynomial equations—or, more generally, quantifier elimination—decidable for the reals, but undecidable for integers. Another classic example of an integer-specific problem is the Collatz problem (or "$3x+1$ problem") [15]. Lagarias' excellent volume shows clearly that this problem is related both to dynamical system theory and to computability theory. Regarding the latter, Conway [9, 10] and subsequent works [12, 7, 14] proved undecidability results for certain *generalized Collatz problems* by showing how to simulate a counter machine. We shall make essential use of this idea, specifically of the approach of [14], who were first to show Π^0_2 hardness of mortality for a generalized Collatz function.

For completeness, we should mention that there are kinds of dynamical systems very remote from our subject, the interested reader is referred to [6].

Preliminary definitions

A closed (respectively open) *half-space* of $\mathbb{R}^n$ is the set defined by $\{x \in \mathbb{R}^n : cx + d \geq 0\}$ (respectively > 0) where $c \in \mathbb{R}^n, d \in \mathbb{R}$. We are interested in *rational* half-spaces, where the components of c, d are rational. A (rational) *convex polyhedron* is the intersection of a finite number of (closed or open) half-spaces, which we sometimes call *the constraints*.

► **Definition 3.** A *piecewise affine function* on $\mathbb{R}^n$ (respectively $\mathbb{Z}^n$) is a function f where

$$f(x) = A_i x + b_i \quad \text{for } x \in H_i \text{ (respectively, } H_i \cap \mathbb{Z}^n) \tag{1}$$

where the sets $H_1, \dots, H_p$ are an exhaustive partition of $\mathbb{R}^n$ into p convex rational polyhedra, and for $i = 1, \dots, p$, $A_i \in \mathbb{Q}^{n\times n}$ and $b_i \in \mathbb{Q}^n$ (respectively, $\mathbb{Z}^{n\times n}$ and $\mathbb{Z}^n$).

The restriction to polyhedra which are convex is somewhat arbitrary, but follows the definition in [3] and other related publications. Section 3 discusses an implication of this restriction.

We use the notation $[a, b]$ for an interval of integers, namely $\{a, a+1, \dots, b\}$.

The counter machine model, due to Minsky [16], is well known. The details of the definition vary in the literature, but the differences are not essential. At this point, it should suffice to remark that we write a state of the machine as $(i, \langle r_1, \dots, r_n\rangle)$ where i is the internal state and r_j the contents of register (counter) R_j.

Π^0_2 is the class of decision problems that can be expressed by a formula of the form $(\forall z)(\exists y)P(x, y, z)$ with P recursive. This class properly contains RE (characterized by formulas $(\exists y)P(x, y)$). A standard Π^0_2-complete set is the totality problem (termination on all inputs) for Turing machines, as well as counter machines.

2 Undecidability in Two Dimensions

Blondel et al. prove their undecidability results for Generalized Collatz Problems by reducing from the *mortality problem for counter machines* (the set of CMs that halt on every given configuration). Indeed, the PAF mortality problem is similar to the GCP, which is also a problem of reachability (of 1 instead of 0), but the functions considered in the GCP are not piecewise affine; their expression involves division and remainders, which make it easier to encode computations and simulate counter machines. Since the reductions in [3] are based on fractional numbers, we introduce new reductions. In addition, we rely on some proof techniques from [14], who proved that mortality of counter machines is Π^0_2-complete (leading to a similar result for GCPs). Note that hardness of mortality does not follow easily from hardness of totality, since many programs, while halting from all initial states, still diverge if started at a configuration that is not reachable in a proper computation (i.e., from an initial state).

▶ **Definition 4.** A function $g : \mathbb{N}_+ \to \mathbb{N}_+$ is called a *generalized Collatz function* if there is an integer $m > 0$, positive integers $\{a_0, \dots, a_{m-1}\}$ and non-negative integers $\{b_0, \dots, b_{m-1}\}$, such that whenever $x \equiv i \bmod m$, $g(x) = a_i(x - i)/m + b_i$.

A *standard representation* of g is the list $m, a_0, b_0, \dots, a_{m-1}, b_{m-1}$.

The standard Collatz function is usually described by $g(x) = 3x + 1$ if x is odd, $g(x) = x/2$ if x is even. In our notation, it is given by $m = 2$, $a_0 = 1, b_0 = 0$, $a_1 = 6, b_1 = 4$.

▶ **Definition 5.** GCP (for Generalized Collatz Problem) is the problem of deciding, from a standard representation of g, whether every trajectory of g reaches 1.

▶ **Theorem 6.** [14] *GCP is Π^0_2-complete.*

Our first result is

▶ **Theorem 7.** *Global convergence and mortality over $\mathbb{Z}$ of piecewise affine functions with integer coefficients is a Π^0_2-complete problem.*

Proof. Like the GCP, our problem is clearly a "$\forall\exists$" problem, hence belonging to Π^0_2. For Π^0_2-hardness, we reduce from the GCP. Given a description $\langle m, a_0, b_0, \dots, a_{m-1}, b_{m-1}\rangle$ of a generalized Collatz function g, our reduction produces the function f defined by the $m + 1$ cases Table 1, where for convenience every region has a label.

region label	constraints	$f(x, y)$
D	$x{>}m,\ y \geq 0$	$(x - m, y + 1)$
R_0	$x{=}0\ ,\ y > 0$	$(a_0y + b_0, 0)$
R_1	$x{=}1\ ,\ y > 0$	$(a_1y + b_1, 0)$
R_2	$x{=}2\ ,\ y \geq 0$	$(a_2y + b_2, 0)$
$\dots$		
R_{m-1}	$x{=}m,\ y \geq 0$	$(a_my + b_m, 0)$
Z	elsewhere	$(0, 0)$

Table 1 PAF representing a GCP.

To simulate a Collatz sequence generated by g, we repeatedly apply f starting from $(x_0, 0)$. Observe that started at $(x, 0)$ for $x > 1$, the computation will stay in Region D (the *division* region) until obtaining the result $(i, (x-i)/m)$ where $x \equiv i \bmod m$. Computation will then reach one of Regions R_0 through R_{m-1} and apply the appropriate case of g, producing $(g(x), 0)$. This process iterates until arriving at $(1, 0)$, which indicates the convergence of the Collatz sequence and is mapped to $(0, 0)$, the stopping state in our problems.

Note that every point in the regions D through R_{m-1} represents in fact an intermediate state of a valid simulation of a g sequence, while all other points map immediately to the origin. Thus, f globally converges to zero if and only if it is mortal if and only if g satisfies the GCP. ◀

In the following sections we prove results which are strictly stronger than Theorem 7, and their proofs are also more involved, and could not be fully given in this extended abstract; hopefully, the above proof was sufficiently clear, and the proof fragments in the sequel will give the reader an idea of the techniques that were necessary to obtain the other results.

3 Undecidability with a Bounded Number of Regions

The number of regions in the definition of function f above depends on the modulus m of the Collatz function. In this section, we establish that the mortality problem is also hard when the number of regions is bounded by some *a priori* constant (if it is large enough). This will follow immediately from proving that a result like Theorem 6 holds for a fixed modulus (which may be interesting in itself)[1].

First, we introduce a special variant of the counter machine. An *enhanced CM* is a counter machine, where an instruction may increase the value of a register by an arbitrary positive constant (a decrease, however, is limited to 1 as usual).

▶ **Theorem 8.** *From an (ordinary) counter machine M, one can compute an enhanced counter machine U_M such that U_M is mortal if and only if M halts on every initial state $(1, \langle x, 0, \ldots, 0\rangle)$. The number of registers and instructions of U_M is independent of M.*

The proof combines the reduction of totality to mortality by [3] with the use of a universal CM. The enhanced instruction set is necessary to keep the size of U_M independent of M.

▶ **Theorem 9.** *There is a constant m such that GCP restricted to modulus m is Π^0_2-complete.*

Proof. We reduce from the standard problem: Given an (ordinary) counter machine M, does it halt on every initial state $(1, \langle x, 0, \ldots, 0\rangle)$? The reduction first maps M to U_M, then translates this counter machine to a generalized Collatz function g. The modulus of g only depends on the number of registers and instructions in U_M, and is independent of M. ◀

▶ **Corollary 10.** *There is a constant m such that global convergence and mortality over $\mathbb{Z}$ of piecewise affine functions f with integer coefficients and m regions are Π^0_2-complete.*

It is natural to ask what the threshold of undecidability is regarding the number of regions. Consider the function f defined in Section 2. How many regions does it have? There are $m+2$ rows in the table. But the last one does not count as a single region according to Definition 3. The problem is that it is not convex, and therefore has to be split into convex polyhedra. It seems natural to adjust the way we count regions by allowing the function to be defined explicitly on certain convex polyhedral regions, and *zero elsewhere.* We are interested in the number $\mathcal{R}$ of these convex regions. The case $\mathcal{R} = 1$ coincides with an important open problem, see [5]. For $\mathcal{R} = 2$ the problem has been recently shown to be undecidable [2].

[1] By constructing a universal GCP, both [12, 7] show that undecidability of the *reachability problem* (is 1 reachable from a given initial point?) holds for a fixed modulus. But this does not imply any conclusion for mortality; universal machines are, of course, immortal.

4 Undecidability for Monic Functions

The result of the last section may be interpreted as showing that the hardness of the problem does not depend on allowing an unbounded number of regions. So, it must come from allowing an unbounded set of possible affine functions for each region. In this section we will show that even when restricting the coefficients in the linear components of these functions to 1, we still have undecidability.

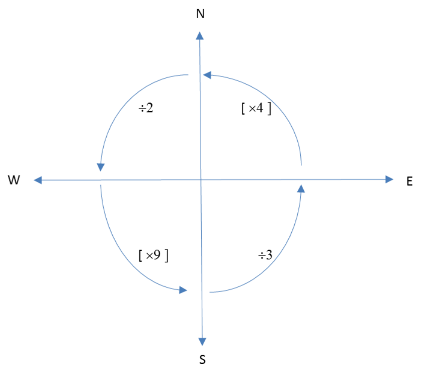


Figure 1 The dynamics of a Compass Collatz-like function.

▶ **Definition 11.** A *monic* piecewise affine function (abbreviated to "monic PAF") on $\mathbb{Z}^2$ is a piecewise affine function where each of the defining affine components has the form $f(x) = (x_{i_1} + b_1, x_{i_2} + b_2)$, where $\{i_1, i_2\} = \{1, 2\}$. In addition, the halfspaces which define the space partition are given by inequalities of the form $cx_i + d \geq 0$ such that $c = \pm 1$.

The main point is that the definition of the function f does not allow for terms ax_j with $a > 1$ (or sums like $x_1 + x_2$).

In the next proof we will use 2-counter machines (2CM). By combining the technique of Blondel et al. with that of Simon and Kurtz, it is not hard to show that 2CM mortality is Π^0_2-complete. It will be useful to restrict the machines under consideration to a certain class of "normal forms". A normalized machine: (1) modifies (increments or decrements) every register in every transition; (2) only halts at a state where the values of the counters are either $\langle 0, 0\rangle$, $\langle 0, 1\rangle$ or $\langle 1, 0\rangle$.

From these machines, we construct a special (and new) variant of the Collatz problem, the *Compass Collatz-like function*. In describing such functions we make use of the set $\mathbf{C} = \{E, N, W, S\}$ of Compass Directions. A pair (x, Δ), where $x \in \mathbb{N}$, may be depicted as a point on one of the axes in the Cartesian plane, in the direction Δ (we may also refer to such a point as lying *on the* Δ *axis*). We refer to it as *a compass point.*

▶ **Definition 12.** A function $g : \mathbb{N}_+ \times \mathbf{C} \to \mathbb{N}_+ \times \mathbf{C}$ is called a *Compass Collatz-like function* if there is a number $m = 6p$ with $p \geq 5$ a prime, sets $R_N, R_S \subseteq [0, m-1]$ and integers $w_i \in [0, m-1]$ for $i = 0, \ldots, m-1$, such that g satisfies the following equations (for convenience we represent its argument in the form $mx + rp + i$, where $x \geq 0$, $0 \leq r < 6$ and $0 \leq i < p$):

$$\begin{aligned}
g(mx+rp+i,E) &= \begin{cases}(mx+rp+i,N) & rp+i \in R_N\\ (4(mx+rp)+i,N) & rp+i \notin R_N\end{cases}\\
g(mx+rp+i,N) &= (\frac{1}{2}mx+\lfloor\frac{1}{2}r\rfloor p+i,W)\\
g(mx+rp+i,W) &= \begin{cases}(mx+rp+w_{rp+i},S) & rp+i \in R_S\\ (9(mx+rp)+w_{rp+i},S) & rp+i \notin R_S\end{cases}\\
g(mx+rp+i,S) &= (\frac{1}{3}mx+\lfloor\frac{1}{3}r\rfloor p+i,E)
\end{aligned} \tag{2}$$

The action of such a function is schematically represented in Figure 1. The "compass" representation is useful for the transition to a 2-dimensional dynamical system (and also makes it easier to visualize the dynamics).

▶ **Definition 13.** CCP (for Compass Collatz-like Problem) is the problem of deciding, from a standard representation of g, whether it is the case that every such sequence reaches some point (x, E) with $x < m$ (the set of such points is called *the final zone*).

▶ **Lemma 14.** *CCP is* Π^0_2*-complete.*

Proof. (Sketch) The proof is a reduction from 2CM mortality. Given a normal 2-counter machine M we simulate it by a Compass Collatz-like function, based on the following outline.

Let $m = 6p$ where $p > 3$ is a prime such that the number of internal states of M is $p - 1$ (there is no loss of generality). We represent a state $s = (i, \langle r_1, r_2\rangle)$ by $\widehat{s} = (2^{r_1}3^{r_2}p + i)$. We design the function g so that it takes every encoded state $(\widehat{s}, E)$, in a bounded number of steps, to a $(\widehat{s'}, E)$ where s' is the successor state to s.

In greater detail: first, the point will move to the North axis, possibly increasing r_1 by two; then, move to the West axis, while decreasing r_1 by one; the result is that r_1 has either been incremented or decremented—according to the instruction i, of course. In a similar way, moving to the South axis and subsequently to the East again, an increment or decrement of the second register is simulated.

If M is mortal, every trajectory starting at any encoded state will arrive at a *halting configuration*, represented by a number of the form $(rp + 0, E)$ with $r \in \{1, 2, 3\}$, so the CCP is satisfied. If M is not mortal, it has a non-ending computation, which translates into an infinite trajectory under g.

Since g has to be total, we have to define g for points which do not encode a state, or appear on the trajectory from a state to its successor, as described above; the full proof (omitted here, for lack of space) includes a complete definition of g and argues that our claim that the CCP is satisfied if M is mortal holds when taking all points into account. ◀

▶ **Theorem 15.** *Global convergence and mortality of monic piecewise-affine functions over* $\mathbb{Z}^2$ *are* Π^0_2*-complete problems.*

The theorem is proved by reducing the CCP to the mortality problem for monic PAFs. The details of this construction could not fit in this abstract; but the reader may be able to gain an idea of how it works from the following example, where the classic $3x + 1$ problem is represented by a monic 2-dimensional PAF, whose iteration reaches $(1, 0)$ from initial point $(x, 0)$ if and only if the Collatz sequence from x reaches 1.

Recall that in the $3x + 1$ problem there are just two possible "updates," division by 2 and the mapping $x \mapsto 3x + 1$. This makes it simpler than the Compass problem, where there are four "updates" and a large modulus. However, we still make use of trajectories

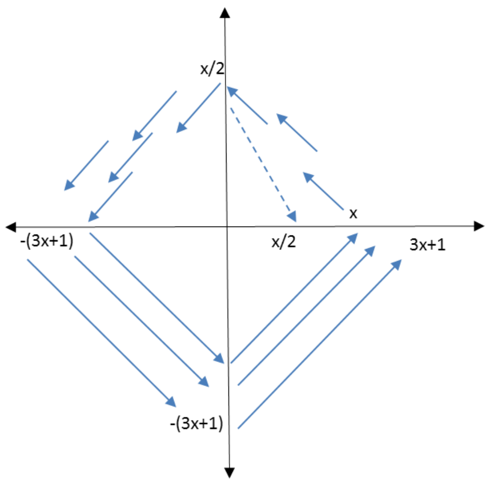


Figure 2 Simulating the $3x+1$ function by a 2-dimensional monic PAF.

that orbit around the origin, starting with the Cartesian point $(x, 0)$ (Figure 2). The NE quadrant carries $(x, 0)$ to $(\lfloor x/2\rfloor, x \bmod 2)$; if the division was even, the point is mapped to

$(x/2, 0)$, ready for the next iteration; otherwise, proceeding counter-clockwise, this point is carried first to $(-(3x+1), 0)$, then to $(0, -(3x+1))$ and finally to $(3x+1, 0)$.

Here is the complete definition of the PAF:

role of region	constraints	$f(x,y)$	region function	constraints	$f(x,y)$
Div by 2	$x{\geq}2,\ y \geq 0$	$(x-2, y+1)$	Compute $3x+1$	$x{<}0\ ,\ y \geq 1$	$(x-6, y-1)$
$x \bmod 2 = 0$	$x{=}0,\ y \geq 0$	(y, x)	West to South	$x{<}0\ ,\ y < 0$	$(x+1, y-1)$
$x \bmod 2 = 1$	$x{=}1,\ y \geq 1$	$(x-5, y)$	South to East	$x{\geq}0\ ,\ y < 0$	$(x+1, y+1)$
			Final zone	$0 \leq x \leq 1,\ y = 0$	(x, y)

5 Decidability and Complexity in One Dimension

Blondel et al. prove that global convergence is decidable for $n = 1$ when the function is continuous. We prove the result for the integers, where continuity is irrelevant. Furthermore, we identify the complexity of the problem, proving it to be PSPACE-complete.

The following examples hint at the difference between mortality over the integers and over the reals. The first function has a fixed point at 10/3 and therefore is not mortal over the reals (or rationals); but it is over the integers.

$$f_1(x) = \begin{cases} -2x+10 & x > 2 \\ 4 & x < 0 \\ 0 & x = 0 \\ 4-2x & 1 \leq x \leq 2 \end{cases} \qquad f_2(x) = \begin{cases} x-3 & x > 2 \\ 4 & x < 0 \\ 0 & x = 0 \\ 4-2x & 1 \leq x \leq 2 \end{cases}$$

Below, we give an algorithm to decide mortality over the integers. We also show that it requires polynomial space (Remark: space complexity is interpreted in the standard way, that is, number of bits used as a function of the number of bits in the input).

Note that in dimension one, the regions are just a partition of $\mathbb{Z}$ into a finite number of intervals. We may assume that these are given explicitly as the list of the end points of closed intervals, e.g.,

$$(-\infty, -3], [-2, 0], [1, +\infty)\,.$$

There will always be one interval infinite to the left, which we denote by $(-\infty, L]$, and one infinite to the right, denoted by $[R, +\infty)$; and by breaking intervals into parts if necessary we can ensure $L < 0$ and $R > 0$. We denote the function in the negative part by $a^- x + b^-$ and the function in the positive part by $a^+ x + b^+$.

We may also assume $f(0) = 0$ since, for the convergence problem, the answer is negative if this does not hold, while for mortality, we may as well modify the function so that $f(0) = 0$.

First, we define

$$\rho_0 = \max(\{R\} \cup \{f(x), \text{ where } L \leq x \leq R\})$$
$$\lambda_0 = \min(\{L\} \cup \{f(x), \text{ where } L \leq x \leq R\})$$

We note that this value can be easily calculated in polynomial time, since for each finite interval, $f(x)$ assumes the maximum (or minimum) at one of its ends. These values show how far away from 0 one can get without using the infinite regions.

Next, we show that one can efficiently either determine that the dynamic system is *divergent*, which means that there is an unbounded trajectory; or find a finite attractor, an interval such that all trajectories eventually stay within it.

▶ **Lemma 16.** *Suppose that at least one of a^+, a^- is non-negative. If either a^+ or a^- is bigger than 1; or $a^+ = 1$ and $b^+ \geq 0$; or $a^- = 1$ and $b^- \leq 0$, then f is divergent, and not mortal. Otherwise, it has the finite attractor $A = [\lambda, \rho]$ where*

$$\rho = \max(\rho_0,\ (\lambda_0 - |b^-|) \cdot \min(a^-, 0))$$
$$\lambda = \min(\lambda_0,\ (\rho_0 + |b^+|) \cdot \min(a^+, 0))\,.$$

The algorithm for analysing mortality is now as follows. First, if both a^+ an a^- are negative, we construct (in polynomial time) a representation of $f \circ f$, whose asymptotic behaviour is the same, and has positive coefficients in the infinite regions; we thus assume that Lemma 16 is applicable. By testing the conditions stated in the lemma, we either conclude immediately that f is not mortal, or get the attactor A. In the latter case, we now proceed to trace, for each point in this interval, the trajectory from this point, until finding that it reaches zero, or that it cycles without meeting the origin—so we know if f is mortal. It is not hard to verify that this can be accomplished in polynomial space. We conclude

▶ **Theorem 17.** *Global convergence over $\mathbb{Z}$ of a piecewise affine function f with integer coefficients is a PSPACE problem.*

It may be interesting to note that the decision procedure for the one-dimensional case in [3] is much simpler, and in fact polynomial-time (for rational coefficients in a standard representation). Keep in mind, however, that it solves a different problem (a continuous state space and a continuous function), which is neither a sub-problem nor a super-problem of the problem we consider. In fact, for our problem, our problem turns out to be PSPACE-hard. To prove it, we next introduce several Turing-machine variants as intermediate representations, and show a sequence of reductions, starting with a standard PSPACE-hard problem and culminating in our mortality problem (in this extended abstract, the more technical steps have been omitted to save space).

The first machine is a restricted form of *linearly bounded automaton* (LBA) [21].

▶ **Definition 18.** A *tally LBA* is a single-tape machine that receives as input a string of the form 0^n for some $n \geq 0$; this string is initially written on its work tape, delimited by endmarkers. The machine is guaranteed to never move beyond the endmarkers. The tape alphabet is binary.

In the next definition, we consider the tally string to be part of the machine's description: hence, a given machine only performs a single computation.

▶ **Definition 19.** A *fixed-space machine with oblivious queue access* is a Turing machine with the following features:

1. The work tape is a *queue* of a fixed capacity n (which we consider to be given, in tally form, as part of the standard description of such a machine). In every step the machine strips a symbol from the front of the queue and adds one to the rear. Hence, an instruction of the machine is given by a 4-tuple (q, b, q', b'), where:
   q is the current control state, q' the next;
   b the symbol read off the queue, b' the symbol appended to the queue.
   We use the customary symbol δ for the set of these 4-tuples.
2. The work-tape (queue) alphabet is binary.
3. The initial contents of the queue are 0^n.

▶ **Definition 20.** A *fixed-space machine with oblivious queue access and a clock*, briefly a fixed-space Q&C machine, is a Turing machine with the following features:

1. The work tape is a *queue*, as in the previous definition.
2. The machine also has a *clock*. This device is a counter (a register of non-negative integer value) that is automatically decremented each time the machine has completed another cycle through the queue (that is, exactly n transitions). If the clock reaches zero, the machine is reset: the queue is cleared, the control state is reset to an initial state (0) and the clock is reset to its initial value.
3. The initial contents of the queue are 0^n; the initial clock value is given—in binary—as part of the standard description of such a machine.

▶ **Lemma 21.** *The halting problem of tally LBAs is a PSPACE-hard problem.*

▶ **Lemma 22.** *The computation of a tally LBA on input 0^n can be simulated by a fixed-space machine with oblivious queue access, starting on a blank queue of capacity $2n$.*

Mortality of fixed-space machines is defined as usual: halting when started with any possible configuration. Note that there are finitely many such configurations, due to the fixed space.

▶ **Lemma 23.** *The halting problem for fixed-space Turing machines with oblivious queue access can be reduced, in logarithmic space, to mortality of a fixed-space Q&C machine.*

Proof. If the given machine ever halts, the length of its computation must be bounded by $2^n \cdot m$, where n is the queue size and m the number of control states. We let this be the initial value of the clock, so that if the machine does halt, it will halt before the clock times out. If the machine does not halt, it will eventually time out, then restart, ad infinitum. To see that a halting machine is transformed into a mortal one, note that regardless of its initial configuration, the machine will either halt or time out, and from that point on, faithfully simulate the given input-free machine. ◀

▶ **Lemma 24.** *Mortality of fixed-space Turing machines with oblivious queue access is PSPACE-hard.*

▶ **Theorem 25.** *Global convergence over $\mathbb{Z}$ of a piecewise affine function f with integer coefficients is PSPACE-hard (under logspace reductions).*

Proof. We reduce from mortality of fixed-space machines with oblivious queue access. The reduction transforms a machine M, given with its space bound n, into a representation of a piecewise-affine function f that simulates M and is mortal if and only if M is.

Let us first describe the essence of this simulation. Suppose that M has m states. A configuration of M is specified by (q, w) where $q \in [0, m-1]$ is the control state, and $w \in \{0,1\}^n$ is the contents of the queue.

By identifying w with the integer that it represents in binary notation, we define an integer that encodes a configuration:

$$\langle q, w\rangle = q \cdot 2^n + w\,. \tag{3}$$

Next, we define f as a function that maps a configuration to the next, simulating the machine; we present the definition by cases.

For each $(q, b, q', b') \in \delta$, we define $\qquad f(\langle q, bz\rangle) = \langle q', zb'\rangle\,. \qquad (4)$

If there is no transition starting with (q, b), $\qquad f(\langle q, bz\rangle) = 0. \qquad (5)$

We now verify that the above definitions yield a piecewise-affine function. Since this is a one-dimensional PAF, its regions of definition are intervals and we use the notation $a + [b, c]$ for $[a + b, a + c]$. Let $R_{q,b}$ be the interval $q \cdot 2^n + b \cdot 2^{n-1} + [0, 2^{n-1} - 1]$. The function f is defined on each such interval according to the applicable case; specifically, every transition described by (4) is translated into: $x \in R_{q,b} \Rightarrow f(x) = 2(x - q \cdot 2^n - b \cdot 2^{n-1}) + q' \cdot 2^n + b'$ while if there is no such transition, $x \in R_{q,b} \Rightarrow f(x) = 0$. ◀

6 Similar Problems

There are several natural problems similar to mortality and global convergence, and results fall out of the same reductions (mostly). Here are two examples. The first behaves just the same as mortality (Π_2^0-complete in two dimensions, PSPACE-complete in one), while the second differs in the one-dimensional case being in PTIME.

Global convergence to a fixed point [13].
Input: f, a PAF. Question: do all trajectories of f reach a fixed point?

Convergence to a Finite Set (In Dynamical System parlance: *Finite Attractor*).
Input: f, as previously. Question: is there a finite set S such that every sequence $x_{t+1} = f(x_t)$ stays within S for all t large enough?

7 Conclusion and Open Problems

We have presented work on the border between Dynamical System Theory (much of which refers to continuous state-spaces) and the Theory of Computation in discrete models. In particular, this work connects Dynamical System Theory to problems of program termination. I'd like to argue that such results may be of interest in the context of continuous-space dynamical systems (e.g., as models of some kinds of physical systems), since, unlike previous proofs, the undecidability is not derived from the encoding of information into the fractional bits of a value, which boils down to assuming unlimited precision in measurement.

The focus of this work has been on deciding global properties of systems, so we have not dwelled on the problem that corresponds to the common Turing-machine *halting problem*, namely: given a function and an initial value x_0, does the sequence beginning with x_0 reach zero. However it is easy to see, from our proofs, that this problem is RE-complete in two dimensions, and PSPACE-complete in one. The problem: does the sequence beginning with x_0 reach a given value y? Is known as the *orbit problem* and was solved in polynomial time for linear transformations over $\mathbb{Q}^n$ [11].

I hope that the results presented are of interest, but also the techniques and connections made to Collatz-like problems and to automata that capture PSPACE. Finally, here are some open problems.

1. Is there any constant bound on the number of regions which suffices to make mortality of monic piecewise-affine functions over $\mathbb{Z}^2$ as hard as the general problem, i.e., Π_2^0 hard?
2. (Braverman's problem) Is mortality decidable for functions which are zero outside a single convex region (and affine within)?
3. What is the complexity of the mortality problem in the one-dimensional case, when the coefficient of x is always 1?
4. What is the complexity of the mortality problem in the one-dimensional case, when the number of intervals is considered a constant?

Acknowledgement. I thank Jakob Grue Simonsen for a helpful discussion.

References

1 Amir M. Ben-Amram and Samir Genaim. On the linear ranking problem for integer linear-constraint loops. In *Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM press, 2013.

2 Amir M. Ben-Amram, Samir Genaim, and Abu Naser Masud. On the termination of integer loops. *ACM Transactions on Programming Languages and Systems*, to appear.

3 Vincent D. Blondel, Olivier Bournez, Pascal Koiran, Christos H. Papadimitriou, and John N. Tsitsiklis. Deciding stability and mortality of piecewise affine dynamical systems. *Theor. Comput. Sci.*, 255(1-2):687–696, 2001.

4 Vincent D. Blondel and John N. Tsitsiklis. A survey of computational complexity results in systems and control. *Automatica*, 36(9):1249–1274, 2000.

5 Mark Braverman. Termination of integer linear programs. In Thomas Ball and Robert B. Jones, editors, *Computer Aided Verification, 18th International Conference, CAV 2006*, volume 4144 of *Lecture Notes in Computer Science*, pages 372–385. Springer, 2006.

6 Michael Brin and Garrett Stuck. *Introduction to dynamical systems*. Cambridge University Press, Cambridge, UK, 2002.

7 Serge Burckel. Functional equations associated with congruential functions. *Theoretical Computer Science*, 123(2):397–406, January 1994.

8 Michael Colón and Henny Sipma. Synthesis of linear ranking functions. In *7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2031 of *Lecture Notes in Computer Science*, pages 67–81. Springer, 2001.

9 John. H. Conway. Unpredictable iterations. In *Proc. 1972 Number Theory Conf., Univ. Colorado, Boulder*, pages 49–52. 1972. Reprinted with historical commentary in [15].

10 John. H. Conway. FRACTRAN: a simple universal programming language for arithmetic. In T. M. Cover and B. Gopinath, editor, *Open Problems in Communication and Computation*, pages 3–27. Springer-Verlag, 1987. Reprinted with historical commentary in [15].

11 R. Kannan and R. J. Lipton. Polynomial-time algorithm for the orbit problem. *J. ACM*, 33(4):808–821, October 1986.

12 František Kaščák. Small universal one–state linear operator algorithm. In Ivan M. Havel and Vaclav Koubek, editors, *Mathematical Foundations of Computer Science (MFCS '92)*, volume 629 of *LNCS*, pages 327–335, Springer, 1992.

13 Pascal Koiran, Michel Cosnard, and Max Garzon. Computability with low-dimensional dynamical systems. *Theor. Comput. Sci.*, 132:113–128, September 1994.

14 Stuart A. Kurtz and Janos Simon. The undecidability of the generalized Collatz problem. In Jin-Yi Cai, S. Barry Cooper, and Hong Zhu, editors, *Theory and Applications of Models of Computation, 4th International Conference, TAMC 2007, Shanghai, China, May 22-25, 2007*, volume 4484 of *Lecture Notes in Computer Science*, pages 542–553. Springer, 2007.

15 Jeffrey C. Lagarias. *The Ultimate Challenge: The* $3x+1$ *Problem*. American Mathematical Society, 1st edition, 2010.

16 Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, USA, 1967.

17 Cristopher Moore. Unpredictability and undecidability in dynamical systems. *Phys. Rev. Lett.*, 64(20):2354–2357, May 1990.

18 Andreas Podelski and Andrey Rybalchenko. A complete method for the synthesis of linear ranking functions. In *VMCAI*, pages 239–251, 2004.

19 H.T. Siegelmann and E.D. Sontag. Analog computation, neural networks, and circuits,. *Theor. Comp. Sci.*, 131:331–360, 1994.

20 Ashish Tiwari. Termination of linear programs. In Rajeev Alur and Doron Peled, editors, *Computer Aided Verification*, volume 3114 of *Lecture Notes in Computer Science*, pages 387–390. Springer Berlin / Heidelberg, 2004.

21 K. Wagner and G. Wechsung. *Computational Complexity*. D. Reidel, Dordrecht, 1986.

On the practically interesting instances of MAXCUT

Yonatan Bilu[1], Amit Daniely[*2], Nati Linial[†3], and Michael Saks[‡4]

1 **Parasight inc**
Agudat sport hapoel 1, Jerusalem, Israel.
`yonatan@gmail.com`

2 **Department of Mathematics, Hebrew University**
Jerusalem 91904, Israel.
`amit.daniely@math.huji.ac.il`

3 **School of Computer Science and Engineering, Hebrew University**
Jerusalem 91904, Israel.
`nati@cs.huji.ac.il`

4 **Department of Mathematics, Rutgers University**
Piscataway, NJ 08854.
`saks@math.rutgers.edu.`

Abstract

For many optimization problems, the instances of practical interest often occupy just a tiny part of the algorithm's space of instances. Following [6], we apply this perspective to MAXCUT, viewed as a clustering problem. Using a variety of techniques, we investigate practically interesting instances of this problem. Specifically, we show how to solve in polynomial time distinguished, metric, expanding and dense instances of MAXCUT under mild stability assumptions. In particular, $(1+\epsilon)$-stability (which is optimal) suffices for metric and dense MAXCUT. We also show how to solve in polynomial time $\Omega(\sqrt{n})$-stable instances of MAXCUT, substantially improving the best previously known result.

1998 ACM Subject Classification F.2.0 Analysis of algorithms and problem complexity

Keywords and phrases MAXCUT, Clustering, Hardness in practice, Stability, Non worst-case analysis

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.526

1 Introduction

As has been noted many times, worst case complexity is often an overly restrictive metric for algorithms. In practice, a more realistic (but fuzzy) criterion would be to say that a problem is feasible if there is an efficient algorithm that correctly solves all of its *practically interesting* instances. The difference can be very substantial, since for many computational problems, the vast majority of instances are completely irrelevant for practical purposes.

An important case in point is clustering, where one seeks a meaningful partition of a given set of data. Almost every formal manifestation of the clustering problem is NP-Hard,

* Supported in part by a binational Israel-USA grant 2008368 and by a Google Europe Fellowship in Learning Theory.
† Supported in part by a binational Israel-USA grant 2008368.
‡ Supported in part by NSF under grants CCF-0832787 and CCF-1218711, and by a binational Israel-USA grant 2008368. Part of this work was done while on sabbatical at Princeton University.

yet, a clustering instance is of practical interest only if the data *can indeed* be partitioned in a meaningful way, and such data sets are very special. Thus, even if no efficient algorithm can find the optimal partition for *every* data set, this does not imply that clustering is hard in practice. As Tali Tishby put it in conversation many years ago, many practitioners hold the opinion that "clustering is either easy or pointless". That is, for a data sets that admit a meaningful partition of the data, finding it is not hard.

Bilu and Linial [6] proposed a framework for studying this issue which applies to optimization problems with a continuous input space and discrete solution space. They proposed two criteria for an optimal solution to be *evidently* optimal. An optimal solution is *stable* if it remains optimal under moderate perturbations of the input. An optimal solution is *distinguished* if the objective function value at any other point is reduced by at least an amount proportional to the distance to the optimal point.

Following [6] we study the (weighted) MAXCUT problem in this framework. Here the input is a weighted graph and the candidate solutions are cuts. A cut is γ-stable (for $\gamma \geq 1$) if it remains optimal even if each input weight w_{ij} is perturbed to a value between w_{ij} and γw_{ij}. A cut is α-*distinguished* (for $\alpha \geq 0$) if moving to any other cut reduces the objective function by at least α times the sum of (weighted) degrees of the vertices that switched side. We also consider a weakening of stability called γ-*local stability.*

Our main results are:

▶ **Theorem 1.** *1. For every $\epsilon > 0$ there is a polynomial time algorithm that correctly solves all $(1+\epsilon)$-locally stable instances of Metric-MAXCUT.*

2. For every $\epsilon > 0$ and $C > 1$ there is a polynomial time algorithm that correctly solves all MAXCUT instances that are $(1+\epsilon)$-locally stable and C-dense.

The condition of C-density (defined in Section 1.2) rules out weight being too concentrated.

▶ **Theorem 2.** *There is a polynomial time algorithm that solves every instance of MAXCUT that is α-distinguished and γ-locally stable with $\gamma > \frac{2}{1-\sqrt{1-\alpha^2}}$. In fact, it suffices that the instance be γ-locally stable with $\gamma > \frac{2}{1-\sqrt{1-h^2}}$, where h is the Cheeger constant of the weighted graph induced by the maximal cut.*

This improves a result from [6] that works only for regular graphs and requires that $\gamma > \frac{5+\sqrt{1-\alpha^2}}{1-\sqrt{1-\alpha^2}}$ or $\gamma > \frac{5+\sqrt{1-h^2}}{1-\sqrt{1-h^2}}$.

▶ **Theorem 3.** *There is a polynomial time algorithm that finds the optimal solution for every $\Omega(\sqrt{n})$-stable instance of MAXCUT.*

This improves on a result in [6] which needed $\Omega(n)$-stability.

Some notation and terminology

The input to the MAXCUT problem on vertex set V is a symmetric weight function $w : V \times V \to \mathbb{R}^+$ with 0 diagonal. Expressions such as "w is bipartite" refer to the graph which is the support of w, which we assume to be connected. The objective is to find the *cut* $(S, \bar{S})$, $S \subseteq V$ of maximum total weight $\sum_{a\in S,\ b\in\bar{S}} w(a,b)$.

For a fixed cut $(S, \bar{S})$, we use the self-explanatory terms “the vertices x, y are *on the same side*" or *“separated"* by this cut. The edge xy a *cut edge* or a *non-cut edge* when x, y are separated resp. on the same side of the cut. For $A, B \subset V$, we define $E(A,B) = \{ab | a \in A,\ b \in B\}$, and $w(A,B) := \sum_{uv\in E(A,B)} w(u,v)$. Also, $\tau_w(A) = \tau(A) = w(A, \bar{A})$ and $\mu(A) = \mu_w(A) = w(A, V)$. Fo $A \subseteq V$, $\xi(A)$, $\xi(A) = \sum_{vu\in E(A,\bar{A})\cap E(S,\bar{S})} w(u,v)$ is the sum of weights of cut edges leaving A and $\iota(A) = \tau(A) - \xi(A)$ is the weight of the non-cut edges.

(The reader may find the following mnemonic useful: τ stands for "total", ξ for "external" and ι for "internal"). We slightly abuse notation for singletons $A = \{v\}$ and pairs $A = \{u, v\}$ and write $\tau(v)$ or $\iota(e)$ etc., where $e = uv$. The *minimal, maximal* and *average* degree of w are denoted by $\underline{\delta}(w) = \min_{v\in V} \mu(v)$, $\bar{\delta}(w) = \max_{v\in V} \mu(v)$ and $\delta(w) = \frac{\sum_{v\in V} \mu(v)}{n}$ respectively.

1.1 Stable instances

▶ **Definition 4.** Let $w : V \times V \to [0, \infty)$ be an instance of MAXCUT and let $\gamma \geq 1$. An instance $w' : V \times V \to [0, \infty)$ is a γ*-perturbation* of w if

$$\forall u, v \in V,\ w(u, v) \leq w'(u, v) \leq \gamma \cdot w(u, v)$$

An instance w is said to be γ*-stable* if there is a cut which forms a maximal cut for every γ-perturbation w' of w.

▶ **Definition 5.** Let $\gamma \geq 1$. An instance $w : V \times V \to [0, \infty)$ for MAXCUT is γ*-locally stable* if there is a maximal cut $(S, \bar{S})$ for which it is impossible to obtain a larger cut by switching the side of some vertex x and multiplying the edges in $E(x, V \setminus \{x\})$ by numbers between 1 and γ.

The definitions of stability and local stability capture the intuition of an "evidently optimal" solution. The following more concrete equivalent definitions are usually more convenient to use.

▶ Observation 1. [6] Let $w : V \times V \to \mathbb{R}$ be an instance of MAXCUT and let $\gamma \geq 1$.

- w is γ-stable iff there is a maximal cut for which $\xi(A) \geq \gamma \cdot \iota(A)$ for every $A \subset V$.
- w is γ-locally stable iff there is a maximal cut for which $\xi(x) \geq \gamma \cdot \iota(x)$ for every $x \in V$.

We say that a (not necessarily maximal) cut $(S, \bar{S})$ is γ*-stable* (resp. γ*-locally stable*) if the first (resp. second) condition in Observation 1 holds.

Every instance is 1-stable and it is easy to see that there is a unique maximal cut if and only if the instance is γ-stable for some $\gamma > 1$.

Stability and local stability are quite different. For $\gamma > 1$ an instance has at most one γ-stable cut but may have many γ-locally stable cuts. The instance where $w = 1$ on the edges of a perfect matching and $\epsilon > 0$ elsewhere. As $\epsilon \to 0$, the local stability tends to ∞ and has exponentially many γ-locally stable maximal cuts, but is not γ-stable for any $\gamma > 1$. While MAXCUT remains NP-hard even under arbitrarily high local stability (see [6]), in Section 2 we prove Theorem 3 by giving an efficient algorithm for $\Omega(\sqrt{n})$-stable instances. Also, it is easy to decide whether a given cut is γ-locally stable, but we do not know how to decide whether a given cut is γ-stable.

1.2 Metric and Dense instances

In Section 3 we study metric instances. This is done through a reduction from metric to dense instances, so we consider such instances as well (Section 3.1).

We call $w : V \times V \to \mathbb{R}$ C**-dense** for $C \geq 1$ if $\forall x, y \in V,\ w(x, y) \leq C \cdot \frac{\tau(x)}{n}$. As shown in [2], for $C > 1$ fixed, C-dense MAXCUT is NP-Hard, but it has a PTAS. As we show, this PTAS can be adapted to correctly solve all instances of MAXCUT that are $(1 + \epsilon)$-locally stable and C-dense for every $\epsilon > 0, C > 1$. The algorithm samples $O(\log n)$ vertices and tests each of their bipartitions as a seed to a cut. As we show, w.h.p., one of the resulting cuts is the maximal cut, proving the second part of Theorem 1.

In Section 3.2 we deal with Metric-MAXCUT. As shown in [15] (with credit to L. Trevisan) Metric-MAXCUT is NP-Hard. That paper also gives a reduction from metric to $(4 + o(1))$-dense instances of MAXCUT, thus yielding a PTAS for Metric-MAXCUT. We prove Theorem 1 by showing that a slight variation of this reduction preserves local stability[1], and therefore yields an efficient algorithms for $(1 + \epsilon)$-locally stable instances of Metric-MAXCUT.

The exponent for this algorithm is quite large. We also provide a faster algorithm for $(3 + \epsilon)$-locally stable metric instances.

1.3 Distinguished and Expanding instances

Let $w : V \times V \to \mathbb{R}^+$ be an instance of MAXCUT whose (unique) maximal cut is $(S, \bar{S})$. We note that if all vertices of $A \subset V$ switch side, then the weight of the cut decreases by $\xi(A) - \iota(A)$. Thus, we define

▶ **Definition 6.** An instance w of MAXCUT is *α-distinguished* for $1 \geq \alpha \geq 0$ if for every $\emptyset \neq A \subset V$, $\xi(A) - \iota(A) \geq \alpha \cdot \min\{\mu(A), \mu(\bar{A})\}$.

Note that every instance is 0-distinguished and being α-distinguished with $\alpha > 0$ is equivalent to having a unique maximal cut. It is not hard to see that $\frac{1+\alpha}{1-\alpha}$-local stability is equivalent to α-*local distinction*, namely $\xi(x) - \iota(x) \geq \alpha \cdot \mu(x)$ for every $x \in V$.

Distinction vs Stability. Let $(S, \bar{S})$ be a maximal cut of $w : V \times V \to [0, \infty)$. On the one hand, every α-distinguished instance is $\frac{1+\alpha}{1-\alpha}$-stable, because $\xi(A) - \iota(A) \geq \alpha\mu(A) \geq \alpha(\xi(A) + \iota(A))$. On the other hand, highly stable instances need not be distinguished as the following bipartite example with $V = \{a_1, \ldots, a_n\} \dot\cup \{b_1, \ldots, b_n\}$ shows. Here $w(a_i, b_j)$ is 1 when $i = j$ and $\epsilon \ll 1$ otherwise. Clearly w is ∞-stable. Yet, switching the sides of all the vertices in $\{a_1, \ldots, a_{\frac{n}{2}}\} \cup \{b_1, \ldots, b_{\frac{n}{2}}\}$ decreases the weight of the cut only slightly. Such examples motivate the stronger notion of distinction. Although the cut $(\{a_1, \ldots, a_n\}, \{b_1, \ldots, b_n\})$ is infinitely stable, its optimality does not seem completely evident.

Distinction and Expansion. Call $w : V \times V \to \mathbb{R}^+$ *β-expanding* if $\beta \leq h(w)$ where $h(w) = \min_{\emptyset \neq A \subset V} \frac{\tau(A)}{\min\{\mu(A), \mu(\bar{A})\}}$ is w's *Cheeger constant*. An α-distinguished instance is α-expanding, though highly expanding instances can even have multiple maximal cuts. However, an instance that is both γ-stable and β-expanding is easily seen to be $(\beta \cdot \frac{\gamma-1}{\gamma+1})$-distinguished. As this discussion implies, distinction is a conjunction of stability and expansion.

In section 4 we prove Theorem 2, using a spectral result from [6].

1.4 Spectral algorithms

In Section 5 we consider consider whether the Goemans-Williams approximation algorithm can be used to exactly solve instances of high stability or local stability.

1.5 Other related work

Recently, Balcan and Liang [5] introduced a relaxed version of stability in which the optimal solution is allowed to change slightly under perturbations of the input, and obtained good algorithms for clustering under the k-medians objective. In [3] polynomial time algorithms are given for 3-stable instances of k-means, k-medians and other "center based" clustering

[1] A word of caution: Our definition of stability and local stability for Metric-MAXCUT is more restrictive than one might think. We require the perturbed instance to satisfy the stability condition *whether or not* it is metric.

problems. The constant 3 was improved in [5] to $(1+\sqrt{2})$ for k-median. The papers [9, 1, 4] consider data sets that admit a good clustering and show how to cluster them efficiently.

Smoothed analysis is the best known example of a method for analysing instances of computational problems based on their practical significance. As this method shows [14], a certain variant of the simplex algorithm solves in polynomial time *almost every* input.

The MAXCUT problem has been shown to solvable in polynomial time with high probability in random models (e.g. [7]) and *semirandom model [10].*

2 Algorithms for stable instances

▶ Observation 2. Let w be a γ-stable instance of MAXCUT, and let w' be obtained from w by merging two vertices[2] on the same side of w's maximal cut. Then w' is γ-stable and its maximal cut is induced from w's maximal cut.

Thus, it suffices to give an efficient algorithm that, for any γ-stable instance, finds a pair of vertices on the same side of the optimal cut. Once two such vertices are found, we merge them and proceed recursively. This applies as well when γ is a non-decreasing function of n.

As an easy warm-up, we show how to find such a pair of vertices in a $2n$-stable MAXCUT instance w, simplifying an algorithm from [6]. Let vu be a heaviest edge, and let vz be the heaviest edge incident on either u or v. We claim that both vu and vz are cut edges and so u and z are on the same side of the cut. To see this Clearly $w(v,u) \geq \frac{1}{n-1}\tau(v)$. By observation 1, $\iota(v) \leq \frac{1}{2n+1}\tau(v)$, so $w(v,u) > \iota(v)$ and we conclude that vu is a cut edge. Again, $w(v,z) \geq \frac{1}{2(n-2)}\tau(\{u,v\})$ and by observation 1, $\iota(\{v,u\}) \leq \frac{1}{2n+1}\tau(\{v,u\})$, implying that $w(v,z) > \iota(\{v,u\})$. Consequently vz is a cut edge.

2.1 A deterministic algorithm for $O(\sqrt{n})$-stable instances

Following observation 2, we will find two vertices which are on the same side of the cut. To find this pair, we'll only need the condition in Observation 1 to hold for subsets $A \subseteq V$ with $|A| \leq 2$. (But full stability is needed to apply induction after merging.) Let w be a γ-stable instance of MAXCUT with $\gamma > \sqrt{8n+4}+1$ and let $(S,\bar{S})$ be a maximal cut. We first deal with very heavy edges. Let T^1 be the set of edges vu for which $w(v,u) > \mu(v)/(\gamma+1)$. By observation 1, all edges in T^1 are cut edges. Thus if there are two incident edges $uv, vz \in T^1$, then u and z are on the same side of the cut and we are done. It remains to consider the case where T^1 is a matching. Let T^2 be the set of edges not in T^1 that satisfy $w(u,v) > \tau(\{u,z\})/(\gamma+1)$ for some $uz \in T^1$. Again, by observation 1, all edges in T^2 are cut edges. If T^2 is nonempty, say $uv \in T^2$, then there exists some $uz \in T^1$ with $w(u,v) > \frac{1}{\gamma+1}\tau(\{u,z\})$, which implies that v and z are on the same side of the cut. We proceed to consider the case where T^2 is empty.

For every $u,v \in V$ define

$$\tilde{w}(u,v) = \begin{cases} 0 & vu \in T^1 \\ w(u,v) & o/w \end{cases}, \quad \hat{w}(v) = \begin{cases} \tau(\{u,v\}) & vu \in T^1 \text{ for some } u \in V \\ \tau(v) & o/w \end{cases}$$

Note that $\hat{w}(v)$ is well defined, since T^1 is a matching by assumption. Since $T^2 = \emptyset$ and T^1 is a matching, we have, for every $u \in V$, $\tilde{w}(v,u) \leq \frac{1}{\gamma+1}\hat{w}(v)$ and, again by observation

[2] Let $w: V \times V \to \mathbb{R}$ be an instance and let $v,u \in V$. The instance $w': V' \times V' \to \mathbb{R}$ obtained upon **merging** v,u is defined as follows. $V' = V \setminus \{u,v\} \cup \{v'\}$ and $w'(x,y) = w(x,y)$ for $x,y \in V \setminus \{v,u\}$, also, $w'(v',x) = w(v,x) + w(u,x)$.

1, $\iota(v) \leq \frac{1}{\gamma+1}\hat{w}(v)$. Next, we observe as well that separated vertices cannot have too many common neighbours. For $u, v \in V$ we define $n(u,v) := \sum_{z\in V} \tilde{w}(v,z)\tilde{w}(z,u)$. If v and u are separated, say $v \in S, u \in \bar{S}$, then

$$\begin{aligned} n(u,v) &= \sum_{z\in\bar{S}} \tilde{w}(v,z)\tilde{w}(z,u) + \sum_{z\in S} \tilde{w}(v,z)\tilde{w}(z,u) \\ &\leq \frac{1}{\gamma+1}\hat{w}(v)\cdot\iota(u) + \frac{1}{\gamma+1}\hat{w}(u)\cdot\iota(v) \leq \frac{2}{(\gamma+1)^2}\hat{w}(u)\cdot\hat{w}(v). \end{aligned}$$

Thus, it suffices to find two vertices v, u with $n(u,v) > \frac{2}{(\gamma+1)^2}\hat{w}(u)\cdot\hat{w}(v)$, and place them on the same side of the cut. Indeed, if no such pair exists we have

$$\begin{aligned} \frac{1}{4}\sum_{v\in V}\hat{w}^2(v) &\leq \sum_{v\in V}\tau_{\tilde{w}}^2(v) = \sum_{u,v,z\in V}\tilde{w}(u,z)\tilde{w}(z,v) \\ &= \sum_{u,v\in V,\ u\neq v} n(u,v) + \sum_{u,z\in V}\tilde{w}^2(u,z) \\ &\leq \frac{2}{(\gamma+1)^2}\sum_{u,v\in V,\ u\neq v}\hat{w}(u)\hat{w}(v) + \sum_{u\in V}\frac{1}{\gamma+1}\hat{w}(u)\sum_{z\in V}\tilde{w}(u,z) \\ &\leq \frac{2}{(\gamma+1)^2}(\sum_{u\in V}\hat{w}(u))^2 + \frac{1}{\gamma+1}\sum_{u\in V}\hat{w}(u)\tau_{\tilde{w}}(u) \\ &\leq \frac{2n}{(\gamma+1)^2}\sum_{u\in V}\hat{w}^2(u) + \frac{1}{\gamma+1}\sum_{u\in V}\hat{w}^2(u), \end{aligned}$$

from which we obtain the contradiction $\gamma \leq \sqrt{8n+4}+1$.

3 Algorithms for locally stable dense and metric instances

3.1 Dense instances

▶ **Theorem 7.** *For every $C \geq 1$ and $\epsilon > 0$ there is a randomized polynomial time algorithm that correctly solves all $(1+\epsilon)$-locally stable, C-dense instances of MAXCUT.*

The analysis of the algorithm is based on the following lemma.

▶ **Lemma 8.** *Suppose that $w : V \times V \to [0,\infty)$ is a C-dense instance and let $(S,\bar{S})$ be a γ-locally stable cut. Let $X_1,\ldots,X_m$ be i.i.d. r.v. that are uniformly distributed on V. For $x \in V$, let A_x be the event that $S_+ > S_-$, where $S_\pm = \sum w(x,X_i)$ over all i s.t. x and X_i are separated resp. on the same side. Then*

$$\Pr\left(\cup_x A_x\right) \leq |V|\cdot\exp\left(-\frac{1}{2}\left(\frac{1}{C}\cdot\frac{\gamma-1}{\gamma+1}\right)^2\cdot m\right)$$

Proof. For every $x \in V$, $S_+ - S_-$ is a sum of m i.i.d. r.v.'s of expectation $\frac{\xi(x)-\iota(x)}{|V|} \geq \frac{\gamma-1}{\gamma+1}\frac{\tau(x)}{|V|}$. These r.v.'s are bounded in absolute value, by $C\cdot\frac{\tau(x)}{|V|}$. Now apply Hoeffding's bound.

◻

Proof. (Of Theorem 7) Let $D = 2\left(C\cdot\frac{2+\epsilon}{\epsilon}\right)^2$ and $m = D\cdot\ln(2|V|)$. Let $X_1,\ldots,X_m$ be independent uniform random samples from V. By Lemma 8, with probability ≥ 0.5, there is a partition $\{X_1,\ldots,X_m\} = L\coprod R$ such that the cut defined by $S = \{x \in V : w(x,R) > w(x,L)\}$ is optimal. Now simply enumerate over the $(2\cdot|V|)^{\ln(2)D} = n^{O(1)}$ such partitions.

3.2 Metric instances

Given an instance $w : V \times V \to [0, \infty)$ of MAXCUT, we split its vertices as follows. Pick a set $\tilde{V}$ and a surjective map $\pi : \tilde{V} \to V$. A MAXCUT instance $\tilde{w}$ on $\tilde{V}$ is defined by $\tilde{w}(\tilde{x}, \tilde{y}) = \frac{w(x,y)}{|\pi^{-1}(x)|\cdot|\pi^{-1}(y)|}$, where $\pi(\tilde{x}) = x, \pi(\tilde{y}) = y$.

▶ Proposition 9. Consider the map $(S, \bar{S}) \mapsto (\pi^{-1}(S), \pi^{-1}(\bar{S}))$ from cuts of w to cuts of $\tilde{w}$.

1. This map preserves weights, stability and local stability of cuts.
2. Restricted to the locally stable cuts (i.e., γ-locally stable cuts with $\gamma > 1$), this is a bijection *onto* the locally stable cuts of $\tilde{w}$.
3. It maps maximal cuts to maximal cuts.
4. If $w(V, V) = 2 \cdot |V|^2$ and if the preimage of every $x \in V$ has cardinality $\lfloor \tau_w(x) \rfloor$ then $\tilde{w}$ is $(4 + o(1))$-dense.

Proof. The first three items are easy to show, so we proceed with the last claim whose proof is essentially due to [15]. Let $\tilde{x}, \tilde{y} \in \tilde{V}$ such that $\pi(\tilde{x}) = x, \pi(\tilde{y}) = y$. It is easy to see that (see [15]) $2 \cdot |V| \cdot \tau_w(x) \geq w(V, V)$, $\lfloor \tau_w(x) \rfloor \geq \left(1 - \frac{1}{|V|}\right) \tau_w(x) = (1 - o(1))\tau_w(x)$, $\tau_{\tilde{w}}(\tilde{x}) = \frac{\tau_w(x)}{\lfloor \tau_w(x) \rfloor} \geq 1$ and $w(x, y) \leq \frac{1}{|V|}(\tau_w(x) + \tau_w(y))$. Thus, we have

$$\begin{aligned}
\tilde{w}(\tilde{x}, \tilde{y}) &= \frac{w(x,y)}{\lfloor \tau_w(x) \rfloor \cdot \lfloor \tau_w(y) \rfloor} \leq (1 + o(1)) \frac{w(x,y)}{\tau_w(x) \cdot \tau_w(y)} \\
&\leq (1 + o(1) \frac{\frac{1}{|V|}[\tau_w(x) + \tau_w(y)]}{\tau_w(x) \cdot \tau_w(y)} = (1 + o(1)) \cdot \left(\frac{1}{|V|\tau_w(x)} + \frac{1}{|V|\tau_w(y)} \right) \\
&\leq (1 + o(1)) \frac{4}{w(V,V)} \leq \frac{4 + o(1)}{|\tilde{V}|} = (4 + o(1)) \frac{\tau_{\tilde{w}}(\tilde{x})}{|\tilde{V}|}
\end{aligned}$$

◻

▶ **Corollary 10.** *For $\epsilon > 0$, there is a randomized polynomial time algorithm for $(1+\epsilon)$-locally stable instances of Metric-MAXCUT.*

The drawback of this algorithm is that the exponent of the polynomial is large. We now sketch a simple $O(n^4)$ algorithm for $(3 + \epsilon)$-stable metric instances.

▶ Proposition 11. Let (L, R) be a γ-locally stable cut of an instance, w, of Metric-MAXCUT. Then, for every $x \in L, z \in R$, $w(x, z) \geq \left(\frac{\gamma^2 - 1}{\gamma}\right) \cdot \frac{w(x,R)}{\gamma \cdot |R| + |L|}$.

Proof. Using γ-local stability and the triangle inequality we obtain

$$\begin{aligned}
\frac{1}{\gamma} w(x, R) &\geq w(x, L) = \sum_{y \in L} w(x, y) \geq \sum_{y \in L} (w(z, y) - w(x, z)) \\
&= w(z, L) - |L| w(x, z) \geq \gamma w(z, R) - |L| w(x, z) = \gamma \sum_{y \in R} w(z, y) - |L| w(x, z) \\
&\geq \gamma \sum_{y \in R} (w(y, x) - w(z, x)) - |L| w(x, z) \\
&= \gamma w(x, R) - \gamma |R| w(x, z) - |L| w(x, z).
\end{aligned}$$

◻

▶ **Theorem 12.** *Let (X, w) be an instance of Metric-MAXCUT and let (L, R) be a $\gamma = (3+\epsilon)$-locally stable cut with $\epsilon > 0$. Then either L or R is a (metric) ball.*

Proof. W.l.o.g., $|L| \geq \frac{n}{2}$. We find some $x \in L$ such that $\forall z \in R$, $w(z,x) > \operatorname{diam}(L)$, thus proving our claim. Select some $x, y \in L$ with $w(x,y) = \operatorname{diam}(L)$. For every $z \in L$, we write $w(x,y) \leq w(x,z) + w(y,z)$. Summing over every $z \in L$, this yields $|L| \cdot w(x,y) \leq w(x,L) + w(y,L)$. W.l.o.g., assume that $w(x,L) \geq \frac{|L|}{2} \cdot w(x,y)$. By local stability,

$$w(x,y) \leq \frac{2}{|L|} w(x,L) \leq \frac{2 \cdot w(x,R)}{\gamma \cdot |L|} \tag{1}$$

By proposition 11, every $z \in R$ satisfies $w(x,z) \geq \left(\frac{\gamma^2-1}{\gamma}\right) \cdot \frac{w(x,R)}{\gamma \cdot |R| + |L|}$. Combined with equation (1), and the assumptions $\gamma > 3$ and $|L| \geq |R|$, we obtain that $w(x,z) > w(x,y)$ as claimed.

◻

By Theorem 12, the maximal cut of $(3+\epsilon)$-locally stable instances of Metric-MAXCUT can be found by simply considering all $O(n^2)$ balls.

▶ Note 13. Theorem 12 is tight in the following sense: There is a $(3-\epsilon)$-stable metric instance where neither side can be expressed as the union of few balls. Let $(X,w) = (L \coprod R, w)$ where $L = \{l_1, \ldots, l_{2n}\}$, $R = \{r_1, \ldots, r_{2n}\}$. For $1 \leq i \leq n$, $w(l_{2i-1}, l_{2i}) = w(r_{2i-1}, r_{2i}) = 2$ and for $1 \leq i \leq 2n$, $w(l_i, r_i) = 2$. All other distances within L and within R are 1, and between L and R are 3. It is not hard to see that w is a $(3-o(1))$-stable metric instance and neither side of the max cut can be decomposed into fewer than $2n$ balls.

4 Distinguished and Expanding Instances

Let $w : V \times V \to [0,\infty)$ be an instance of MAXCUT with a maximal cut $(S,\bar{S})$. We identify w with an $n \times n$ matrix W, where $W_{ij} = w(i,j)$. Define $w_{cut} : V \times V \to \mathbb{R}$ by $w_{cut}(u,v) = w(u,v)$ for $uv \in E(S,\bar{S})$ and $w_{cut}(u,v) = 0$ otherwise. Similarly, denote $w_{uncut} = w - w_{cut}$. Denote by W_{cut} and W_{uncut} the matrices corresponding to w_{cut} and w_{uncut} respectively. Finally, let D^{cut}, D^{uncut}, D and D' be the diagonal matrices defined by $D^{cut}_{ii} = \sum_j W^{cut}_{ij}$, $D^{uncut}_{ii} = \sum_j W^{uncut}_{ij}$, $D = D^{cut} + D^{uncut}$ and $D' = D^{cut} - D^{uncut}$.

▶ **Lemma 14.** *If w is γ-locally stable where $\gamma > \frac{2}{1-\sqrt{1-(h(w_{cut}))^2}}$, then $W + D'$ is a PSD matrix of rank $n-1$.*

As shown in [6] there is an efficient algorithm that correctly solves all instances that satisfy the conclusion of the Lemma. (Alternatively, by Theorem 20 such instances are GW-bipolar, and the GW-algorithm solves all such instances.) This proves the second part of Theorem 2.

Proof. First, we note that it is enough to prove that $D^{-\frac{1}{2}}(W+D')D^{-\frac{1}{2}}$ is a PSD matrix of rank $n-1$. Let $f : V \to \mathbb{R}$ be the vector defined by $f_i = \sqrt{D_{ii}}$ for $i \in S$ and $f_i = -\sqrt{D_{ii}}$ for $i \in \bar{S}$. Since $f^T D^{-\frac{1}{2}}(W+D')D^{-\frac{1}{2}} f = 0$, it is enough to show that $v^T D^{-\frac{1}{2}}(W+D')D^{-\frac{1}{2}} v > 0$ for every unit vector v that is orthogonal to f. Note that

$$D^{-\frac{1}{2}}(W+D')D^{-\frac{1}{2}} = D^{-\frac{1}{2}}(D^{cut} + W^{cut} - D^{uncut} + W^{uncut})D^{-\frac{1}{2}} \tag{2}$$

The matrix $D^{-\frac{1}{2}}(W^{cut} + D^{cut})D^{-\frac{1}{2}}$ is positive semi-definite and f is in its kernel (to see that, note that for $u \in \mathbb{R}^n$, $u^T(W^{cut} + D^{cut})u = \sum_{ij} W^{cut}_{ij}(u_i + u_j)^2)$. Therefore we have

$$v^T D^{-\frac{1}{2}}(W^{cut} + D^{cut})D^{-\frac{1}{2}} v \geq \lambda_2 \tag{3}$$

where $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ are the eigenvalues of $D^{-\frac{1}{2}}(W^{cut} + D^{cut})D^{-\frac{1}{2}}$. Moreover, $W^{uncut} + D^{uncut} \succeq 0 \Rightarrow 2D^{uncut} \succeq D^{uncut} - W^{uncut}$, where $A \succeq B$ means that the matrix $A - B$ is PSD. Thus, we have,

$$v^T D^{-\frac{1}{2}}(D^{uncut} - W^{uncut})D^{-\frac{1}{2}}v \leq 2 \cdot v^T D^{-\frac{1}{2}} D^{uncut} D^{-\frac{1}{2}} v \leq 2 \cdot \max_i \frac{D^{uncut}_{ii}}{D_{ii}} \leq \frac{2}{\gamma+1} \quad (4)$$

Combining equations (2), (3) and (4), it is enough to show that $\lambda_2 > \frac{2}{\gamma+1}$. However, since w_{cut} is bipartite, the matrices $D^{-\frac{1}{2}}(D^{cut} + W^{cut})D^{-\frac{1}{2}}$ and $D^{-\frac{1}{2}}(D^{cut} - W^{cut})D^{-\frac{1}{2}}$ have the same spectrum[3]. Also, $D^{-\frac{1}{2}}(D^{cut} - W^{cut})D^{-\frac{1}{2}}$ and $D^{-1}(D^{cut} - W^{cut})$ have the same spectrum[4] so it suffices to show that $\mu_2 > \frac{2}{\gamma+1}$, where μ_2 is the second smallest eigenvalue of $D^{-1}(D^{cut} - W^{cut})$. By the known relation between expansion and the second eigenvalue of the Laplacian (e.g., Theorem 2.2 in [11]), it follows that $\mu_2 \geq \min_i \frac{D^{cut}_{ii}}{D_{ii}} \cdot (1 - \sqrt{1 - h(w_{cut})^2}) \geq \frac{\gamma}{\gamma+1}(1 - \sqrt{1 - h(w_{cut})^2})$

□

Finally, to prove the first part of Theorem 2, it is enough to show that if w is α-distinguished then $h(w_{cut}) \geq \alpha$. Indeed, for $\emptyset \neq A \subset V$ we have

$$\tau_{w_{cut}}(A) = \xi_w(A) \geq \xi_w(A) - \iota_w(A) \geq \alpha \cdot \min\{\mu_w(A), \mu_w(\bar{A})\} \geq \alpha \cdot \min\{\mu_{w_{cut}}(A), \mu_{w_{cut}}(\bar{A})\}$$

5 The Spectral approach and the GW algorithm

We now consider a class of algorithms called *spectral algorithms* which have been used to give approximations or heuristics for MAXCUT (e.g. [7, 8, 12, 13]). We make various observations, including that the Goemans-Williamson (GW) approximation algorithm for MAXCUT is spectral. This study is motivated in part by the hope that such algorithms may do well on stable instances. We obtain a modest result (Corollary 19) in this direction.

In this section we view an instance of MAXCUT as an $n \times n$ matrix W and associate a cut $(S, \bar{S})$ with its *characteristic vector* δ_S which is 1 on S and -1 on $\bar{S}$. A vector $v \in \mathbb{R}^n$ is called a *generalized least eigenvector (GLEV)* of W if there is a diagonal matrix D such that v it is an eigenvector of $W + D$, corresponding to $(W + D)$'s least eigenvalue, λ. By letting $\Delta := D - \lambda I$ we see that v is a GLEV iff v is in the kernel of $W + \Delta$ for Δ diagonal with $W + D \succeq 0$. (As usual $A \succeq 0$ means that A is positive semi-definite). A vector $v \in \mathbb{R}^n$ *induces* the cut $(S, \bar{S})$ where $S = \{i : v_i > 0\}$. An algorithm for MAXCUT is called *spectral* if it always returns a cut that is induced by a GLEV.

Spectral algorithms arise naturally from the following formulation of MAXCUT:

$$\text{minimize} \quad v^T(W + D)v \quad \text{subject to} \quad v \in \{1, -1\}^n. \quad (5)$$

Here D can be any diagonal matrix. A natural relaxations to this problem is.

$$\text{minimize} \quad v^T(W + D)v \quad \text{subject to} \quad ||v|| = 1 \quad (6)$$

[3] This is essentially due to the fact that bipartite graphs have a symmetric spectrum and eigenvectors that come in pairs u and Pu, where $P : \mathbb{R}^n \to \mathbb{R}^n$ is the operator that reverses the sign of the coordinates corresponding to one side of the cut and leaves the other coordinates unchanged. This operator commutes with diagonal matrices and satisfies $WP = -PW$. Thus, v is an eigenvector of $D^{-\frac{1}{2}}(D^{cut} + W^{cut})D^{-\frac{1}{2}}$ with an eigenvalue λ iff Pv an eigenvector of $D^{-\frac{1}{2}}(D^{cut} + W^{cut})D^{-\frac{1}{2}}$ with an eigenvalue λ.

[4] Since v is an eigenvector of $D^{-\frac{1}{2}}(D^{cut} - W^{cut})D^{-\frac{1}{2}}$ with eigenvalue λ iff $D^{-\frac{1}{2}}v$ is an eigenvector of $D^{-1}(D^{cut} - W^{cut})$ with eigenvalue λ.

where $||\cdot||$ denotes the Euclidean norm. Solutions v of (6) are least eigenvectors of $W+D$. In view of (5), it is natural to consider the cut induced by such v.

The GW-Algorithm

In (5) we seek n vectors $v_1,\ldots,v_n$ in the 0 dimensional sphere $S^0=\{-1,1\}$ to minimize $\sum_{i,j} W_{i,j}\langle v_i,v_j\rangle$. In [12], an alternate relaxation is proposed (seemingly unrelated to (6)):

$$\text{minimize} \quad \sum_{i,j} W_{i,j}\langle v_i,v_j\rangle \quad \text{subject to} \quad v_i\in S^{n-1} \tag{7}$$

The GW-algorithm[12] returns the cut induced by vector u defined by $u_i=\langle v,v_i\rangle$ where $v\in S^{n-1}$ is sampled uniformly. This yields the approximation ratio 0.879 for MAXCUT.

To solve (7) the GW algorithm finds first a solution P to the problem

$$\text{minimize} \quad P\circ W \quad \text{subject to} \quad P\succeq 0, \quad P_{ii}=1,\ \forall i\in[n]. \tag{8}$$

Where $P\circ W := \sum_{1\le i,j\le n} P_{ij}\cdot W_{ij}$. Since $P\succeq 0$ it is possible to find next vectors $v_1,\ldots,v_n$ such that $P_{ij}=\langle v_i,v_j\rangle$. The dual to (8) is (see [12])

$$\text{maximize} \quad \sum_{i=1}^{n} D_{ii} \quad \text{subject to} \quad W-D\succeq 0, \quad D \text{ is diagonal.} \tag{9}$$

As observed in [12], by SDP duality the optima of (8) and (9) coincide. Denote by $\mathcal{P}(W)$ and $\mathcal{D}(W)$ the set of optimal solutions to (8) and (9) respectively. Denote also $\mathcal{P}=\{P\in M_n(\mathbb{R}) : P\succeq 0 \text{ and } \forall i,\ P_{ii}=1\}$, $\mathcal{D}=\{D\in M_n(\mathbb{R}) : D \text{ is diagonal}\}$. We say that W is *GW-bipolar* if there exists a solution to (9) that also solves the binary problem (6) (i.e., it is contained in a copy of S^0 embedded in S^{n-1}). Equivalently, W is GW-bipolar if $\mathcal{P}$(W) contains a matrix of the form $v\cdot v^T$ for some $v\in\{-1,1\}^n$. Finally, we shall say that W is *strongly GW-bipolar* if *every* solution to (7) is also a solution of (6). The maximal cut of such an instance can be immediately read of the output of the GW-algorithm.

In the rest of this section, we prove that an instance can be correctly solved by *some* spectral algorithm iff it is has a certain perturbation that is GW-bipolar. We also give a primal-dual characterization of the set of solutions to the GW-relaxation, which allows us to conclude that the GW-algorithm is a spectral algorithm.

▶ **Theorem 15.** *Let W be a MAXCUT instance and v be a GLEV of W. Then the cut S induced by v is a maximum cut if and only if the matrix W' with entries $W_{i,j}=|v_i||v_j|W_{i,j}$ is GW-bipolar. In particular, W has a ± 1-GLEV iff W is GW-bipolar.*

Proof. As noted before, v is a GLEV if and only if v is in the kernel of $W+D$ for some diagonal matrix D for which $W+D\succeq 0$. Thus, v is a GLEV of W if and only if the optimum of the following SDP is 0.

$$\underset{P}{\text{minimize}} \quad v^T(W+D)v \quad \text{subject to} \quad W+D\succeq 0 \quad D \text{ is diagonal.} \tag{10}$$

The dual program of (10) is:

$$\underset{P}{\text{maximize}} \quad v^TWv - P\circ W \quad \text{subject to} \quad P_{ii}=v_i^2, \quad P\succeq 0. \tag{11}$$

Since (10) has a positive definite solution, strong duality holds. Thus, v is a GLEV iff the optimum of (11) is 0. We now show this latter condition is equivalent to W' being

GW-bipolar. Note that the mapping $P' \mapsto P$ where $P_{ij} = |v_i| \cdot |v_j| \cdot P'_{ij}$ maps the feasible solutions to the primal GW-relaxation (8) for W' onto the feasible solution to (11). Moreover, $P \circ W = P' \circ W'$. Thus, the optimum of (11) is zero iff the optimum of the primal GW relaxation of W' is $v^T W v = \delta_S^T W' \delta_S$. Consequently, the optimum of (11) is 0 iff the optimum of (8) is attained by a ± 1 vector, making W' GW-bipolar.

Next we give a primal-dual characterization of $\mathcal{D}(W)$ and $\mathcal{P}(W)$.

▶ **Theorem 16.** *Let W be a non-negative symmetric matrix with 0-diagonal. Then (1) $\mathcal{D}(W)$ is a singleton*[5]*, and (2) $\mathcal{P}(W) = \{P \in \mathcal{P} : P(W - \mathcal{D}(W)) = 0\}$.*

▶ **Lemma 17.** *For every $D^0 \in \mathcal{D}(W)$, $P^0 \in \mathcal{P}(W)$, $\mathcal{P}(W) = \{P \in \mathcal{P} : P(W - D^0) = 0\}$ and $\mathcal{D}(W) = \{D \in \mathcal{D} : (W - D) \succeq 0,\ P^0(W - D) = 0\}$.*

Proof. Let $D^0 \in \mathcal{D}(W)$, $P \in \mathcal{P}$. By strong duality,

$$P \in \mathcal{P}(W) \Leftrightarrow W \circ P = \sum_{i=1}^{n} D_i^0 \Leftrightarrow W \circ P = D^0 \circ P$$

Since $W - D^0$ and P are PSDs, $P \circ (W - D^0) = 0 \Leftrightarrow P(W - D^0) = 0$. Thus, $\mathcal{P}(W) = \{P \in \mathcal{P} : P(W - D^0) = 0\}$. Let $P^0 \in \mathcal{P}(W)$, and suppose $D \in \mathcal{D}$ satisfies $W - D \succeq 0$. Then

$$D \in \mathcal{D}(W) \Leftrightarrow W \circ P^0 = \sum_{i=1}^{n} D_i \Leftrightarrow W \circ P^0 = D \circ P^0$$

Thus $\mathcal{D}(W) = \{D \in \mathcal{D} : (W - D) \succeq 0,\ P^0(W - D) = 0\}$.

◻

Proof. (of Theorem 16) (2) follows from (1) and Lemma 17, so it remains to prove (1). Fix some $P^0 \in \mathcal{P}(W)$ and let $D \in \mathcal{D}(W)$. By considering the (j, j) entry of $P^0(W - D) = 0$, we have $D_{jj} = \sum_{i=1}^{n} P^0_{ji} W_{ij}$, which determines D uniquely.

▶ **Corollary 18.** *GW is a spectral algorithm.*

Proof. Let P be an optimum of the GW-relaxation and let $v_1, \ldots, v_n \in S^{n-1}$ be vectors such that $P_{ij} = \langle v_i, v_j \rangle$. Let V be the matrix with columns $v_1, \ldots, v_n$. Let $v \in S^{n-1}$ be the vector sampled by the algorithm and let $\sum_{j=1}^{n} \alpha_j v_j$ be its orthogonal projection on $span\{v_1, \ldots, v_n\}$. The cut returned by the algorithm is the one induced by the vector $u = v^T V = \sum_j \alpha_j P_{ij}$, and so by Theorem 16 it is in the kernel of the PSD matrix $W - \mathcal{D}(W)$.

▶ **Corollary 19.** *The GW algorithm correctly solves $\Omega(n^3)$-stable instances.*

Proof. In [6] it is shown that if u is a GLEV of a γ-stable instance W with $\gamma \geq \frac{\max_{(i,j) \in E} |u_i u_j|}{\min_{(i,j) \in E} |u_i u_j|}$ then u induces the optimal cut. Let u be defined as in the proof of Corollary 18. As shown, u is a GLEV. By an easy probabilistic argument, w.h.p., $\forall j, n^{-1.5} \leq |u_j| \leq 1$.

▶ **Theorem 20.** *Let W be an MAXCUT instance with max cut S. Let $v = \delta_S$ and let D be the diagonal matrix defined by $D_{ii} = -v_i \sum_j W_{ij} v_j$. The following conditions are equivalent.*

1. *W is GW-bipolar.*
2. *δ_S is a GLEV of W.*
3. *$W + D \succeq 0$*
4. *The optimum of the dual of the GW-relaxation is attained at $-D$.*

[5] Henceforth we usually do not distinguish between $\mathcal{D}(W)$ and the single matrix that it contains.

Proof. By Theorem 15, (1) is equivalent to (2). It not hard to see that (3) implies that δ_S is in the kernel of $W + D$, and hence (2) . (4) clearly implies (3). Finally, suppose that (1) holds. Let D' be the solution of problem (9). Since W is GW-bipolar, $\delta_S \cdot \delta_S^T$ is an optimal primal solution. By Lemma 17, $\delta_S \in ker(W - D')$ and $D' = -D$. Hence (4) holds.

6 Some open problems

- We have shown that $O(\sqrt{n})$-stability suffices to solve MAXCUT optimally. On the other hand, we can't rule out the possibility that for any $\gamma^* > 1$, every γ^*-stable instances can be solved in polynomial time. In particular, we don't know any hardness reductions.
- What is the best possible dependency of γ on α in Theorem 2?
- Regarding Corollary 10, is there a simple practical algorithm to handle 2-locally stable metric instances?
- More broadly, analyse other problems with respect to the stability approach. (See [5] for recent work in this direction.)

References

1 M. Ackerman and S. Ben David. Which data sets are clusterable? a theoretical study of clusterability. *NIPS (2009)*.

2 S. Arora, D. Karger, and M. Karpinski Approximation schemes for dense instances of NP-hard problems. *STOC (1995)*, pages 284-294.

3 P. Awasthi, A. Blum, and O. Sheffet. Center-based clustering under perturbation stability. *Information Processing Letters*, volume 112, pages 49-54, 2011.

4 M.F. Balcan, A. Blum, and S. Vempala. A discriminative framework for clustering via similarity functions. *STOC (2008)*, pages 671-680.

5 M. F. Balcan and Y. Liang. Clustering under Perturbation Resilience. ICALP (1) 2012: 63-74 (see also `http://arxiv.org/pdf/1112.0826v3.pdf`).

6 Y. Bilu and N. Linial Are Stable instances Easy? *Innovations in Computer Science (Beijing, China, 2010)*, pages 332-341.

7 R. Boppana. Eigenvalues and graph bisection: An average case analysis. *FOCS (1987)*, pages 280-285.

8 C. Delorme and S. Poljak. Laplacian eigenvalues and the maximum cut problem. *Math. Programming*, 62(3, Ser. A):557-574, 1993.

9 A. Daniely, N. Linial, and M. Saks. Clustering is difficult only when it does not matter. *To appear* (see `http://www.cs.huji.ac.il/~nati/PAPERS/cluster_ez.pdf`), 2012.

10 U. Feige and J. Kilian. Heuristics for semirandom graph problems. *J. Comput. System Sci.*, 63(4):639- 671, 2001. Special issue on FOCS (1998).

11 S. Friedland and R. Nabban. On Cheeger-type inequalities for weighted graphs. *Journal of Graph Theory*, Volume 41, Issue 1, pages 1-17, 2002.

12 M. X. Geomans and D. P. Williamson. Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming. *Journal of the ACM*, Volume 42, pages 1115-1145, 1995.

13 F. McSherry. Spectral partitioning of random graphs. *FOCS(2001)*, pages 529-537.

14 D. Spielman and S. H. Teng. Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. J. ACM 51(3): 385-463 (2004)

15 W. Fernandez de la Vega and Claire Kenyon. A Randomized Approximation Scheme for Metric MAX-CUT. *FOCS (1998)*, pages 468-471.

First Fit bin packing: A tight analysis

György Dósa[1] and Jiří Sgall[2]

1 Department of Mathematics, University of Pannonia
Veszprém, Hungary
dosagy@almos.vein.hu

2 Computer Science Institute of Charles University
Faculty of Mathematics and Physics
Malostranské nám. 25, CZ-11800 Praha 1, Czech Republic
sgall@iuuk.mff.cuni.cz

Abstract

In the bin packing problem we are given an instance consisting of a sequence of items with sizes between 0 and 1. The objective is to pack these items into the smallest possible number of bins of unit size. FIRSTFIT algorithm packs each item into the first bin where it fits, possibly opening a new bin if the item cannot fit into any currently open bin. In early seventies it was shown that the *asymptotic* approximation ratio of FIRSTFIT bin packing is equal to 1.7.

We prove that also the *absolute* approximation ratio for FIRSTFIT bin packing is exactly 1.7. This means that if the optimum needs OPT bins, FIRSTFIT always uses at most $\lfloor 1.7 \cdot \mathrm{OPT} \rfloor$ bins.

Furthermore we show *matching lower bounds* for a majority of values of OPT, i.e., we give instances on which FIRSTFIT uses exactly $\lfloor 1.7 \cdot \mathrm{OPT} \rfloor$ bins.

Such matching upper and lower bounds were previously known only for finitely many small values of OPT. The previous published bound on the absolute approximation ratio of FIRSTFIT was $12/7 \approx 1.7143$. Recently a bound of $101/59 \approx 1.7119$ was claimed.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Approximation algorithms, online algorithms, bin packing, First Fit

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.538

1 Introduction

Bin packing is a classical combinatorial optimization problem in which we are given an instance consisting of a sequence of items with rational sizes between 0 and 1, and the goal is to pack these items into the smallest possible number of bins of unit size. FIRSTFIT algorithm packs each item into the first bin where it fits, possibly opening a new bin if the item does not fit into any currently open bin.

Johnson's thesis [8] on bin packing together with Graham's work on scheduling [6, 7] belong to the early influential works that started and formed the whole area of approximation algorithms. The proof that the asymptotic approximation ratio of FIRSTFIT bin packing is 1.7 given by Ullman [13] and subsequent works by Garey et al. and Johnson et al. [5, 9] were among these first results on approximation algorithms.

In this paper, we prove that also the *absolute* approximation ratio for FIRSTFIT bin packing is exactly 1.7. This means that if the optimum needs OPT bins, FIRSTFIT always uses at most $\lfloor 1.7 \cdot \mathrm{OPT} \rfloor$ bins. Thus we settle this open problem after almost 40 years.

Furthermore we show matching lower bounds for a majority of values of OPT, i.e., we give instances on which FIRSTFIT uses exactly $\lfloor 1.7 \cdot \mathrm{OPT} \rfloor$ bins. More precisely, we give these lower bounds for all values of Opt, except when Opt mod 10 equals to 0 or 3; for these two remaining cases we show a lower bound of $\lfloor 1.7 \cdot \text{Opt} \rfloor - 1$.

Such matching upper and lower bounds were previously known only for finitely many small values of Opt. Thus our results not only give the exact worst case for most values of Opt (8 out of 10 residue classes), but actually even give the first infinite sequence of values of Opt for which the exact worst-case performance of FirstFit is known.

1.1 History and related work

The upper bound on FirstFit was first shown by Ullman in 1971 [13]; he proved that for any instance, $\text{FF} \leq 1.7 \cdot \text{Opt} + 3$, where FF and Opt denote the number of bins used by FirstFit and the optimum, respectively. Still in seventies, the additive term was improved first in [5] to 2 and then in [4] to $\text{FF} \leq \lceil 1.7 \cdot \text{Opt} \rceil$; due to integrality of FF and Opt this is equivalent to $\text{FF} \leq 1.7 \cdot \text{Opt} + 0.9$. Recently the additive term of the asymptotic bound was improved to $\text{FF} \leq 1.7 \cdot \text{Opt} + 0.7$ in [15].

The absolute approximation ratio of FirstFit got some attention recently. A significant step towards settling the question of the absolute approximation ratio was the upper bound of 1.75 by Simchi-Levy [12]. This was improved independently by Xia and Tan [15] and Boyar et al. [1] to $12/7 \approx 1.7143$ and recently Németh claimed an upper bound of $101/59 \approx 1.7119$ [10].

For the lower bound, the early works give examples both for the asymptotic and absolute ratios. The example for the asymptotic bound gives $\text{FF} = 1.7 \cdot \text{Opt}$ whenever $\text{Opt} = 10k+1$, thus it shows that the asymptotic upper bound of 1.7 is tight, see [13, 5, 9]. For the absolute ratio, an example is given with $\text{FF} = 17$ and $\text{Opt} = 10$, which shows that the absolute approximation ratio cannot be better than 1.7 [5, 9]. (Also an example with $\text{FF} = 34$ and $\text{Opt} = 20$ is claimed, but it seems that this example has never been published.)

Johnson [8, 9] has also analyzed the First Fit Decreasing algorithm, which behaves like FirstFit but receives the items on the input sorted from the largest one to the smallest, and proved that the asymptotic approximation ratio is equal to 11/9. Johnson's bound had an additive constant of 4; this was improved several times and finally it was shown that the additive constant is exactly 2/3 [3]. That is, $\frac{11}{9}\text{Opt} + \frac{2}{3}$ bins are sufficient for First Fit Decreasing, but this number of bins is actually also necessary for some instances for infinitely many values of Opt. Thus for First Fit Decreasing, the asymptotic and absolute approximation ratios are not equal. In fact, the results of [3] give the exact value of the worst case for every value of Opt and show that the worst case absolute ratio is equal to 4/3, attained for $\text{Opt} = 6$ when 8 bins may be needed for First Fit Decreasing. In light of this result, it is rather surprising that for FirstFit the asymptotic and absolute approximation ratios *are* equal and no additive term is needed.

We have mentioned only directly relevant previous work. Of course, there is much more work on bin packing, in particular there exist approximation schemes for this problem, as well as many other algorithms. We refer to the survey [2] or to the recent excellent book [14].

1.2 Main ideas of our results

Once the asymptotic bound with a small additive constant is shown, a natural approach to improve absolute upper bounds is to study fixed small values of Opt and to exclude the possibility of a higher absolute ratio for them. Indeed, solving a few such cases necessarily improves upper bounds on the absolute ratio—but cannot give a tight result. Of course, this is still far from trivial: Even for a fixed Opt, each such problem seems to lead to a new and more extensive case analysis. Instead of joining this race of incremental results, we choose a different approach to attack arbitrarily large values of Opt directly.

The first important step is a combination of amortization and weight function analysis. To illustrate our technique, we now present a new short proof of the asymptotic ratio 1.7 for FirstFit. It uses the same weight function as the traditional analysis of FirstFit. To use amortization, we split the weight of each item into two parts. Identifying an item a with its size, the weight of a is its scaled size $\frac{6}{5}a$ plus the bonus $v(a)$ defined as

$$v(a) = \begin{cases} 0 & \text{if } a \le \frac{1}{6}\,, \\ \frac{3}{5}(a - \frac{1}{6}) & \text{if } a \in \left(\frac{1}{6}, \frac{1}{3}\right)\,, \\ 0.1 & \text{if } a \in \left[\frac{1}{3}, \frac{1}{2}\right]\,, \\ 0.4 & \text{if } a > \frac{1}{2}\,. \end{cases}$$

Note that there is a discontinuity only at $a = 1/2$. For a set of items B, $v(B) = \sum_{a \in B} v(a)$ denotes the total bonus and $s(B) = \sum_{a \in B} a$ the total size.

It is easy to observe that the weight of any bin B, i.e., of any set with $s(B) \le 1$, is at most 1.7: The scaled size of B is at most 1.2, so we only need to check that $v(B) \le 0.5$. If B contains no item larger than 1/2, there are at most 5 items with non-zero $v(a)$ and $v(a) \le 0.1$ for each of them. Otherwise the large item has bonus 0.4; there are at most two other items with non-zero bonus and it is easy to check that their total bonus is at most 0.1.

Consider an instance I. The previous bound implies that the weight of the whole instance $\frac{6}{5}s(I) + v(I)$ is at most $1.7 \cdot \text{Opt}$.

The key part is to show that, on average, the weight of each FirstFit bin is at least 1. Lemma 1.1 implies that for almost all bins with two or more items, the first part of its weight plus the second part of the weight of the *following* such bin is at least 1.

▶ **Lemma 1.1.** *Let B, C be two bins in the* FirstFit *packing such that $s(B) \ge 2/3$, C contains at least two items, and B is opened before C. Then $\frac{6}{5}s(B) + v(C) \ge 1$.*

Proof. Since C is after B in the FirstFit packing, C contains two items c and c' that do not fit in B, i.e., $c, c' > 1 - s(B)$. If $s(B) \ge 5/6$ then the lemma follows trivially without considering $v(C)$. In the remaining case, let $x \in (0, \frac{1}{6}]$ be such that $s(C) = \frac{5}{6} - x$. Thus $c, c' > \frac{1}{6} + x$ and $v(c), v(c') > \frac{3}{5}x$. We get $\frac{6}{5}s(B) + v(c) + v(c') > \frac{6}{5}(\frac{5}{6} - x) + \frac{3}{5}x + \frac{3}{5}x = 1$. ◀

Consider any FF-bin B with a single item. If $s(B) > 1/2$, then $b(B) = 0.4$ and $\frac{6}{5}s(B) + v(B) > 1$. Furthermore, at most one FF-bin has $s(B) \le 1/2$, by the definition of FirstFit.

Now consider FF-bins with two or more items. Similarly, at most one of them has size less than 2/3: If we have one such bin, any item in any later bin is larger than 1/3 and thus any later bin with two items is larger than 2/3. Now we use Lemma 1.1 for every FirstFit bin B with two or more items and $s(B) \ge 2/3$ (except for the last such bin); the bin C is chosen as the next bin with the same properties.

Summing the bounds for bins with a single item plus the bounds from Lemma 1.1 for bins with two or more items (note that each bin is used at most once as B and at most once as C), we obtain that $\frac{6}{5}s(I) + v(I) \ge \text{FF} - 3$. The additive constant 3 comes from the fact that we did not bound the weight of at most three FF-bins: (i) one bin with a single item and $s(B) \le 1/2$, (ii) one bin with two or more items and $s(B) < 2/3$, and (iii) the last bin with two or more items. Combining this with the previous bound on the total weight, we obtain $FF - 3 \le \frac{6}{5}s(I) + v(I) \le 1.7 \cdot \text{Opt}$ and the asymptotic bound follows.

By itself, this simplified analysis can decrease the additive constant to 0.6 (after examining the remaining three bins in the FF packing) but cannot remove it completely. To obtain the tight bound, we need to analyze different types of bins in the FF packing quite carefully. In the typical worst case, FF packing starts by bins with five or more items of size around 1/6 or smaller, followed by Opt/2 bins with two items slightly larger than 1/3, and ends by Opt bins with a single item slightly larger than 1/2. We analyze these three types of bins separately. To handle various possible situations we slightly modify the weight function (see Definition 3.5) and the amortization lemma (see Lemma 3.8).

The most delicate part of the proof analyzes the FirstFit bins containing three or four items—or rather shows that they cannot play an important role in the worst case; here it is important that the amortization uses the bonus of only two items and thus the bins with three or four items are "wasteful". In the final steps of the proof, the parity of the items of size around 1/3 comes into play: Typically they come in pairs, as described above, but for odd values of Opt one of them is missing (or is in a FirstFit bin of 3 or more items), and this allows us to remove the last 0.1 of the additive term. Our analysis sketched above still leaves a few values of Opt that need to be analyzed separately. However, with our framework of the general proof, even this is relatively simple compared to the previous proofs in this area. The upper bound proof is presented in Section 3.

Similar amortization was used in [11] to analyze the Best Fit bin packing algorithm. There the situation is more complicated, as the notion of the "following bin" is not clear, in fact a careful choice is needed. Currently we are not able to fully extend our bounds to Best Fit. The bottleneck seems to be the analysis of the bins with three and four items.

For the lower bounds we modify the instance from [5, 9]. The original construction is quite intricate. Fortunately—and perhaps also surprisingly—it is sufficient to carefully analyze the high-level structure of the instance, add to it a few new jobs, and carefully position them in the input instance. See Section 4 for the details.

2 Notations

Let us fix an instance I with items $a_1, \dots a_n$ and denote the number of bins in the FirstFit and optimal solutions by FF and Opt, respectively. We will often identify an item and its size. For a set of items A, let $s(A) = \sum_{a\in A} a$, i.e., the total size of items in A and also for a set of bins $\mathcal{A}$, let $s(\mathcal{A}) = \sum_{A\in\mathcal{A}} s(A)$. Furthermore, let $S = s(I)$ be the total size of all items of I. Obviously $S \leq \text{Opt}$.

The bins in the FF packing are ordered by the time they are opened (i.e., the first item is packed into them). We refer to this order when we say that one bin is before or after another one, or when we speak about the first or last bin.

A bin is called a *k-bin* or *k^+-bin*, if it contains exactly k items or at least k items, respectively, for an integer k. An item is called k-item if FF packs it into a k-bin.

We classify the the FF bins into three groups. If a 2^+-bin B satisfies $s(B) \geq 5/6$, it is a *big* bin; $\mathcal{B}$ denotes the set of all big bins and β their number. Any other 2^+-bin C is a *common* bin; $\mathcal{C}$ denotes the set of all common bins and γ their number. Finally, any 1-bin D is a *dedicated* bin; $\mathcal{D}$ denotes the set of all dedicated bins and δ their number. The items in big, common, and dedicated bins are called B-items, C-items, and D-items, respectively. Finally, let C2-items be the items in common 2-bins. The common and dedicated bins are typically denoted by C and D, and C-items and D-items by c and d (with indices and other decorations). We use B for generic bins (typically big or common) and b for items that may be in big or common bins. If there exists a D-item with size at most 1/2, denote it d_0; otherwise d_0 is undefined. We shall see in Lemma 3.2(i) that there is at most one such item.

3 The upper bound

3.1 Preliminaries

We state a few basic properties of FF packings. Assumption 3.1 as well as all parts of Lemma 3.2 are known and easy facts used explicitly or implicitly in previous works on FirstFit including [12, 15, 1].

▶ Assumption 3.1. We assume, without loss of generality, that no two items a_i and a_j are packed into the same bin both in FF and Opt solutions.

This is w.l.o.g., since any such items may be replaced by a single item of size $a_i + a_j$ that arrives at the time of arrival of the first of the original items. It is easy to see that both FF and Opt solutions are unchanged (except for this replacement).

▶ **Lemma 3.2.** *In the* FF *packing the following holds:*

(i) *The sum of sizes of any two* FF*-bins is greater than 1. The total size of any $k \geq 2$* FF*-bins is greater than $k/2$.*
(ii) *The D-items are packed into different optimal bins. Thus $\delta \leq$* Opt*.*
(iii) *There is at most one common bin C_0 with $s(C_0) \leq 2/3$. Furthermore, if $s(C_0) = 2/3 - 2x$ for $x \geq 0$ then for any other 2^+-bin (i.e., any other common or big bin) B we have $s(B) > 2/3 + x$; in addition, if B is opened after C_0, then $s(B) > 2/3 + 4x$.*
(iv) *If $k \geq 3$, then the total size of k arbitrary 2^+-bins is greater than $\frac{2}{3}k$.*
(v) *Suppose that $k \geq 1$, we have $k+1$* FF*-bins B_1, B_2, ..., B_k, B, in this order, and such that B is a k^+-bin. Then the sum of the sizes of these $k+1$ bins is greater than k.*

Proof. (i): The first item in any FF-bin does not fit in any previous bin, thus the sum of their sizes is greater than 1 already at the time when the second bin is opened. For k bins, order the bins cyclically and sum the inequalities $s(B_i) + s(B_j) > 1$ for pairs of adjacent bins.

(ii): Follows from (i), as the size of each D-item equals the size of its dedicated FF-bin.

(iii): If B is after C_0, then it contains only items of size larger than $1-s(C_0) = 1/3+2x$; since it contains two items, $s(B) > 2/3 + 4x$ follows. If B is before C_0, then notice that C_0 contains an item of size at most $s(C_0)/2 = 1/3 - x$; This item was not packed into B, thus it follows that $s(B) > 2/3 + x$.

(iv): Follows immediately from (iii).

(v): Let x be the minimum of $s(B_i)$, $i = 1, \ldots, k$. Then by the FF-rule, any item in bin B is larger than $1-x$. Since there are at least k items in bin B, we have $s(B)+\sum_{i=1}^{k} s(B_i) > k(1 - x) + kx = k$. ◀

Now we assume that the instance violates the absolute ratio 1.7 and derive some easy consequences that exclude some degenerate cases. The first claim, Opt ≥ 7, follows from [1, 15]; we include its proof for completeness. Note that the values of $1.7 \cdot$ Opt are multiples of 0.1 and FF is an integer, thus FF $> 1.7 \cdot$ Opt implies FF $\geq 1.7 \cdot$ Opt $+ 0.1$. Typically we derive a contradiction with the fact $S \leq$ Opt stated above.

▶ **Lemma 3.3.** *If* FF $> 1.7 \cdot$ Opt *then the following holds:*

(i) Opt ≥ 7.
(ii) *No common bin C has size $s(C) \leq 1/2$.*
(iii) *The number of dedicated bins is bounded by $\delta \geq 3$.*
(iv) *The number of common bins is bounded by $\gamma \geq$* Opt*$/2+1 > 4$. If* FF $\geq 1.7 \cdot$Opt$+\tau/10$ *for some integer $\tau \geq 1$ then $\gamma >$ (*Opt *$+ \tau)/2$.*

Proof. (i): If $\textsc{Opt} \in \{3,4,5,6\}$ and $\text{FF} > 1.7 \cdot \textsc{Opt}$ then we can verify that both $\text{FF} \geq 2 \cdot \textsc{Opt} - 1$ and $\text{FF} \geq \textsc{Opt} + 3$. Using Lemma 3.2(ii), the number of 2^+-bins is $\beta + \gamma = \text{FF} - \delta \geq \text{FF} - \textsc{Opt} \geq 3$. Thus we can use Lemma 3.2(v) and obtain a contradiction:

$$S > \frac{2}{3}(\beta+\gamma) + \frac{1}{2}\delta = \frac{1}{6}(\beta+\gamma) + \frac{1}{2}\text{FF} \geq \frac{1}{6}\cdot 3 + \frac{1}{2}(2\cdot\textsc{Opt} - 1) = \textsc{Opt}\,.$$

If $\textsc{Opt} = 2$ and $\text{FF} > 1.7 \cdot \textsc{Opt}$ then $\text{FF} \geq 4$, and by Lemma 3.2(i) we have $S > 4 \cdot \frac{1}{2} = \textsc{Opt}$, a contradiction. For $\textsc{Opt} = 1$, FirstFit is trivially optimal.

(ii): Suppose that $s(C_0) \leq 1/2$ for a contradiction. Lemma 3.2(iii) implies that any big or common bin C before C_0 has $s(C) \geq 3/4$. Furthermore, any bin after C_0 is a D-bin (as it can contain only items larger than $1/2$) and by Lemma 3.2(i), the total size of C_0 and all D-bins is at least $(\delta+1)/2$. Thus we can obtain a contradiction by using $\textsc{Opt} \geq 7$ from (i) and $\delta \leq \textsc{Opt}$ from Lemma 3.2(ii) as follows:

$$\begin{aligned} S &> \frac{3}{4}(\beta+\gamma-1) + \frac{1}{2}(\delta+1) = \frac{3}{4}\text{FF} - \frac{1}{4}(\delta+1) \\ &\geq \frac{3}{4}\left(\frac{17}{10}\textsc{Opt} + \frac{1}{10}\right) - \frac{1}{4}(\textsc{Opt}+1) = \frac{41}{40}\textsc{Opt} - \frac{7}{40} \geq \textsc{Opt}\,. \end{aligned}$$

(iii): Suppose for a contradiction that $\delta \leq 2$. Then each FF-bin contains at least two items, except for at most two dedicated FF-bins. Since $\textsc{Opt} \geq 7$ from (i), we can apply Lemma 3.2(iv) for the $\text{FF} - 2 \geq 3$ of 2^+-bins and Lemma 3.2(i) for the remaining two bins, and thus we obtain a contradiction as follows:

$$S > \frac{2}{3}(\text{FF}-2) + 1 \geq \frac{2}{3}\left(\frac{17}{10}\textsc{Opt} + \frac{1}{10} - 2\right) + 1 = \frac{17}{15}\textsc{Opt} - \frac{4}{15} > \textsc{Opt}\,.$$

(iv): To obtain the first bound from the second one, use $\tau = 1$ and the integrality of $\textsc{Opt}$. Now suppose for a contradiction that $\gamma \leq (\textsc{Opt}+\tau)/2$. If $\gamma \geq 3$, then we use Lemma 3.2(v) for $\mathcal{C}$, Lemma 3.2(i) for $\mathcal{D}$, and the fact that the remaining bins are big, and we obtain

$$\begin{aligned} S &> \frac{5}{6}(\text{FF}-\gamma-\delta) + \frac{2}{3}\gamma + \frac{1}{2}\delta = \frac{5}{6}\text{FF} - \frac{1}{6}\gamma - \frac{1}{3}\delta \\ &\geq \frac{5}{6}\left(\frac{17}{10}\textsc{Opt} + \frac{\tau}{10}\right) - \frac{\textsc{Opt}+\tau}{12} - \frac{1}{3}\textsc{Opt} = \textsc{Opt}\,, \end{aligned}$$

a contradiction. If $\gamma \leq 2$ then

$$\begin{aligned} S &> \frac{5}{6}(\text{FF}-\delta-2) + \frac{1}{2}(\delta+2) = \frac{5}{6}\text{FF} - \frac{1}{3}(\delta+2) \\ &\geq \frac{5}{6}\left(\frac{17}{10}\textsc{Opt} + \frac{\tau}{10}\right) - \frac{1}{3}\textsc{Opt} - \frac{2}{3} \geq \textsc{Opt} + \frac{\textsc{Opt}+1}{12} - \frac{2}{3} \geq \textsc{Opt}\,, \end{aligned}$$

using (i) in the last step, and we obtain a contradiction as well. ◀

3.2 The weight function and the main lemma

Now we introduce the main ingredients of our analysis: the modified weight function and the main lemma that is used for the amortized analysis of the weight of FF bins. As in the simple proof in the introduction and previous bin packing literature, our ultimate goal is to prove that each $\textsc{Opt}$-bin has weight at most 1.7 and each FF-bin has an amortized (average) weight at least 1.

It is convenient to describe the weight of each item a in two parts. The first part, $\overline{\overline{w}}(a)$, is called the regular (part of the) weight, and it is proportional to the size of a; it is the same as in the simple proof. The other part, $\overline{w}(a)$ is called the bonus and it is modified so that it depends both on the size of a and the type of FF-bin where a is packed. B-items have no bonus. C-items have bonus equal to 0 for items of size at most 1/6, equal to 0.1 for items of size at least 1/3, and linearly interpolated between these values. D-items have bonus 0.4 if they have size at least 1/2 and slightly smaller if they have smaller size (this concerns only the single item d_0).

Compared to the simple proof and the previous literature, we make several modifications to the weight function. The first two are mostly a matter of convenience and simplification of the case analysis in the proof. First, we move the bonus from the items larger than 1/2 to the D-items. Mostly these are actually the same items, except for d_0. As we shall see later, in the tight cases, each OPT-bin contains a D-item and this change allows a more uniform analysis. Second, we decrease some of the weights that we do not use in the proof, namely we do not put any bonus on B-items and decrease the bonus on d_0 (this is necessary to guarantee that its OPT-bin has weight at most 1.7; however, in tight cases d_0 is very close to 1/2). The third change is essential in our last step of the proof where we remove the remaining additive constant of 0.1. We define a set of at most two exceptional C-items whose bonus is decreased to 0. Since they are in 3^+-bins in the FF packing, this does not change the analysis of the FF packing significantly. On the other hand, the exceptional items are chosen so that, if they exist, then one OPT-bin is guaranteed to have weight at most 1.6, which is exactly the necessary improvement.

Formally we define the exceptional items as follows:

▶ **Definition 3.4.** If $\text{OPT} \equiv 7 \pmod{10}$ and there exists an OPT-bin E that contains no C2-item, then fix any such bin E for the rest of the proof. Otherwise E is undefined. If E contains at most two C-items with size larger than 1/6, denote the set of these items E'. Otherwise (if there are three or more C-items in E or no E exists) put $E' = \emptyset$.

Let us call E the *exceptional* bin and the items in E' the *exceptional* items.

Note that there is at most one exceptional item in each FF-bin by Assumption 3.1. Later we shall show that in a potential counterexample with $\text{FF} = 1.7 \cdot \text{OPT} + 0.1$ the bin E exists.

▶ **Definition 3.5.** The weight function is defined as follows:

For a B-item b we define $\quad \overline{w}(b) = 0.$

For a C-item c we define
$$\overline{w}(c) = \begin{cases} 0 & \text{if } c \le \frac{1}{6} \text{ or } c \in E', \\ \frac{3}{5}(c - \frac{1}{6}) & \text{if } c \in \left[\frac{1}{6}, \frac{1}{3}\right] \text{ and } c \notin E', \\ 0.1 & \text{if } c \ge \frac{1}{3} \text{ and } c \notin E'. \end{cases}$$

For a D-item d we define
$$\overline{w}(d) = \begin{cases} 0.4 & \text{if } d \ge \frac{1}{2}, \\ 0.4 - \frac{3}{5}(\frac{1}{2} - d) & \text{if } d < \frac{1}{2}. \end{cases}$$

For every item a we define $\quad \overline{\overline{w}}(a) = \frac{6}{5}a$ and $w(a) = \overline{\overline{w}}(a) + \overline{w}(a)$.

For a set of items A and a set of bins $\mathcal{A}$, let $w(A)$ and $w(\mathcal{A})$ denote the total weight of all items in A or $\mathcal{A}$; similarly for $\overline{\overline{w}}$ and $\overline{w}$. Furthermore, let $W = w(I)$ be the total weight of all items of the instance I.

In the previous definition, the function $\bar{w}$ is continuous on the case boundaries. Furthermore, if we have a set A of k C-items not from E with size in $[\frac{1}{6}, \frac{1}{3}]$, then the definition implies that the bonus of A is exactly $\overline{w}(A) = \frac{3}{5}\left(s(A) - \frac{k}{6}\right)$. More generally, if A contains at least k items, each of size at least 1/6, and no D-item, then we get an upper bound $\overline{w}(A) \le \frac{3}{5}\left(s(A) - \frac{k}{6}\right)$.

First we analyze the weight of the OPT-bins.

▶ **Lemma 3.6.** *For every optimal bin A its weight $w(A)$ can be bounded as follows:*
(i) $w(A) \leq 1.7$.
(ii) *If E is the exceptional* OPT*-bin then $w(E) \leq 1.6$.*
(iii) *If A contains no D-item, then $w(A) \leq 1.5$.*

Proof. In all cases $\overline{\overline{w}}(A) \leq 1.2$, thus it remains to bound $\overline{w}(A)$. We distinguish three cases:

Case 1: A contains no D-item. Either it contains at least 4 items with non-zero bonus, in which case their total bonus is at most $\overline{w}(A) \leq \frac{3}{5}(s(A) - \frac{4}{6}) \leq \frac{3}{5} \cdot \frac{2}{6} = 0.2$. Or else it contains at most 3 items with non-zero bonus and $\overline{w}(A) \leq 0.3$. In both subcases (iii) follows and thus (ii) also holds if $E = A$.

Case 2: A contains a D-item larger than $1/2$. The bonus of the D-item is 0.4. If $E = A$ then A has no other item with non-zero bonus and both (i) and (ii) hold. Otherwise, in addition to the D-item, A contains at most 2 items larger than $1/6$ and no other items have non-zero bonus. If there is at most one such item, its bonus is at most 0.1 and (i) follows. If there are two such items, let their total size be y; note that $y < 1/2$. The bonus of A is at most $\overline{w}(A) \leq 0.4 + \frac{3}{5}(y - \frac{2}{6}) < 0.4 + \frac{3}{5} \cdot \frac{1}{6} = 0.5$.

Case 3: A contains d_0. Let the size of d_0 be $\frac{1}{2} - x$ for $x \geq 0$. We have $\overline{w}(d_0) = 0.4 - \frac{3}{5}x$. We distinguish two subcases.

Case 3.1: A contains at most two items other than d_0 and larger than $1/6$. Then their total size is at most $\frac{1}{2} + x$. If $E = A$ then they have no bonus and both (i) and (ii) hold. Otherwise their bonus is at most $0.1 + \frac{3}{5}x$ and (i) holds.

Case 3.2: If A contains at least three items other than d_0 and larger than $1/6$. Then their total bonus is at most $\frac{3}{5}x$, thus $\overline{w}(A) \leq 0.4$ and both (i) and (ii) hold. (This subcase may also happen if $E = A$, but there is no need to distinguish this in the proof.) ◀

Next we analyze the weight of FF-bins. The case of big and dedicated bins is easy:

▶ **Lemma 3.7.** (i) *The total weight of the big bins is $w(\mathcal{B}) \geq \beta$.*
(ii) *The total weight of the dedicated bins is $w(\mathcal{D}) > \delta$.*

Proof. (i): For every big bin B, $w(B) = \overline{\overline{w}}(B) = \frac{6}{5}s(B) \geq \frac{6}{5} \cdot \frac{5}{6} = 1$.

(ii): If d_0 is undefined then for every dedicated bin D, $w(D) = \frac{6}{5}s(D) + 0.4 > \frac{6}{5} \cdot \frac{1}{2} + 0.4 = 1$ and the claim follows. If d_0 exists and has size $\frac{1}{2} - x$ for $x \geq 0$, then every other D-item has size strictly larger than $\frac{1}{2} + x$. We also have $\delta \geq 3$ by Lemma 3.3(iii). Thus

$$w(\mathcal{D}) > (\delta - 1)\left(\frac{6}{5}\left(\frac{1}{2} + x\right) + 0.4\right) + \frac{6}{5}\left(\frac{1}{2} - x\right) + 0.4 - \frac{3}{5}x = \delta + \left((\delta - 1)\frac{6}{5} - \frac{6}{5} - \frac{3}{5}\right)x \geq \delta.$$

◀

Now we focus on the common FF-bins. The next lemma gives the key insight for the amortized analysis. It shows that for most common bins, the regular weight of the bin plus the bonus of the *next* common bin is at least 1. A similar method was used for the analysis of BESTFIT in [11]. For the rest of the upper bound section, number the common bins as $C_1, \ldots, C_\gamma$, in the order of their opening. The bins $C_2, \ldots, C_{\gamma-1}$ are called *inner* common bins. Note that there are some inner common bins, as $\gamma \geq 5$ by Lemma 3.3(iv).

▶ **Lemma 3.8.** *Let $i = 2, \ldots, \gamma$ be such that $s(C_{i-1}) \geq 2/3$. Then there exist two items $c, c' \in C_i \setminus E'$ and for any such items*

$$\overline{\overline{w}}(C_{i-1}) + \overline{w}(c) + \overline{w}(c') \geq 1\,.$$

Thus we have $\overline{\overline{w}}(C_{i-1}) + \overline{w}(C_i) \geq 1$.

Proof. If C_i is a 2-bin, then it contains no exceptional item. If C_i is a 3^+-bin, then it contains at most one exceptional item by Assumption 3.1. In both cases c and c' exist. Since C_{i-1} is common, the size of this bin is smaller than 5/6 and it is at least 2/3 by the assumption of the lemma. Let $x \in (0, \frac{1}{6}]$ be such that $s(C_{i-1}) = \frac{5}{6} - x$. Thus $c, c' > \frac{1}{6} + x$ and $\overline{w}(c), \overline{w}(c') > \frac{3}{5}x$. We get $\overline{\overline{w}}(C_{i-1}) + \overline{w}(c) + \overline{w}(c') > \frac{6}{5}(\frac{5}{6} - x) + \frac{3}{5}x + \frac{3}{5}x = 1$. ◀

3.3 The last common bin is large

The outline of the rest of the proof is this: We prove that the common FF-bins have total weight at least $\gamma - 0.2$. This part of analysis is considerably harder in case when the last common bin is smaller than 2/3, and we omit that part in this conference version. Then, since the total weight of the dedicated bins is strictly greater than δ, this implies $W > \text{FF} - 0.2$. Together with $W \leq 1.7 \cdot \text{OPT}$ now $\text{FF} \leq 1.7 \cdot \text{OPT} + 0.1$ follows. However, $\text{FF} = 1.7 \cdot \text{OPT} + 0.1$ can hold only if $\text{OPT} \equiv 7 \pmod{10}$. Then we show that the exceptional bin is defined, thus $W \leq 1.7 \cdot \text{OPT} - 0.1$ and we save the last 0.1.

▶ **Lemma 3.9.** *If $s(C_\gamma) \geq 2/3$, then the total weight of the common bins is $w(\mathcal{C}) \geq \gamma - 0.2$.*

Proof. First consider the case when every common bin has size at least 2/3. We apply Lemma 3.8 for every $i = 2, \ldots, \gamma$. The regular weight of the last bin is at least $\overline{\overline{w}}(C_\gamma) \geq \frac{6}{5} \cdot \frac{2}{3} = 0.8$. Summing all of these inequalities we obtain

$$w(\mathcal{C}) = \sum_{i=1}^{\gamma} w(C_i) \geq \overline{\overline{w}}(C_\gamma) + \sum_{i=2}^{\gamma} (\overline{\overline{w}}(C_{i-1}) + \overline{w}(C_i)) \geq 0.8 + (\gamma - 1) = \gamma - 0.2.$$

Now suppose that $s(C_k) = 2/3 - x$ for $x > 0$ and $k < \gamma$. Using Lemma 3.2(iii), each C_j, $j > k$, contains (exactly) two items larger than $1/3 + x$. Thus $\overline{w}(C_j) = 0.2$ and also $s(C_j) > 2/3 + 2x$ which implies $\sum_{i=k}^{\gamma} s(C_i) > (\gamma + 1 - k)\frac{2}{3}$. Combining these we have $\overline{\overline{w}}(C_k) + \sum_{j=k+1}^{\gamma} w(C_j) \geq (\gamma + 1 - k) - 0.2$. Adding the last inequality and the inequalities $\overline{\overline{w}}(C_{i-1}) + \overline{w}(C_i) \geq 1$ from Lemma 3.8 for $i = 2, \ldots, k$, the lemma follows. ◀

▶ **Lemma 3.10.** *Suppose $w(\mathcal{C}) \geq \gamma - 0.2$. Then*
(i) $\text{FF} \leq 1.7 \cdot \text{OPT} + 0.1$*, and*
(ii) *if the exceptional bin E is defined, then* $\text{FF} \leq 1.7 \cdot \text{OPT}$.

Proof. By Lemma 3.7 and the assumption we have $W > \beta + (\gamma - 0.2) + \delta = \text{FF} - 0.2$. By Lemma 3.6(i) we have $W \leq 1.7 \cdot \text{OPT}$. Thus $\text{FF} - 0.2 < W \leq 1.7 \cdot \text{OPT}$. Since FF and OPT are integers, (i) follows. If E is defined then by Lemma 3.6(i) and (ii) we have $W \leq 1.7 \cdot \text{OPT} - 0.1$. Thus $\text{FF} - 0.2 < W \leq 1.7 \cdot \text{OPT} - 0.1$ and (ii) follows. ◀

To decrease the bound by the last one tenth, we only need to show that the exceptional OPT-bin is defined. First yet another auxiliary lemma:

▶ **Lemma 3.11.** *Suppose that every* OPT*-bin contains a D-item. Then no* OPT *bin contains two 2-items c_1 and c_2.*

Proof. For contradiction, assume we have such c_1 and c_2 and number them so that the FF-bin of c_1 is before the FF-bin of c_2. (Note that by Assumption 3.1, c_1 and c_2 are not in the same FF-bin.) Let c_3 be the other item in the FF-bin of c_1. Since c_2 was not packed into this bin, which contains only c_1 and c_3, we have $c_1 + c_2 + c_3 > 1$. This implies that c_3 cannot be in the OPT-bin of c_1 and c_2. Every OPT-bin contains a D-item by the assumption; let d_1 be the D-item in the OPT-bin of c_1 and c_2 and d_3 the D-item in the OPT-bin of c_3. By Lemma 3.2(i), $d_1 + d_3 > 1$ and thus $c_1 + c_2 + c_3 + d_1 + d_3 > 2$. As all these items are in two OPT-bins, this is a contradiction. ◀

▶ Proposition 3.12. Suppose that $s(\mathcal{C}) \geq \gamma - 0.2$. Then $\mathrm{FF} \leq 1.7 \cdot \mathrm{OPT}$.

Proof. By Lemma 3.10(i) we have $\mathrm{FF} \leq 1.7 \cdot \mathrm{OPT} + 0.1$. If $\mathrm{OPT} \not\equiv 7 \pmod{10}$ then by checking all the other residue classes we can verify that $1.7 \cdot \mathrm{OPT} + 0.1$ is non-integral. Thus $\mathrm{FF} \leq 1.7 \cdot \mathrm{OPT} + 0.1$ implies $\mathrm{FF} \leq 1.7 \cdot \mathrm{OPT}$ and we are done.

It remains to handle the case when $\mathrm{OPT} \equiv 7 \pmod{10}$ and $\mathrm{FF} = 1.7 \cdot \mathrm{OPT} + 0.1$.

First we claim that every OPT-bin contains a D-item and thus $\delta = \mathrm{OPT}$. If some OPT-bin does not contain a D-item, its weight is at most 1.5 by Lemma 3.6(iii). Thus $W \leq 1.7 \cdot \mathrm{OPT} - 0.2$. Since $\mathrm{FF} > W - 0.2$, we obtain $\mathrm{FF} \leq 1.7 \cdot \mathrm{OPT}$, a contradiction.

Lemma 3.11 now implies that no OPT-bin contains two C2-items. Note that OPT is odd, as $\mathrm{OPT} \equiv 7 \pmod{10}$. On the other hand, the number of C2-items is even (in any FF-bin there are either zero or two C2-items). Thus some OPT-bin contains no C2-item. This bin satisfies all the conditions of Definition 3.4 of the exceptional bin. Thus E is defined and by Lemma 3.10(ii) the proposition follows. ◀

Together with the omitted case of $s(\mathcal{C}) < \gamma - 0.2$, we obtain our main result.

▶ **Theorem 3.13.** *For any instance of bin packing,* $\mathrm{FF} \leq 1.7 \cdot \mathrm{OPT}$.

4 Lower bounds

To prove the lower bounds, we use the classical lower bound construction from [5, 9]. We have an input instance L with three regions of items. In the first region there are items of size close to 1/6, in the second region come items close to 1/3, and in the third region there are items with the equal size $1/2 + \delta$, for a small $\delta > 0$. We will not modify the items in this list, only add some new items before or after L, and also in between the three regions of L. Thus we need to review the properties of L with the focus on the resulting FF packing in each region; the details within each region are somewhat delicate but fortunately we can use that part as a black box. We formulate the properties of L in the next lemma, before giving our lower bound in Theorem 4.2.

▶ **Lemma 4.1** ([5, 9]). *For every k and a sufficiently small $\delta > 0$ there exists an instance L of $30k$ items such that $\mathrm{OPT} = 10k + 1$ and $\mathrm{FF} = 17k$ for L. Furthermore the following holds for $\varepsilon = 46 \cdot 18^{k-1}\delta = O(\delta)$:*

(i) *The first $10k$ items of L have size at least $1/6 - \epsilon$ and are packed into the first $2k$ FF-bins; no further item is packed later into these bins. Each of these $2k$ FF-bins is a big 5-bin, and has size at least $5/6 + \delta$;*

(ii) *The next $10k$ items of L have size at least $1/3 - \epsilon$ and are packed into the next $5k$ FF-bins; no further item is packed later into these bins. Each of these FF-bins is a common 2-bin and has size at least $2/3 + 2\delta$.*

(iii) *The last $10k$ items of L have size exactly $1/2+\delta$ are packed into the next $10k$ FF-bins. Each of these FF-bins is a dedicated bin and has size exactly $1/2 + \delta$.*

(iv) *Moreover, all items of L, except three items, fit into $10k-1$ bins, each of size $1-O(\delta)$. The three remaining items have sizes $1/3 + \varepsilon$, $1/6 - 3\delta$, and $1/2 + \delta$.*

▶ **Theorem 4.2.** *For all integers $k \geq 1$ and $0 \leq i \leq 9$, there exists an instance I such that $\mathrm{OPT} = 10k + i$ and the lower bound in the top row of the next table holds. The bottom row of the table gives the upper bounds from Theorem 3.13 for a comparison.*

$i =$	0	1	2	3	4	5	6	7	8	9
$\mathrm{FF} \geq 17k +$	-1	1	3	4	6	8	10	11	13	15
$\mathrm{FF} \leq \lfloor 17k + 1.7i \rfloor = 17k +$	0	1	3	5	6	8	10	11	13	15

Furthermore, for $i = 1, \ldots, 9$ *there exist instances with* $\text{OPT} = i$ *and* $\text{FF} = \lfloor 1.7 \cdot i \rfloor$.

Proof. We show how the instance L from Lemma 4.1 can be modified to prove the theorem. We only show in each case that $\text{OPT} \leq 10k + i$. However, equality follows as the lower bound on FF is always larger than the upper bound on FF for $\text{OPT} - 1$ (see the table in the theorem).

For $i = 0$, we create I by deleting one item of size $1/2 + \delta$ from L. Then $\text{FF} = 17k - 1$. Optimum uses $10k - 1$ bins as in Lemma 4.1(iv). Only two items of sizes $1/3 + \varepsilon$ and $1/6 - 3\delta$ remain and they are packed into the last bin, thus obtaining $\text{OPT} \leq 10k$. For $k = 1, 2$, better examples with $\text{FF} = 17k$ and $\text{OPT} = 10k$ exist [5, 9], no such examples are known for $k > 2$.

For $i \geq 1$ and $k \geq 1$, we modify L by inserting new items. First we describe an optimal packing with $10k + i$ optimal bins together with the new items. The first $10k - 1$ bins contain the same items as in Lemma 4.1(iv). The $(10k)$th bin contains two of the remaining items from L, namely $1/2 + \delta$ and $1/6 - 3\delta$ and a new item $c_0 = 1/3 + 2\delta$. The $(10k + 1)$st bin contains the last remaining item from L, namely $1/3 + \varepsilon$, and two new items $d_0 = 1/2 + \delta/4$ and $b_0 = 1/6 - \delta/4 - \varepsilon$. If $i > 1$, then the $(10k + j)$th bin of the optimal packing, $j = 2, \ldots, i$, contains three new items $d_j = 1/2 + \delta/4$, $c_j = 1/3 + \delta/4$ and $b_j = 1/6 - \delta/2$.

The items b_j, c_j and d_j are called B-items, C-items and D-items, respectively; they are typically packed into big, common and dedicated bins of the optimum. We have exactly i new items of each type.

Now we describe the new instance I, together with the FF packing. The instance I consists of L and some of the new items. In some cases we do not need all new items. Then we remove the remaining new items; this can obviously only decrease the optimum, thus $\text{OPT} \leq 10k + i$.

All the new D-items are put at the end of L. Lemma 4.1 implies that they do not fit into any previous FF bin and thus FF puts each of them into a new dedicated bin. Furthermore, $2\lfloor i/2 \rfloor$ smallest new C-items are inserted in between the C-items and D-items in L. Since there is an even number of these new C-items and they do not fit into any of the previous bins, in FF packing they are put into $\lfloor i/2 \rfloor$ C-bins. Note that no D-item, old or new, does not fit into these new bins.

At this point we have created $\lfloor 3i/2 \rfloor$ new bins in the FF packing. Comparing this value with the table in the theorem, we have sufficiently many FF-bins for $i = 1, 2, 3, 4$, while for $i = 5, 6, 7, 8$ we need one additional FF-bin and for $i = 9$ two additional FF-bins. To create these bins, we have available all i new B-items and for odd i also one C-item, namely c_0, which is the largest one. We distinguish a few cases.

Case $i = 1, 2, 3, 4$: We discard all the remaining new items.

Case $i = 5$: We put one new C-item and four new B-items in front of L. They fit into a bin, thus FF packs them into the first bin and no other item fits in it. More precisely, the size of this bin is $1 - O(\delta)$, thus for a sufficiently small δ, no other item fits into it, as all the items have size at least $1/6 - O(\delta)$. The remaining B-item is discarded.

Case $i = 6, 7, 8$: We put 6 new B-items at the beginning of the list. They are packed in the first FF-bin and no other item will fit into it. The remaining items are discarded for $i = 7, 8$.

Case $i = 9$: We put 6 new B-items including b_0 at the beginning of the list. Again, they are packed in the first FF-bin and no other item will fit into it. We also insert $c_0 = 1/3 + 2\delta$, and the three remaining new B-items of size $1/6 - \delta/2$ between the B-items and C-items of L. None of these items fits in the previous bins, as those have size at least $5/6 + \delta$ by Lemma 4.1(i). Thus they are packed into one FF-bin of size about $5/6$. Since all the following items have size at least $1/3 - O(\delta)$, for a sufficiently small δ no further item fits into this bin. Thus this will be the second additional bin.

This completes the proof for $\textsc{Opt} \geq 10$. For $\textsc{Opt} \leq 9$, let $1 \leq i \leq 9$. Then I contains i items of each of the three sizes $1/6 - 2\delta$, $1/3 + \delta$, and $1/2 + \delta$. The items are ordered by non-decreasing size. It is easy to verify that for all $i = 1, \ldots, 9$, we have $\mathrm{FF} = \lfloor 1.7 \cdot i \rfloor$ and also $\textsc{Opt} = i$, as we can pack into each bin three items of different sizes. ◀

Acknowledgements We are grateful to anonymous referees for numerous helpful comments, in particular for correcting some historical references. G. Dósa was financially supported by the Hungarian State and the European Union under the TAMOP-4.2.2.A-11/1/KONV-2012-0072. J. Sgall was partially supported by the Center of Excellence – Inst. for Theor. Comp. Sci., Prague (project P202/12/G061 of GA ČR) and grant IAA100190902 of GA AV ČR; he is also grateful to Miloš Vyleťal and YMCA Setkání for the environment at Dům Setkání.

References

1 J. Boyar, G. Dósa, and L. Epstein. On the absolute approximation ratio for First Fit and related results. *Discrete Appl. Math.*, 160:1914–1923, 2012.

2 E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In D. Hochbaum, editor, *Approximation algorithms*. PWS Publishing Company, 1997.

3 G. Dósa. The tight bound of First Fit Decreasing bin-packing algorithm is $FFD(I) \leq 11/9\,OPT(I) + 6/9$. In *Proc. 1st International Symp. on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies (ESCAPE)*, volume 4614 of *Lecture Notes in Comput. Sci.*, pages 1–11. Springer, 2007.

4 M. R. Garey, R. L. Graham, D. S. Johnson, and A. C.-C. Yao. Resource constrained scheduling as generalized bin packing. *J. Combin. Theory Ser. A*, 21:257–298, 1976.

5 M. R. Garey, R. L. Graham, and J. D. Ullman. Worst-case analysis of memory allocation algorithms. In *Proc. 4th Symp. Theory of Computing (STOC)*, pages 143–150. ACM, 1973.

6 R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical J.*, 45:1563–1581, 1966.

7 R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.*, 17:263–269, 1969.

8 D. S. Johnson. *Near-optimal bin packing algorithms.* PhD thesis, MIT, Cambridge, MA, 1973.

9 D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM J. Comput.*, 3:256–278, 1974.

10 Z. Németh. A first fit algoritmus abszolút hibájáról (in Hungarian). Eötvös Loránd Univ., Budapest, Hungary, 2011.

11 J. Sgall. A new analysis of Best Fit bin packing. In *Proc. of 6th Int. Conference FUN with Algorithms*, volume 7288 of *Lecture Notes in Comput. Sci.*, pages 315–321. Springer, 2012.

12 D. Simchi-Levi. New worst case results for the bin-packing problem. *Naval Res. Logist.*, 41:579–585, 1994.

13 J. D. Ullman. The performance of a memory allocation algorithm. Technical Report 100, Princeton Univ., Princeton, NJ, 1971.

14 D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms.* Cambridge University Press, 2011.

15 B. Xia and Z. Tan. Tighter bounds of the First Fit algorithm for the bin-packing problem. *Discrete Appl. Math.*, 158:1668–1675, 2010.

Constrained Binary Identification Problem

Amin Karbasi[1] **and Morteza Zadimoghaddam**[2]

1 **I&C, EPFL, Switzerland,** `amin.karbasi@epfl.ch`
2 **CSAIL, MIT, Cambridge-MA-USA,** `morteza@mit.edu`

Abstract

We consider the problem of building a binary decision tree, to locate an object within a set by way of the least number of membership queries. This problem is equivalent to the "20 questions game" of information theory and is closely related to lossless source compression. If any query is admissible, Huffman coding is optimal with close to H$[P]$ questions on average, the entropy of the prior distribution P over objects. However, in many realistic scenarios, there are constraints on which queries can be asked, and solving the problem optimally is NP-hard.

We provide novel polynomial time approximation algorithms where constraints are defined in terms of "graph", general "cost", and "submodular" functions. In particular, we show that under graph constraints, there exists a constant approximation algorithm for locating the target in the set. We then extend our approach for scenarios where the constraints are defined in terms of general cost functions that depend only on the size of the query and provide an approximation algorithm that can find the target within $O(\log(\log n))$ gap from the cost of the optimum algorithm. Submodular functions come as a natural generalization of cost functions with decreasing marginals. Under submodular set constraints, we devise an approximation algorithm that can find the target within $O(\log n)$ gap from the cost of the optimum algorithm. The proposed algorithms are greedy in a sense that at each step they select a query that most evenly splits the set without violating the underlying constraints. These results can be applied to network tomography, active learning and interactive content search.

1998 ACM Subject Classification G.2.2 Graph Algorithms and Network Problems, I.1.2 Analysis of Approximation Algorithms, H.3.3. Information Search and Retrieval.

Keywords and phrases Network Tomography, Binary Identification Problem, Approximation Algorithms, Graph Algorithms, Tree Search Strategies, Entropy.

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.550

1 Introduction

Constructing a binary search tree is one of the fundamental problems in discrete mathematics. Formally, we are given a set $\mathcal{N} = \{1, \ldots, n\}$ of objects, identified with integers from 1 to n, as well as a probability distribution $P = (p_1, \ldots, p_n)$ over $\mathcal{N}$, $p_i \geq 0$, $i = 1, \ldots, n$, and $\sum_{i=1}^{n} p_i = 1$. The goal is to locate an object within a set $\mathcal{N}$ by way of the least number of membership queries. Each binary split is a partition of $\mathcal{N}$ into a subset $Q \subset \mathcal{N}$ and its complement $\mathcal{N} \setminus Q$. Given that a membership oracle provides answers without noise, what is the best we can do in a computationally feasible manner? This problem is equivalent to the "twenty questions game" of information theory [7], where a player has to determine the identity of an object from a set (say, a famous person) by asking a minimum number of yes/no questions. If any split is an admissible query, Huffman coding is optimal with close to H$[P]$ questions on average [7], the entropy of the distribution P from which the target object is drawn at random.

However, in realistic scenarios, there are *constraints* on which queries can be asked, and solving the problem optimally is NP-hard [15]. Instances of these constraints arise frequently in practice such as network tomography and interactive content search.

Network Tomography

A frequent situation that arises in sensor network tomography can be briefly explained as follows [4]. Suppose we are to remotely maintain a sensor network, whose nodes are severely restricted. First, being connected in a graph G, they can communicate with their neighbours only. Second, although they can receive broadcast messages from a base station, communication from any node back to the base must be kept to an absolute minimum due to power constraints. As nodes fail regularly, it is necessary to periodically search for faults. For simplicity, we assume that at most one node is faulty. To minimize the communication back to base, we should use the least number of broadcasts of the form "To nodes in Q: Is one of you faulty?". If Q is a connected subgraph of G, the question can be answered by a simple local message passing protocol, where a designated root in Q reports positively to the base only if it receives messages from all neighbours in Q, etc. Given that this does not happen, the root can poll the branch in question, whereby the faulty node can be identified recursively and passed back to the root, which reports it back to base. Finally, if the base does not get any reply after a reasonable time, it concludes that the chosen root must be faulty.

We address the constraints of this type under the general and realistic notion of *graph constraints* where we assume that the set $\mathcal{N}$ is furnished with an undirected graph structure G, and $Q \subset \mathcal{N}$ is an admissible query if and only if the subgraph $G|_Q$ induced by Q (containing all edges of G between vertices in Q) is connected. We then provide an efficient constant factor approximation to the "graph-constrained" twenty questions problem, which finds the target in $\leq 4H[P] + 2$ queries on average.

Interactive Content Search

In content search applications with humans involved, it is of obvious value to keep the number of queries as small as possible, as human interventions are disproportionally costly. Examples include visual recognition [2] and pattern classification [11], where questions are restricted to simple visual attributes. Whenever people are in the loop, queries are limited by their ability to meaningfully disambiguate in the presence of many attributes. As these examples show, it is usually not practical to allow for *any* query $Q \subset \mathcal{N}$. In these scenarios, it is more natural to define the constraints in terms of a cost rather than through a graph.

More formally, we are given a *non-decreasing cost function* $C : \{1, 2, \cdots, n\} \to \mathbb{R}$ where the cost of making an oracle query on set S is $C(|S|)$. Notice that the cost only depends on the size of the set and not the elements it contains. When a human plays the role of the membership oracle, it is more difficult to answer a query with a bigger set than one with a smaller set [2]. This is why in many interactive search strategies such a cost function arises naturally. In this case, we provide an approximation algorithm that can find the target within $O(\log(\log n))$ gap from the cost of the optimum algorithm.

To generalize our analysis beyond non-decreasing cost functions, we consider the case where constraints are defined in terms of a *submodular function* $Cost : 2^{\mathcal{N}} \to \mathbb{R}$. In this setting, for all subsets $S_1 \subseteq S_2 \subseteq \mathcal{N}$ and an item $i \in \mathcal{N}$ we have $Cost(S_1 \cup \{i\}) - Cost(S_1) \geq Cost(S_2 \cup \{i\}) - Cost(S_2)$. In words, adding a new item i to a larger set should produce an incremental cost no more than adding i to a smaller set. Under submudular constraints, we provide an approximation algorithm that can find the target within $O(\log(n))$ gap from the cost of the optimum algorithm. Further applications of submodular functions in active learning can be found in [12].

2 Related Work

The problem studied here can be seen as a special case of the binary identification problem (BIP) [10]. Suppose that we are given a set of objects $\mathcal{N}$ and a set of tests $\{t_1, \ldots .t_k\}$ where one of the objects is marked. Each test determines whether the marked object is in the test set or not. The goal is to define a strategy that minimizes the number of tests to find the marked object. It is known that both the average case minimization and worse case minimization are NP-complete [15]. Moreover, it is even NP-Hard to have an $o(\log n)$-approximation algorithm for the average case [5]. In both cases, there exist heuristic algorithms that admit $O(\log n)$-approximation [20, 6]. The closest work to ours is [14] where the authors study the same problem in a setting that a cost is associated with each test and the goal is to minimize the average total cost. They propose an $O(\log n)$-approximation algorithm. Unfortunately, there is no straight way to follow their approach and obtain similar performance guarantees for the submodular cost functions (for the other two problems that we consider in this paper, our approximation factors are better). Note that in this case, there are 2^n number of tests and a naive reduction admits an exponential running time. To overcome this barrier we propose a novel algorithm with a similar approximation guarantee.

Adding some structure to the set of tests leads to interesting special cases. For instance, if we let the set of tests be the power set of objects, then the optimal average case strategy is attained by Huffman coding [7]. Another basic variant of BIP is finding a marked element in a totally ordered set, a problem that is very well studied in the literature [19]. This can be generalized to searching in structures where the input has a partial order between its elements instead of a total order [1, 3]. It is known that searching in posets for the worst case minimization is NP-hard [3]. However, in [20], the authors showed a greedy algorithm with $O(\log n)$ approximation which was further improved to a constant factor approximation by [21]. It has been recently shown in [17] that the average case minimization is also NP-hard for the class of trees with diameter at most 4.

Constrained BIP is strongly related to *active learning* [8, 22, 13, 9, 18] in which a *hypothesis space* $\mathcal{H}$ is defined as a set of binary valued functions defined over a finite set $\mathcal{Q}$, called the *query space*. Each hypothesis $h \in \mathcal{H}$ generates a label from $\{-1, +1\}$ for every query $q \in \mathcal{Q}$. A target hypothesis h^* is sampled from $\mathcal{H}$ according to some prior P; asking a query q amounts to revealing the value of $h^*(q)$, thereby restricting the possible candidate hypotheses. The goal is to determine h^* by asking as few queries as possible. In our setting, the hypothesis space $\mathcal{H}$ is the set $\mathcal{N}$, and the query space $\mathcal{Q}$ is the set of all admissible subsets. The target hypothesis sampled from P is the target t_*. A well-known algorithm for determining the true hypothesis in the active-learning setting is the *generalized binary search* (GBS) or *splitting* algorithm [8, 22, 9]. Define the *version space* $V \subseteq \mathcal{H}$ to be the set of possible hypotheses that are consistent with the query answers observed so far. At each step, GBS selects the query $q \in \mathcal{Q}$ that minimizes $|\sum_{h \in V} P(h)h(q)|$. Recently, Golovin and Krause [12] showed that GBS makes at most $OPT \cdot \big(H_{\max}(\mu) + 1\big)$ queries in expectation to identify hypothesis $h^* \in \mathcal{N}$, where OPT is the minimum expected number of queries made by any adaptive policy and $H_{\max}(P) = \max_{x \in \mathrm{supp}(P)} \log \frac{1}{P(x)}$. In our setting, the version space V comprises all possible objects in $z \in \mathcal{N}$ that are consistent with answers given so far. Under graph, cost, and submodular constraints, we replace $H_{\max}$ (which in practice can be

Algorithm 1 The role of size

```
Input: Connected constraint graph $G = (\mathcal{N}, \mathcal{E})$, root node $v_R \in \mathcal{N}$.
  Grow a connected subtree $T \subset G$, starting from $v_R$, by a breadth-first search (BFS), until
  $T$ spans $\lceil n/2 \rceil$ vertices.
  Query the vertex set of $T$ (size $\lceil n/2 \rceil$).
  if $\mathrm{I}_{\{t_* \in T\}} = 1$ then
    Recurse on $T$, root $v_R$.
  else
    Collapse all nodes in $T$ into a single node $v_T$, which is connected to all $v \in \mathcal{N}$ adjacent
    to nodes in $T$ in the original $G$. Create $G'$ by Connecting all neighbours of $v_T$ to one
    another and removing $v_T$. Recurse on $G'$.
  end if
```

quite large) by a constant, $O(\log(\log(n)))$, and $O(\log(n))$, respectively.

The use of interactive methods for searching in a dataset has a long history in literature. Relevance feedback [24] is a method for interactive image retrieval, in which users mark the relevance of image search results, which then used to create a refined search query. We use the membership oracle to model the role of a user for identifying a target in a database. In practice, the cost of a query depends on the characteristics of the query, e.g., its size [11, 2]. However, due to the lack of analytical results, in many such applications only heuristic methods were proposed. To close this gap, we introduce general constraints, in terms of graph, cost and submodular functions, on the set of queries and establish analytical guarantees associated with them.

3 Graph Constraints

Let us assume that the set $\mathcal{N}$ is endowed with undirected graph structure, $G = (\mathcal{N}, \mathcal{E})$, where $\mathcal{E}$ contains the edges. For any subset $A \subset \mathcal{N}$, $G - A$ denotes the graph obtained from G by removing all vertices A and all edges adjacent to A, while $G|_A$ is the graph induced by A (vertex set A, edge set those $e \in \mathcal{E}$ between vertices in A).

Given G, a query $Q \subset \mathcal{N}$ is *admissible* if and only if the graph $G|_Q$ is connected. Our detection algorithm can submit any admissible query Q to a membership oracle, which initially sampled the target t_* at random from P: it will answer by one bit of information, $\mathrm{I}_{\{t_* \in Q\}} = 1$ if $t_* \in Q$, 0 otherwise. Our goal is to detect t_* with as few queries to the oracle as possible. Here, we restrict ourselves to deterministic policies which terminate only once the version space for t_* consistent with all previous queries $Q_1, \ldots, Q_k$ (formally, $(\cap_{k|t_* \in Q_k} Q_k) \cap (\cap_{k|t_* \notin Q_k} (\mathcal{N} \setminus Q_k))$) reaches size one. Throughout this section, we assume that the algorithm knows the distribution P the target is drawn from.

We begin with Algorithm 1, a simple divide-and-conquer scheme which detects the target with no more than $\lceil \log n \rceil$ queries, if seeded with an arbitrary $v_R \in \mathcal{N}$ as root node. Here and elsewhere, we use binary logarithms (base 2). Essentially, this algorithm mirrors the principle of binary search by way of intermediate sets produced during a breadth-first-search (BFS). Given adequate backpointer structures, its running cost is that of a single BFS.

▶ **Lemma 1.** *Presented with a connected constraint graph $G = (\mathcal{N}, \mathcal{E})$, where $|\mathcal{N}| = n$, Algorithm 1 detects the target t_* with no more than $\lceil \log n \rceil$ queries to the membership oracle.*

While Algorithm 1 is simple and efficient, it has the obvious drawback of not exploiting knowledge about the distribution P at all. However, it is useful as subroutine for our main algorithm developed next.

Suppose w.l.o.g. that $P = (p_1, \dots, p_n)$ is such that $p_1 \geq p_2 \geq \dots \geq p_n$. Define the nested subsets $A_1 \subset A_2 \subset \dots \subset \mathcal{N}$ by

$$A_i = \{1, \dots, j_i\}, \quad j_i = \min \left\{ j \;\middle|\; \sum\nolimits_{k>j} p_k \leq 2^{-i} \right\}.$$

Put differently, A_i is the smallest subset of $\mathcal{N}$ so that $\sum_{k \in A_i} p_k \geq 1 - 2^{-i}$. Also, define $a_i = \log(1/p_{j_i})$, so that $p_k \geq 2^{-a_i}$ for all $k \in A_i$, moreover $a_0 = 0$. The intuition behind this choice is that A_1 contains about half of the probability mass, $A_2 \setminus A_1$ half of the remaining mass, and so on.

Our algorithm processes the A_i in order, $i = 1, 2, \dots$. For A_i, it calls Algorithm 1, which will detect t_* if it is in A_i, otherwise we move to the next set. However, we cannot simply pass the induced subgraph $G|_{A_i}$ to Algorithm 1, as it may well not be connected. In this case, we generate $G_i = (A_i, V_i)$ passed to the algorithm as follows. We initialize $G_i \leftarrow G|_{A_i}$ and cluster the nodes into connected components. We then join each pair of disjoint components by a shortest path π, a central part of which necessarily features nodes $V(\pi)$ with $V(\pi) \cap A_i = \emptyset$. We collapse this part of π into a new *virtual* edge between nodes in A_i, which is labelled by the vertices $V(\pi)$ and added to V_i. This process stops when G_i is connected. Finally, when running Algorithm 1 on G_i, we need to translate queries back to connected subgraphs of the original G. For a query $Q \subset A_i$ in question, this is done by adding nodes with which virtual edges of $G_i|_Q$ are labelled. Our construction of G_i from A_i and labeling of virtual edges ensures that any query processed in this way corresponds to a connected subgraph of G, therefore is admissible.

Note that due to the presence of virtual edges, Algorithm 1 may receive positive answers from the oracle, even though $t_* \notin A_i$. After all, t_* might be among the vertices on virtual edges. However, in this case, Algorithm 1 simply descends to a single vertex $v_i \in A_i$, so that one more query $\{v_i\}$ settles the question "$t_* \in A_i$". Our main result is as follows.

▶ **Theorem 2.** *Given a connected constraint graph G with n vertices and any distribution P over $\{1, \dots, n\}$, our algorithm finds a target t_* sampled at random from P with no more than $2 + 4\mathrm{H}[P]$ admissible queries, on average over draws of t_*.*

Proof. We prepare our proof with a lemma whose proof can be found in the full version.

▶ **Lemma 3.** *For the numbers a_k defined above, we have that $\sum_{k=1}^{\infty} a_k/2^{k+1} \leq \mathrm{H}[P]$.*

To establish Theorem 2, recall that we visit sets A_i in order, $i = 1, 2, \dots$, calling Algorithm 1 on $G_i = (A_i, V_i)$ constructed as detailed above. Since $p_j \geq 2^{-a_i}$ for all $j \in A_i$, the size of A_i is bounded by 2^{a_i}, and Algorithm 1 returns after $\leq a_i$ queries. As noted above, due to "virtual edge" complications, we may have to query one more single node, yet after $\leq a_i + 1$ questions we know whether $t_* \in A_i$ or not, and in the former case will have detected t_*. Now, the probability of not finding t_* in A_i or earlier is bounded by $\sum_{j | p_j < 2^{-a_i}} p_j \leq 2^{-i}$. This means that the expected number of queries in in our algorithm is at most $\sum_{i \geq 1} (a_i + 1)/2^{i-1} \leq 2 + 4\mathrm{H}[P]$, where this inequality is due to Lemma 3. ◀

4 Cost Function Constraints

In this section we analyze another variant of the binary identification problem where the constraints are defined in terms of a cost function rather than a graph. More formally, we are given a non-decreasing cost function $C : \{1, 2, \cdots, n\} \to \mathrm{R}$ where the cost of making an

Algorithm 2 Binary Identification Algorithm with Cost Function Constraints

```
Input: $n$ objects with a probability distribution on them, and a cost function $C : \{1, 2, \cdots, n\} \to$ R, fixed constant $\epsilon > 0$.
  Create clusters $S_1, S_2, \dots S_l$ for $l = \log(n^2/\epsilon)$.
  (Phase one) Use Procedure 3 to determine which cluster contains the target $t_*$.
  (Phase two) If cluster $S_i$ contains the target, find it by using the dynamic program 2.
  (Phase three) If the target is not found in any of the above clusters, query each of the non-clustered objects.
```

oracle query on set S is $C(|S|)$[1]. Notice that the cost only depends on the size of the set not the elements it contains. When a human plays the role of the membership oracle, it is more difficult to answer a query with a bigger set than one with a smaller set. This is why in many interactive search strategies such a cost function arises naturally. To avoid confusion in using term "cost" for queries and algorithms, we formally define the notion of cost for sets and algorithms as follows.

▶ **Definition 4.** We refer to the cost of making a query on a set S by the “cost of set S". On the other hand, we use the term “expected search cost" of an algorithm to represent the expected value of total cost of queries the algorithm makes. Formally, An algorithm A consists of a family of possible queries $\mathcal{F}_A$. Algorithm A asks each query $S \in \mathcal{F}_A$ with probability $\Pr(A, S)$ which is a function of both the algorithm A, and the probability distribution of objects, P. We define the expected search cost of algorithm A to be $\sum_{S \in \mathcal{F}_A} \Pr(A, S) C(|S|)$.

By the above definition, it makes sense to talk about the expected search cost of Algorithm 2 or any other algorithm such as the optimum algorithm. We can also refer to the expected cost associated with a part of Algorithm 2 (it has three phases) and we can similarly define its expected cost in each phase. Note that the expected search cost of an algorithm is exactly the expected value of its total cost according to distribution P.

▶ **Definition 5.** Let $\epsilon > 0$ to be a small and fixed constant. We place objects in $l = \log(n^2/\epsilon)$ clusters based on their probabilities as follows. Let S_i be the cluster of objects with probability in range $(1/2^i, 1/2^{i-1}]$ for $1 \le i \le l$. For any $1 \le i \le l$, we define $\Pr(S_i)$ to be the sum of probabilities of objects in cluster S_i, and we define $\Pr[i, j] \doteq \sum_{x=i}^{j} \Pr(S_x)$ for $1 \le i \le j \le l$. We let $\Pr[i, j] = 0$ for $i > j$.

The choice of ϵ in the above definition is for ensuring that non-clustered objects have negligible probabilities, namely, at most ϵ/n^2.

Our algorithm for finding the target is shown in Algorithm 2. In phase one, we run a recursive algorithm that uses procedure *ClusterFinder* (shown in Algorithm 3) as a subroutine. The procedure *ClusterFinder* gets two numbers i and j, and finds the cluster containing the target only if the target is in one of the clusters $\{S_i, S_{i+1}, \cdots, S_j\}$. More specifically, Algorithm 3 finds a number k and a collection of clusters Q to query as follows:

$$Q = \bigcup_{i \le m \le k} S_m, \quad k = \max\left\{k' \,\middle|\, \Pr[i, k'] < \frac{\Pr[i, j]}{2}\right\} \tag{1}$$

[1] Note that there are two ways to determine whether or not the target lies in some set S. We can query set S or we can query the complement set $\mathcal{N} \setminus \mathcal{S}$. Here, we assume that the cost is always a function of the size of the queried set and it is the job of the algorithm to determine which set needs to be queried.

Algorithm 3 ClusterFinder(i,j)

```
Input: Clusters $S_i, S_{i+1}, \cdots, S_j$
1: Find the number $k$ and the set $Q$ according to Equation 1. Query set $Q$.
2: If $t_* \in Q$ call $ClusterFinder(i, k)$. Otherwise, query $S_{k+1}$.
3: If $t_* \in S_{k+1}$ return $S_{k+1}$. Otherwise, call $ClusterFinder(k + 2, j)$.
```

If the target is in Q, the algorithm calls procedure $ClusterFinder(i, k)$. Otherwise, it queries set S_{k+1}, and if the target is not there either, it calls $ClusterFinder(k + 2, l)$. Note that in Eq. 1, we might have $\Pr(S_i) = \Pr[i, i] \geq \Pr[i, j]/2$ which causes k to be $i - 1$. In this case, the set Q will be $\emptyset$.

▶ **Definition 6.** The procedure calls of Algorithm 3 can be represented by a binary tree T as follows. The root node is the procedure $ClusterFinder(1, l)$. The left child of the root is $ClusterFinder(1, k)$, and its right child is $ClusterFinder(k + 2, l)$. In the same fashion, we can define the rest of the tree nodes based on the recursive calls.

In phase two of Algorithm 2, we are given a cluster S_i that contains the target, and we want to find the target object. Let x be the number of objects in S_i. Our algorithm assumes that the probabilities of all objects in S_i are identical. Since they are in the same cluster, their probabilities are close to each other (we prove later that this assumption does not incur much cost for us). By this assumption we have symmetry among objects in cluster S_i. Therefore, the only relevant characterization of a subset of S_i is its size. We define a dynamic programming array $A[j]$, for $1 \leq j \leq x$, which is the expected search cost of optimal strategy to find the target given that the target is in a subset of S_i with size j under the assumption that these objects have the same probability, and therefore are identical. We can fill up the entries of array A as follows. It is clear that $A[1]$ is zero. We can update each $A[j]$ using the lower entries $A[1], A[2], \cdots, A[j-1]$ as follows:

$$A[j] = \min_{1 \leq j' < j} \{C(j') + A[j'](j'/j) + A[j - j']((j - j')/j)\}. \tag{2}$$

The optimal strategy chooses some $1 \leq j' < j$, and a subset of size j' among the remaining objects (it does not matter which subset it chooses, the only important factor is the size of the subset because of the identical probabilities assumption). Making a query on the selected subset of size j' has cost $C(j')$. With probability j'/j, the target is in the selected set, and we have to pay $A[j']$ in this case. With probability $(j - j')/j$, the target is not in the selected set, and we have to pay $A[j - j']$ in this case. Since we assume that S_i contains x elements, the optimal cost for finding the target in cluster S_i is therefore $A[x]$.

Finally, phase three of Algorithm 2 makes sure that if we have not found the target in the previous phases, we query each nonclustered object until we find the target.

▶ **Theorem 7.** *Let C_{Greedy} and C_{OPT} denote the expected search costs of Algorithm 2 and the optimum algorithm, respectively. Then, we have $C_{Greedy} = O(\log(\log(n))C_{OPT})$.*

Before we proceed to the proof of this theorem, let us consider the following definitions for *general* cost functions. Note that in general, the cost of a query is a function of the query set and not necessarily its size.

▶ **Definition 8.** Let $I(X)$ denote an instance of the search problem on a subset $X \subset \mathcal{N}$ where the probabilities of objects in X are normalized to make sure their sum is one, and we know that the target is in X. We denote by $Cost(X')$ the cost of making a query on

$X' \subset X$. Let also $Opt(I(X))$ represent the expected search cost of the optimum algorithm for instance $I(X)$. We define

$$Cost_X^i = \min\left\{Cost(X')|X' \subseteq X, \Pr(X') \geq 1 - 1/2^i\right\}.$$

Let $S(X,i)$ denote the subset for which $Cost(S(X,i)) = Cost_X^i$ and $\Pr(S(X,i)) \geq 1 - 1/2^i$.

Note that the particular cost function we consider in this section is $Cost(X') = C(|X'|)$. The following lemma will provide us with a general lower bound on the expected search cost of the optimum solution.

▶ **Lemma 9.** $Opt(I(X)) \geq \sum_{i=1}^{\infty} Cost_X^i/2^{i+1}$.

Lemma 9 help us bound the expected search cost of the optimum algorithm from below.

Proof of Theorem 7. We first show that the expected search cost caused by nonclustered objects is negligible.

▶ **Lemma 10.** *The expected search cost incurred by the nonclustered objects in phase three of Algorithm 2 is at most* $\epsilon \cdot C_{OPT}$.

Proof. The probability of each nonclustered objects is at most $1/2^l = \epsilon/n^2$. There are at most n non-clustered objects, so the probability of the target is one of these non-clustered objects is at most ϵ/n. With probability at most ϵ/n, we make a query for each non-clustered object. Hence, its expected search cost is at most $\epsilon n \times nC(1) = \epsilon C(1)$. On the other hand, note that $C(1)$ is a lower bound for the optimum solution because no matter where the target is we always have to make at least a query of size at least one to find the target. ◀

Lemma 10 entails that the expected search cost incurred by nonclustered objects is much less than the optimal expected search cost. Thus for simplicity we can assume that there exists no nonclustered object. In order to bound the expected search cost of the first phase of Algorithm 2 we need a few intermediate results. Let us start with the following definition.

▶ **Definition 11.** A set of nodes S of a tree T' is sparse, if and only if there do not exist three nodes v_1, v_2 and v_3 in S such that v_1 is one of the ancestors of v_2 and v_3, and neither of v_2 and v_3 is an ancestor of one another.

Hence, in a sparse set S there cannot be two nodes such that they don't have an ancestor/-descendant relationship and simultaneously have a common ancestor in S. The following lemma bounds the number of possible sparse partitions of any tree

▶ **Lemma 12.** *The nodes of any tree* T' *can be partitioned into* $\lceil \log(|T'|) \rceil$ *sparse sets where* $|T'|$ *is the number of nodes in* T'.

To link the expected search cost of the first phase of Algorithm 2 to nodes of the tree T, we need the following definition.

▶ **Definition 13.** Let a node $v \in T$ represent procedure $ClusterFinder(i,j)$ for some $1 \leq i \leq j \leq l$. With probability $\Pr[i,j]$, we go to this node, and we make a query on the set Q as defined in Eq. 1. If the target is not in Q (i.e., it is in one of the clusters $S_{k+1}, S_{k+2}, \cdots, S_j$), we make a second query on the set S_{k+1}. Hence, the expected search cost of our algorithm at node v is $\Pr[i,j]C(|Q|) + \Pr[k+1,j]C(|S_{k+1}|)$. We define this quantity as the price of node v.

It is clear that the expected search cost of our algorithm in phase 1 is the sum of the prices of all nodes in tree T. In Lemma 14, we bound the sum of prices of nodes of a sparse set which provide us with the key result to bound the expected search cost of the first phase. The proof of Lemma 14 heavily relies on the observation we made in Lemma 9.

▶ **Lemma 14.** *The sum of prices of nodes of a sparse set is* $O(C_{OPT})$.

Due to the above lemma, we know that the total prices of the nodes of every sparse set is $O(C_{OPT})$. By recalling Lemma 12 we also know that we can partition the nodes of T into $O(\log(\log(n)))$ sparse sets. Hence, Lemma 14 together with Lemma 12 readily bounds the expected search cost of the first phase which is the sum of prices of all nodes in tree T.

▶ **Lemma 15.** *The expected search cost for identifying which cluster contains the target* t_* *(first phase of Algorithm 2) is* $O(\log(\log(n)) \cdot C_{OPT})$.

We are ready to analyze the second phase of algorithm 2. In this part, we know the cluster, say S_i, that contains the target t_* and we just need to find it. Let us define I_i to be this instance: we are given the information that the target is in S_i, and we have to find it. Following Definition 8, we denote the optimum expected search cost to identify the target in I_i by $OPT(I_i)$. The following lemma bounds the expected search cost of Algorithm 2 (phase two) in terms of $OPT(I_i)$.

▶ **Lemma 16.** *Assume that the target* t_* *is in cluster* S_i. *Then, the expected search cost of phase two of Algorithm 2 to find the target is at most* $2\Pr(S_i)OPT(I_i)$.

Lemma 16 helps us relate the expected search cost of phase two to C_{OPT}.

▶ **Lemma 17.** *The expected search cost of phase two of Algorithm 2 is at most* $2C_{OPT}$.

By combining Lemmas 10, 15, and 17 we can conclude that the expected search cost of Algorithm 2 is $O(\log(\log(n)) \cdot C_{OPT})$. ◀

5 Submodular Constraints

In this section we analyze another variant of the binary identification problem where the constraints are defined in terms of a submodular function. More specifically, we are given a set of objects $\mathcal{N} = \{1, 2, \cdots, n\}$ and a non-decreasing submodular function $C_{sub} : \mathcal{F}_n \to \mathbb{R}$ where $\mathcal{F}_n = 2^{\mathcal{N}}$ is the family of all subsets of $\mathcal{N}$. We denote the cost of making a query on the set $S \subset \mathcal{N}$ by $C_{sub}(S)$ where we assume that the cost function is submodular:

$$\forall S_1 \subseteq S_2 \subseteq U,\ i \in U : C_{sub}(S_1 \cup \{i\}) - C_{sub}(S_1) \geq C_{sub}(S_2 \cup \{i\}) - C_{sub}(S_2).$$

Intuitively, this property insures that adding an object i to a set S_1 increases its cost at least as much as adding i to a superset S_2. Another equivalent definition that we later use reads as follows: $\forall S_1, S_2 \subseteq \mathcal{N}, C_{sub}(S_1) + C_{sub}(S_2) \geq C_{sub}(S_1 \cup S_2) + C_{sub}(S_1 \cap S_2)$. In view of Definition 8 the cost function of a subset $S \in \mathcal{N}$ is $Cost(S) = C_{sub}(S)$.

Before elaborating on our new algorithm, we should note that it is possible to mimic the first phase of Algorithm 2 and obtain an approximation factor of $O(\log(\log(n)))$. However, it is no longer possible to run the second phase with the same approximation factor for submodular functions. Recall that in the second phase, we relied on a dynamic program that only depends on the size of the queried sets. Due to the fact that in our new setup, the cost of a query depends on the set (an not only on its size), we need to reformulate the dynamic program: it matters which subset is chosen. To do so, we ought to construct an

Algorithm 4 Bicriteria Approx. Alg.

```
Input: $\epsilon > 0$.
1: Find all $\alpha_k$'s in (3) and their corresponding sets $S_{\alpha_k}$.
2: Among all $S_{\alpha_k}$, keep only those that have $\Pr(S_{\alpha_k}) \geq 1/6$.
3: Among the remaining sets, choose the one with minimum $C_{sub}(S_{\alpha_k})$. Call this set $S'$.
4: Keep removing objects from $S'$ while its probability remains at least $1/6$.
```

Algorithm 5 BIP with Submod. Cnst.

```
Input: Set of objects $\mathcal{N}$.
1: Find set $S' \subset \mathcal{N}$ (and $\bar{S}' = \mathcal{N} \setminus S'$) by using Algorithm 4. Query $S'$.
2: if set $S'$ contains the target $t_*$ then
3:     Query one of the objects $i \in S'$. Either $i$ is the target, or otherwise recurs on $S' \setminus \{i\}$.
4: else
5:     Recurs on $\bar{S}'$.
6: end if
```

exponential size array (one for each subset) and as a result, the algorithm will no longer run in polynomial time. To avoid this drawback, we can devise a $\log(n)$-approximation algorithm for the second phase of Algorithm 2 which can incorporate submodular constraints. Instead, in this section we present a much simpler algorithm with the same approximation guarantee. To do so, we first need an intermediate step.

Submodular Functions Under Linear Constraints

The problem of “minimizing submodular functions under linear constraints" is to find a subset $S^* \in \mathcal{N}$ such that $\Pr(S^*) \geq 1/2$ and its cost, namely $C_{sub}(S^*)$, is minimum. Here, we present a bicriteria approximation algorithm that finds a set S' with $C_{sub}(S') \leq 3C_{sub}(S^*)$ and $\Pr(S') \geq 1/6$. To do so, we first need to define another submodular function $f^\alpha : 2^{\mathcal{N}} \to \mathbb{R}$ for which $f^\alpha(S) = C_{sub}(S) + \alpha \cdot \Pr(\bar{S})$, where $\bar{S} = \mathcal{N} \setminus S$, and $\alpha > 0$ is a real number. The reason that f^α is a submodular function simply follows from the fact that we only added a liner term to the submodular function C_{sub} (check the equivalent definition we provided earlier). It is a folklore result that submodular function minimization problem has strongly polynomial time exact algorithms due to [23] and [16]. Hence, one can find a subset $S_\alpha \subseteq \mathcal{N}$ in polynomial time that admits the minimum value of f^α. Let $c_{\min} = \min_{i\in\mathcal{N}} C_{sub}(\{i\})$. For a fixed $\epsilon > 0$ we also define

$$\alpha_k = 3c_{\min}(1+\epsilon)^k, \quad \text{for all } k = 0, 1, \ldots, \lceil \log_{1+\epsilon}(C_{sub}(\mathcal{N}))/c_{\min} \rceil. \tag{3}$$

Note that there exists a $k = \kappa$ for which $3C_{sub}(S^*) \leq \alpha_\kappa \leq 4C_{sub}(S^*)$.

▶ **Lemma 18.** *Let S_{α_κ} denote the set that admits the minimum of the function f^α for $\alpha = \alpha_\kappa$. Then $C_{sub}(S_{\alpha_\kappa}) \leq 3C_{sub}(S^*)$ and $\Pr(S_{\alpha_\kappa}) \geq 1/6$.*

Since the value of $C_{sub}(S^*)$ is not known a priori, we cannot apply Lemma 18 without modification. To this end, we propose Algorithm 4. It first calculates all values of α_k. For each α_k it finds the corresponding set S_{α_k} that minimizes f^{α_k}. It then finds among those S_{α_k} with $\Pr(S_{\alpha_k}) \geq 1/6$, the one that has minimum cost $C_{min}(S_{\alpha_k})$. Once such a set S' is found, it starts removing objects from S' while keeping $\Pr(S') \geq 1/6$.

▶ **Lemma 19.** *Algorithm 4 outputs a set $S' \subset \mathcal{N}$ with $C_{sub}(S') \leq 3C_{sub}(S^*)$ and $\Pr(S') \geq 1/6$. Moreover, for any $i \in S'$, we have $\Pr(S' \setminus \{i\}) < 1/6$.*

Approximation Algorithm with Submodular Constraints

Algorithm 5 starts by finding a set $S' \subset \mathcal{N}$. To this end, it calls on Algorithm 4. According to Lemma 19, for the set S' we have $\Pr(S') \geq 1/6$ and $C_{sub}(S') \leq 3C_{sub}(S^*)$. Remember S^*

is the set with minimum cost and the probability at least a half. Ideally, we would like to query the set S^*. Unfortunately, it is not easy to find such a set. Instead, Algorithm 5 queries the set S' that approximate our ideal candidate. If the target t_* is in S', then Algorithm 5 chooses an arbitrary object $i \in S'$ and see whether i is the target or not. In case it is not the target, the whole procedure repeats now from the set $S' \setminus \{i\}$, namely, Algorithm 4 will be called for the set $S' \setminus \{i\}$. If the target is not in S' from the beginning, the algorithm recurs on set $\bar{S}' = \mathcal{N} \setminus S'$. Note that by making a singleton query before recursion, Algorithm 5 makes sure that the probability of the remaining set will shrink by a factor of 1/6 (Lemma 19) if the target is in set $S' \setminus \{i\}$. Otherwise, the probability shrinks by a factor of 5/6 because we have that $\Pr(\bar{S}') = 1 - \Pr(S') \leq 1 - 1/6$.

▶ **Theorem 20.** *Let C_{Greedy} and C_{OPT} denote the expected search costs of Algorithm 5 and the optimum algorithm, respectively. Then, we have $C_{Greedy} = O(\log(n) \cdot C_{OPT})$.*

Proof. The proof of this theorem is similar, *mutatis mutandis*, to the proof of phase one of Theorem 7. Let us first modify Definition 8 as follows.
Let $Cost^i_X = \min_{X' \subseteq X | Pr(X') \geq 1-(5/6)^i} Cost(X')$. Then, we can show

▶ **Lemma 21.** $Opt(I(X)) \leq \sum_{i=1}^{\infty} \frac{Cost^i_X}{(5/6)^{i+1}/5}$.

The search mechanism portrayed in Algorithm 5 defines a binary tree T, similar to the one we saw in the previous section. In brief, the tree starts from the root $\mathcal{N}$. In each internal node S with probability at least 1/6 we go to the left child (representing set $S' \setminus \{i\}$) and with the remaining probability we go to the right child (representing $S \setminus S'$). Note that there are at most $2n$ nodes in this tree where the probability of each child is at most 5/6 of the probability of its father (instead of 1/2 in the previous section). Because the tree T has at most $2n$ nodes, instead of $O(\log\log(n))$ sparse sets, we need $\log(2n)$ sparse sets to cover the expected cost of all nodes. As a result one can provide a similar proof showing that the expected cost of Algorithm 5 with submodular constraints is at most $O(\log(n))$ times the cost of the optimum solution.

◀

6 Conclusion

In this work we considered the problem of binary identification problem (BIP) under the general notion of graph, cost and submodular constraints. We also provided novel polynomial time approximation algorithms with provable analytical guarantees. Even though we believe that the three variants of BIP we considered are all NP-hard, we did not provide any rigorous proof. This is an interesting future direction we would like to pursue.

References

1 Yosi Ben-Asher, Eitan Farchi, and Ilan Newman. Optimal search in trees, 1999.

2 Steve Branson, Catherine Wah, Florian Schroff, Boris Babenko, Peter Welinder, Pietro Perona, and Serge Belongie. Visual recognition with humans in the loop. In *ECCV (4)*, pages 438–451, 2010.

3 R. Carmo, J. Donadelli, Y. Kohayakawa, and E. Laber. Searching in random partially ordered sets. *Theor. Comput. Sci.*, 321:41–57, 2004.

4 R. Castro, M. Coates, G. Liang, R. Nowak, and B. Yu. Network tomography: Recent developments. *Statistical Science*, 19(3):499–517, 2004.

5 Venkatesan T. Chakaravarthy, Vinayaka Pandit, Sambuddha Roy, Pranjal Awasthi, and Mukesh K. Mohania. Decision trees for entity identification: Approximation algorithms and hardness results. *ACM Transactions on Algorithms*, 7(2):15, 2011.

6 Venkatesan T. Chakaravarthy, Vinayaka Pandit, Sambuddha Roy, and Yogish Sabharwal. Approximating decision trees with multiway branches. In *ICALP (1)*, pages 210–221, 2009.

7 Thomas M. Cover and Joy Thomas. *Elements of Information Theory.* Wiley, 1991.

8 S. Dasgupta. Analysis of a greedy active learning strategy. *NIPS*, 2005.

9 M. R. Garey and R. L. Graham. Performance bounds on the splitting algorithm for binary testing. *Acta Informatica*, 3:347–355, 1974.

10 M.R. Garey. Optimal binary identification procedures. In *SIAM J. Appl. Math*, 1972.

11 Donald Geman and Bruno Jedynak. Shape recognition and twenty questions. Technical report, in Proc. Reconnaissance des Formes et Intelligence Artificielle (RFIA, 1993.

12 Daniel Golovin and Andreas Krause. Adaptive submodularity: A new approach to active learning and stochastic optimization. *Journal of Artificial Intelligence Research (JAIR)*, 42:427–486, 2011.

13 Andrew Guillory and Jeff A. Bilmes. Average-case active learning with costs. In *ALT*, pages 141–155, 2009.

14 Anupam Gupta, Viswanath Nagarajan, and R. Ravi. Approximation algorithms for optimal decision trees and adaptive tsp problems. In *Proceedings of the 37th international colloquium conference on Automata, languages and programming*, ICALP'10, pages 690–701, Berlin, Heidelberg, 2010. Springer-Verlag.

15 L. Hyafil and R. Rivest. Constructing optimal binary decision trees is np-complete. In *Information Processing Letters*, pages 15–17, 1976.

16 Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *J. ACM*, 48(4):761–777, 2001.

17 Tobias Jacobs, Ferdinando Cicalese, Eduardo Laber, and Marco Molinaro. On the complexity of searching in trees: average-case minimization. In *Proceedings of the 37th international colloquium conference on Automata, languages and programming*, ICALP'10, pages 527–539, 2010.

18 Amin Karbasi, Stratis Ioannidis, and Laurent Massoulie. Comparison-based learning with rank nets. In *29th International Conference on Machine Learning (ICML)*, 2012.

19 D. Knuth. *The Art of Computer Programming*, volume 3. Addison Wesley Longman Publishing Co., Inc, 1998.

20 S. Rao Kosaraju, Teresa M. Przytycka, and Ryan S. Borgstrom. On an optimal split tree problem. In *Proceedings of the 6th International Workshop on Algorithms and Data Structures*, WADS '99, pages 157–168. Springer-Verlag, 1999.

21 Eduardo Laber and Marco Molinaro. An approximation algorithm for binary searching in trees. In *Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part I*, pages 459–471, Berlin, Heidelberg, 2008. Springer-Verlag.

22 R.D. Nowak. The geometry of generalized binary search. *Transactions on Information Theory*, 5, 2012.

23 Alexander Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *J. Comb. Theory, Ser. B*, 80(2):346–355, 2000.

24 Xiang S. Zhou and Thomas S. Huang. Relevance feedback in image retrieval: A comprehensive review. *Multimedia Systems*, 8(6):536–544, 2003.

Regular languages of thin trees

Mikołaj Bojańczyk, Tomasz Idziaszek, and Michał Skrzypczak

University of Warsaw[*]
`{bojan,idziaszek,mskrzypczak}@mimuw.edu.pl`

Abstract

An infinite tree is called thin if it contains only countably many infinite branches. Thin trees can be seen as intermediate structures between infinite words and infinite trees. In this work we investigate properties of regular languages of thin trees.

Our main tool is an algebra suitable for thin trees. Using this framework we characterize various classes of regular languages: commutative, open in the standard topology, closed under two variants of bisimulational equivalence, and definable in WMSO logic among all trees.

We also show that in various meanings thin trees are not as rich as all infinite trees. In particular we observe a parity index collapse to level $(1, 3)$ and a topological complexity collapse to co-analytic sets. Moreover, a *gap property* is shown: a regular language of thin trees is either WMSO-definable among all trees or co-analytic-complete.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases infinite trees, regular languages, effective characterizations, topological complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.562

1 Introduction

Since the decidability results by Büchi [7] and Rabin [17], regular languages of infinite words and trees have been intensively studied. Those languages can be equivalently described in monadic second-order (MSO) logic, by nondeterministic finite automata, or in terms of homomorphisms to finite algebras. Apart from the emptiness problem, which is known to be decidable, one ask about decidability for other, more subtle properties of a given language.

Suppose that X is a subclass of regular languages of infinite trees, e.g. X can be the languages that are definable in first-order (FO) logic with descendant; or definable in weak MSO (WMSO); or recognized by a nondeterministic parity automaton with priorities $\{i, \dots, j\}$. An *effective characterization for* X is an algorithm which inputs a regular language of infinite trees and answers if the language belongs to X. As far as decidability is concerned the representation of the language is not very important, since there are decidable translations between the many ways of representing regular languages of infinite trees.

Effective characterizations are a lively and important topic in the theory of regular languages. In the case of finite words there are many celebrated results, e.g. characterizations of FO [18], two-variable FO [21] or piecewise testable languages [19]. Many of these results carry over to infinite words, see [23], [16], or [12]. For finite trees much less is known, but still there are some techniques [3]. The main reason why effective characterizations are studied is that an effective characterization of a class X requires a deep insight into the structure of the class. Usually this insight is achieved through an algebraic framework, such as semigroups for finite words, Wilke semigroups for infinite words, or forest algebra for finite trees. Apart

* All authors were supported by ERC Starting Grant "Sosna" no. 239850

from having a well-developed structure theory, another advantage of algebra is that many effective characterizations can be elegantly stated in terms of identities.

Effective characterizations are technically challenging, and in fact there are very few effective characterizations for languages of infinite trees: for languages recognized by top-down deterministic automata one can compute the Wadge degree [14], for arbitrary regular languages one can decide definability in the temporal logic EF [4] or in the topological class of Boolean combinations of open sets [5]. One of the reasons why effective characterizations are so difficult for infinite trees is that, so far, there is no satisfactory algebraic approach to infinite trees, or even a canonical way to present a regular language. Proposed algebras (see [4], [2]) either have no finite representation or yield no effective characterizations.

In this paper, we propose to study *thin trees*, which generalize both finite trees and infinite words, but which are still simpler than arbitrary infinite trees. A tree is called thin if it has only countably many infinite branches (or equivalently, it does not contain a full binary tree as a minor). We believe that thin trees are a good stepping stone on the way to understanding regular languages of arbitrary infinite trees.

Our contributions can be divided into two sets:

Effective characterizations. We characterize the following classes of regular languages of thin trees in terms of finite sets of identities:

- closed under rearranging of siblings,
- closed under bisimulation equivalence (in two variants),
- open in the standard topology,
- definable in the temporal logic EF,
- definable among all forests in WMSO logic.

The crucial ingredient of these characterizations is an observation that a regular language of thin trees can be canonically represented by a finite algebraic object, called its syntactic thin-forest algebra. For general trees no such representation is known.

Upper bounds. We show that in various contexts thin trees are not as rich as generic trees:

- The Rabin-Mostowski index hierarchy collapses to level $(1, 3)$ on thin trees.
- The projective hierarchy of regular languages collapses to level $\mathbf{\Pi}^1_1$ on thin trees (comparing to $\mathbf{\Delta}^1_2$ in the case of all trees).
- We observe a *gap property* (see [15]): a regular language of thin trees treated as a subset of all trees is either definable in WMSO logic or non-Borel.
- If we treat thin trees as our universe then no regular language is topologically harder than Borel sets.

2 Preliminaries

This section introduces basic notions and facts used in the proofs. To avoid technical difficulties when introducing algebras, we operate on finitely branching forests instead of partial binary trees. The difference is only technical, all the results can be naturally transferred back to the framework of partial binary trees.

2.1 Forests

Fix a finite alphabet A. By A^{For} we denote the set of all A-labelled forests. Formally a forest is a partial mapping from its set of nodes $\mathrm{dom}(t) \subset \omega^+$ into A. We additionally assume that a forest is finitely branching: for every $w \in \omega^*$ there are only finitely many nodes of the form $w0, w1, w2, \ldots, wn$ in $\mathrm{dom}(t)$. For $w = \epsilon$ those nodes are called *roots* of the forest

t and for $w \neq \epsilon$ these are *children* of the node w. In both cases the list of nodes of the form wn ordered by n is called a *list of siblings in* t.

A node $w \in \mathrm{dom}(t)$ is *branching* if it has at least two distinct children $wn_1, wn_2 \in \mathrm{dom}(t)$. A node in $\mathrm{dom}(t)$ is a leaf of t if it has no children in t.

A forest with exactly one root is called a *tree*. The empty forest is denoted as 0. For a given forest t and a node $x \in \mathrm{dom}(t)$ by $t\restriction_x$ we denote the subtree of t rooted in x: $\mathrm{dom}(t\restriction_x) = \{0 \cdot w \in \omega^* :\ xw \in \mathrm{dom}(t)\}$, $t\restriction_x (0 \cdot w) = t(xw)$.

Let t be a forest. A sequence $\pi \in \omega^*$ is a *finite branch of* t if either $\pi = \epsilon$ and $t = 0$ or $\pi \in \mathrm{dom}(t)$ and π (as an element of ω^+) is a leaf of t. A sequence $\pi \in \omega^\omega$ is an *infinite branch of* t if for every sequence $w \in \omega^+$ such that $w \prec \pi$ we have that w is a node of t.

A forest is *regular* if it has finitely many distinct subtrees. A forest is *thin* if it has countably many branches. The set of all thin forests is denoted as $A^{\mathsf{ThinFor}} \subset A^{\mathsf{For}}$. A forest is thin if and only if it is a *tame tree* in the meaning of [13].

We say that a forest s is a *prefix* of a forest t if $\mathrm{dom}(s) \subseteq \mathrm{dom}(t)$ and for every $x \in \mathrm{dom}(s)$ we have $s(x) = t(x)$. We denote it by $s \subseteq t$.

Let t be a forest and $s \subseteq t$ be a prefix of t. A node $y \in t$ is *off* s if $y \notin s$ and either y is a root, or the parent of y is in s. Since a branch π of t can be treated as a prefix of t this definition also extends to branches.

An A-labelled *context* is a forest over the alphabet $A \cup \{\Box\}$, where the label $\Box$ is a special marker, called the *hole*, which occurs exactly once and in a leaf. A context is *guarded* if its hole is not in a root. For every letter $a \in A$ we denote by $a\Box$ the single-letter tree context with a in the root and the hole below it.

Since we are interested in algebraic frameworks for forests, we need a set of operations which will allow to build forest from basic elements. Following [6] we introduce following operations on forests. For a graphical presentation of these operations, compare Figure 1 and Figure 2 in [6]. We can

- concatenate two forests s, t, which results in the forest $s + t$,
- compose a context p with a forest t, which results in the forest pt, obtained from p by replacing the hole with t,
- compose a context p with a context q, which results in the context pq that satisfies $(pq)t = p(qt)$.

We write at, ap for a composition of a single-letter context $a\Box$ with t or p (thus $a0$ is a forest of one node labelled a). Additionally we have an operation which allows us to produce infinite forests:

- compose a guarded context p with itself infinitely many times, which results in the forest p^∞ that satisfies $p(p^\infty) = p^\infty$. Note that we exclude non-guarded contexts from this definition. (For example the result of $(\Box + a0)^\infty$, even if well-defined, is not finitely branching.)

2.2 Automata and regular languages

A (nondeterministic parity) *forest automaton* over an alphabet A is given by a set of states Q equipped with a monoid structure, a transition relation $\Delta \subseteq Q \times A \times Q$, a set of initial states $Q_I \subseteq Q$ and a parity condition $\Omega \colon Q \to \mathbb{N}$. We use additive notation $+$ for the monoid operation in Q, and we write 0 for the neutral element.

We say that a forest automaton $\mathcal{A}$ *has index* (i, j) (or shortly that $\mathcal{A}$ is (i, j)-automaton) if i is the minimal and j is the maximal value of Ω on Q.

A *run* of this automaton over a forest t is a labelling $\rho\colon \mathrm{dom}(t) \to Q$ of forest nodes with states such that for any node x with children $x_1, \ldots, x_n$

$$(\rho(x_1) + \rho(x_2) + \cdots + \rho(x_n), t(x), \rho(x)) \in \Delta.$$

Note that if x is a leaf, then the above implies $(0, t(x), \rho(x)) \in \Delta$.

A run is *accepting* if for every (infinite) branch π of t, the highest value of $\Omega(q)$ is even among those states q which appear infinitely often along the branch π. The *value* of a run over a forest t is obtained by adding, using $+$, all the states assigned to roots of the forest. A forest is *accepted* if it has an accepting run whose value belongs to Q_I. The set of forests accepted by an automaton is called the language *recognized* by the automaton.

A language is *regular* if it is definable by a formula of monadic second-order logic (MSO).

▶ **Theorem 1** ([10]). *A language of thin forests is regular if and only if it is recognized by some forest automaton. Every nonempty language of thin forests contains a regular forest.*

We use MSO logic to describe properties of infinite forests. An infinite forest is treated as a relational structure, where the universe is the nodes, and the predicates are: a binary child predicate, a binary next sibling predicate, and one unary predicate for each label in the alphabet. Additionally, we consider WMSO: the logic with the same syntax as MSO but with the semantical restriction that all set quantifiers range over finite subsets of the domain. Since the property that a given set is finite is MSO-definable on finitely branching infinite forests, so WMSO can be naturally embedded into MSO. There are examples of languages of infinite forests that are definable in MSO but not in WMSO.

2.3 Topology

A topological space X is *Polish* if it is separable and has a complete metrics. Polish topological spaces are the principal objects studied in descriptive set theory.

The set of forests A^{For}, equipped with the natural Tikhonov topology, is an uncountable Polish topological space. The base of the topology is given by the sets of the form $\{t : t\restriction_{\omega^{\leq d}} = r\}$ for finite forests r and a number (depth) d.

Let X be an uncountable Polish topological space. The class of open sets in X is denoted as $\mathbf{\Sigma}^0_1(X)$. The class of complements of open sets (called closed) is denoted as $\mathbf{\Pi}^0_1(X)$. The Borel hierarchy is defined inductively, the building ingredients are countable unions and intersections. For a countable ordinal α let:

- $\mathbf{\Sigma}^0_\alpha(X)$ be the class of countable unions of sets from $\bigcup_{\beta<\alpha} \mathbf{\Pi}^0_\beta(X)$,
- $\mathbf{\Pi}^0_\alpha(X)$ be the class of countable intersections of sets from $\bigcup_{\beta<\alpha} \mathbf{\Sigma}^0_\beta(X)$.

The class of Borel sets is the union of all classes $\mathbf{\Sigma}^0_\alpha$ and $\mathbf{\Pi}^0_\alpha$ for $\alpha < \omega_1$. A more detailed introduction to the Borel hierarchy can be found e.g. in [11, Chapter II]. If the space is clear from the context we will omit it and write just $\mathbf{\Sigma}^0_\alpha$ and $\mathbf{\Pi}^0_\alpha$.

The class of Borel sets is not closed under projection. Each set that is a projection of a Borel set is called *analytic*. The class of analytic sets is denoted by $\mathbf{\Sigma}^1_1$. The superscript 1 means that the class is a part of the projective hierarchy. The rest of the projective hierarchy is defined as follows:

- $\mathbf{\Pi}^1_i$ consists of the complements of the sets from $\mathbf{\Sigma}^1_i$,
- $\mathbf{\Sigma}^1_{i+1}$ consists of the projections of the sets from $\mathbf{\Pi}^1_i$.

The sets from the class $\mathbf{\Pi}^1_1$ are called *co-analytic*.

The Borel hierarchy together with the projective hierarchy constitute the so-called *boldface hierarchy*. The most important property of this hierarchy is strictness: all the inclusions on the following diagram are strict.

$$\begin{tikzcd}
\mathbf{\Sigma}^0_1 \arrow[ddr, no head] & \mathbf{\Sigma}^0_2 \arrow[ddr, no head] & \mathbf{\Sigma}^0_3 & & \mathbf{\Sigma}^0_\omega \arrow[ddr, no head] & \mathbf{\Sigma}^0_{\omega+1} & & \mathbf{\Sigma}^0_{2\omega} & & \mathbf{\Sigma}^1_1 \arrow[ddr, no head] & \mathbf{\Sigma}^1_2 \arrow[ddr, no head] & \mathbf{\Sigma}^1_3 & \\
& & & \cdots & & & \cdots & & \cdots & & & & \cdots \\
\mathbf{\Pi}^0_1 \arrow[uur, no head] & \mathbf{\Pi}^0_2 \arrow[uur, no head] & \mathbf{\Pi}^0_3 & & \mathbf{\Pi}^0_\omega \arrow[uur, no head] & \mathbf{\Pi}^0_{\omega+1} & & \mathbf{\Pi}^0_{2\omega} & & \mathbf{\Pi}^1_1 \arrow[uur, no head] & \mathbf{\Pi}^1_2 \arrow[uur, no head] & \mathbf{\Pi}^1_3 &
\end{tikzcd}$$

▶ Fact 2. Every regular language of forests is in the intersection of $\mathbf{\Sigma}^1_2$ and $\mathbf{\Pi}^1_2$ (denoted by $\mathbf{\Delta}^1_2$).

The set of thin forests A^{ThinFor} is $\mathbf{\Pi}^1_1(A^{\mathsf{For}})$-complete, thus non-Borel.

2.4 Ranks and skeletons

The crucial tool in our analysis of thin forests is structural induction — we inductively decompose a given forest into *simpler* ones. A measure of complexity of thin forests is called a *rank* — a function that assigns to each thin forest a countable ordinal number. The rank we use, denoted CB-rank (or shortly $\mathrm{rank}^{\mathrm{CB}}$), is based on the Cantor-Bendixson derivative on closed subsets of ω^ω.

Intuitively, a forest t has $\mathrm{rank}^{\mathrm{CB}}$ equal M if t contains M levels of infinite branches:

- The CB-rank of the empty forest is 0,
- The CB-rank of a forest with finitely many branches is 1,
- if s is a prefix of t of rank 1 and for every x that is off s we have $\mathrm{rank}^{\mathrm{CB}}(t\restriction_x) \leq M$, then $\mathrm{rank}^{\mathrm{CB}}(t) \leq M+1$.

The set of forests of CB-rank bounded by a given ordinal η is denoted as $A^{\mathsf{ThinFor}\leq\eta}$.

The second tool used to analyze structural properties of thin forests are *skeletons*. A skeleton can be seen as a witness that a given forest is thin. Moreover, a skeleton of a thin forest t represents a structural decomposition of t.

A subset of nodes $\sigma \subseteq \mathrm{dom}(t)$ of a given forest $t \in A^{\mathsf{For}}$ is a *skeleton of* t if:

- from every set of siblings in t exactly one is in σ,
- on every infinite branch π of the forest t almost all nodes $x \prec \pi$ belong to σ.

Observe that we can identify σ with its characteristic function — a labelling of nodes of t by $\{0,1\}$. Therefore, $\sigma \in \{0,1\}^{\mathsf{For}}$ and we can treat a pair of a forest and a skeleton (t,σ) as an element of $A \times \{0,1\}^{\mathsf{For}}$.

An easy inductive argument shows that a forest t has a skeleton if and only if t is a thin forest. Moreover, for every thin forest t one can define its *canonical skeleton* $\sigma(t)$.

3 Algebra

In this section we define thin-forest algebra. Its operations and axioms are constructed in such a manner that the free object of this algebra is the set of all regular thin forests and regular thin contexts. Thin-forest algebra is a common generalization of both Wilke algebra [22] and forest algebra [6].

A thin-forest algebra is a three-sorted algebra $(H, V_+, V_\square, \mathit{act}, \mathit{in}_l, \mathit{in}_r, \mathit{inf})$. It consists of two monoids H and $V = V_+ \cup V_\square$ (partitioned into a subsemigroup V_+ and a submonoid $V_\square$) along with an operation of left action $\mathit{act}\colon H \times V \to H$ of V on H, two operations $\mathit{in}_l, \mathit{in}_r\colon H \to V_\square$ and an infinite loop operation $\mathit{inf}\colon V_+ \to H$. Instead of writing $\mathit{act}(h,v)$, we write vh (notice a reversal of arguments). Instead of writing $\mathit{inf}(v)$, we write v^∞. We will call H the *horizontal monoid* and V the *vertical monoid*.

The above construction is based on forest algebra (see [6]). In fact we take forest algebra and introduce the new operation inf; this operation corresponds to infinite composition of contexts. However, since infinite composition is defined only for guarded contexts, we are forced to make a distinction between guarded and non-guarded objects, therefore we partition the sort V into two parts V_+ and $V_\Box$ respectively.

3.1 Axioms and free objects

A thin-forest algebra must satisfy the following axioms:

A1. $(H, +, 0)$ is a monoid with operation $+$ and neutral element 0,

A2. $(V, \cdot, \Box)$ is a monoid with operation $\cdot$ and neutral element $\Box$; it contains two disjoint subalgebras: $(V_\Box, \cdot, \Box)$ is a monoid and $(V_+, \cdot)$ is a semigroup,

A3. (action axiom) $(vw)h = v(wh)$ for every $v, w \in V$, $h \in H$,

A4. (insertion axiom) $in_l(h)g = h + g$, $in_r(h)g = g + h$ for every $h, g \in H$,

A5. $(vw)^\infty = v(wv)^\infty$ for $v, w \in V$, excluding the case when $v, w \in V_\Box$,

A6. $(v^n)^\infty = v^\infty$ for $v \in V_+$ and every $n \geq 1$.

Given an alphabet A we define the *free thin-forest algebra* over A, which is denoted by $A^{\mathsf{regThin}\triangle}$, as follows:

1. the horizontal monoid is the set of regular thin forests over A, with the operation of forest concatenation;
2. the vertical monoid is the set of regular thin contexts over A (respectively guarded and non-guarded), with the operation of context composition;
3. the action is the substitution of forests in contexts;
4. the in_l operation takes a regular thin forest and transforms it into a regular thin context with the hole to the right of all the roots in the forest (similarly for in_r but the hole is to the left of the roots);
5. the infinite loop operation takes a regular thin context and transforms it into a regular thin forest by performing infinite composition.

▶ **Theorem 3.** *The algebra $A^{\mathsf{regThin}\triangle}$ is a thin-forest algebra. Moreover it is the free algebra in the class of thin-forest algebras over the generator set $A\Box = \{a\Box : a \in A\}$.*

Since the insertion operations are somewhat cumbersome to use, we will use the operation $+$ to concatenate forests with contexts, meaning $h + v = in_l(h)v$, $v + h = in_r(h)v$.

We note that it is possible to introduce an algebra where the free object would be the set of all thin forests and all thin contexts (not only regular ones). This can be done by generalizing ω-semigroups. However, since regular languages of forests are uniquely described by regular forests which they contain, this more general algebra gives us the same information about the language as thin-forest algebra. See [10] for more details.

3.2 Recognizability by thin-forest algebra and regularity

A *morphism* between two thin-forest algebras is defined in a natural way. A set L of thin forests over an alphabet A is *recognized* by a morphism $\alpha\colon A^{\mathsf{regThin}\triangle} \to (H, V)$ if $L = \alpha^{-1}(I)$ for some $I \subseteq H$.

We will consider terms in the signature of thin-forest algebra with typed variables. Variables can be of type τ_H, τ_V, or τ_{V_+}, which means that a valuation of a term should assign to the variable an element of the sort H, V or V_+ respectively. Similarly a term is of certain type if a valuation of this term results in an element from the corresponding sort.

Two thin forests t, s are *L-equivalent* if for every term σ over the signature of thin-forest algebra of type τ_H of one variable x of type τ_H, either both or none of the forests

$\sigma[x \leftarrow t], \sigma[x \leftarrow s]$ belong to L (note that we evaluate the term σ in the free thin-forest algebra). Similarly we define the L-equivalence of contexts (but now the variable x is of type τ_V).

The relation of L-equivalence is a congruence, and the quotient of $A^{\mathsf{regThin}\triangle}$ with respect to L-equivalence is the *syntactic thin-forest algebra* for L. The *syntactic morphism* of L assigns to every element of $A^{\mathsf{regThin}\triangle}$ its equivalence class in the syntactic thin-forest algebra of L.

▶ **Theorem 4.** *A language of thin forests is recognizable by a finite thin-forest algebra if and only if it is regular. Every regular language of thin forests is recognizable by its syntactic morphism. The syntactic thin-algebra and the syntactic morphism can be effectively calculated, based on a parity automaton.*

Let L be a regular language of thin forests and $\alpha\colon A^{\mathsf{regThin}\triangle} \to (H, V)$ its syntactic morphism. We say that an element $h \in H$ is *the bottom element for* L if $\alpha^{-1}(h) \cap L = \emptyset$ and $vh = h$ for every $v \in V$.

Note that the bottom element is unique, since if h_1 and h_2 are both bottom elements, then $h_1 = (\square + h_2)h_1 = h_1 + h_2 = (h_1 + \square)h_2 = h_2$.

4 Applications of thin-forest algebra

In this section we show how thin-forest algebra can be used to give decidable characterizations of certain properties of languages. Many such characterizations boil down to checking whether the syntactic algebra of a given regular language satisfies a set of identities. An *identity* is a pair of terms (of the same type) in the signature of thin-forest algebra over typed variables. An algebra satisfies an identity if for every valuation the two terms have the same value. We usually assume that the operation $v \mapsto v^\omega$ is a part of the signature. This operation assigns to every $v \in V$ its *idempotent power*, i.e. such a power v^k that satisfies $v^k \cdot v^k = v^k$. For every v there exists a unique idempotent power, since V is a semigroup [16] (the number k is not unique, but the value v^k is).

In the following subsections we show how to decide whether a given regular language of thin forests is commutative, invariant under bisimulation, open in the standard topology, and definable by a formula of the temporal logic EF.

4.1 Commutative languages

The notion of *commutative language* of finite forests is quite natural: it is a language closed under rearranging of siblings. In the case of finite forests, a language is commutative if and only if its syntactic algebra satisfies the identity

$$h + g = g + h \qquad \text{for } g, h \in H. \tag{1}$$

In the case of infinite forests we have more flexibility. We get different "degrees of commutativity" by allowing rearranging of siblings finitely many times, finitely many times on every branch, or arbitrarily many times. We think that the last (unrestricted) definition is the most appealing. However, it is not captured by the identity (1). Consider the language L = "every node has 0 or 2 children and every branch goes left only finite number of times". The language L does satisfy (1), but it is not commutative, as witnessed by two thin forests $a(a0 + a\square)^\infty \in L$, $a(a\square + a0)^\infty \notin L$. The problem with the above example is that we would like to be able not only to rearrange forests, but also to rearrange a forest with a context. This property is expressed by the following identity:

▶ **Theorem 5.** *A regular language of thin forests L is commutative if and only if its syntactic thin-forest algebra satisfies the identity*

$$h + v = v + h \qquad \textit{for } h \in H \textit{ and } v \in V.$$

Identity (1) corresponds to a weaker notion of commutativity, where on every branch we allow only finite number of rearrangements of siblings (see [10]).

4.2 Languages invariant under bisimulation

Two forests t_0 and t_1 are called *bisimilar* if Duplicator wins the following game, which is played by players Spoiler and Duplicator. Spoiler begins the game by choosing some $i \in \{0, 1\}$ and a root node x_i of the forest t_i. Duplicator responds by chosing a root node x_{1-i} of the other forest t_{1-i}, which has the same label (if no such node exists, the game is terminated and Spiler wins). For $i \in \{0, 1\}$, let s_i be the forest obtained by taking the subtree of t_i rooted in x_i and removing the root. If Duplicator did not lose, then a new round of the game is played with the forests being s_0 and s_1. Duplicator wins if infinitely many rounds are played without Spoiler winning.

A language of thin forests L is called *invariant under bisimulation* if for every forests which are bisimilar, either both or none belong to L.

▶ **Theorem 6.** *A regular language of thin forests L is invariant under bisimulation if and only if its syntactic thin-forest algebra satisfies the following identities:*

$$h + v = v + h, \qquad h + h = h, \qquad (w^\infty + w)^\infty = w^\infty \qquad \textit{for } v \in V,\ w \in V_+ \textit{ and } h \in H.$$

4.3 Open languages

In this section we give a characterization of the class of languages that are open in the standard topology on forests (see Section 2.3). An equivalent definition says that a forest language L is open if for every forest $t \in L$ there is a finite prefix of t such that changing nodes outside of the prefix does not affect membership in L. Checking whether a given regular forest language L is open was known to be decidable, our contribution lies in showing that for thin forests it can be done by testing the syntactic morphism of L:

▶ **Theorem 7.** *A regular language of thin forests L is open if and only if its syntactic morphism $\alpha\colon A^{\mathsf{regThin}\triangle} \to (H, V)$ satisfies the following condition for $v \in V_+$ and $h \in H$:*

$$\textit{if} \quad v^\infty \in \alpha(L) \quad \textit{then} \quad v^\omega h \in \alpha(L).$$

The notion of open sets is also applicable to the case of infinite words. It is interesting to note that the above condition also characterizes open languages of infinite words.

Moreover, one can extend the theory of ordered algebras (see [16]) to thin-forest algebras. Then the above condition could be simply stated as $v^\infty \geq v^\omega h$.

4.4 Temporal logic EF

The logic EF is a simple temporal logic which uses only one operator EF, which stands for "Exists Finally". Formulas of the logic EF are defined as follows:

1. every letter a is an EF formula, which is true in trees with root label a,
2. EF formulas admit Boolean operations, including negation,

3. if φ is an EF formula, then $\mathsf{EF}\varphi$ is an EF formula, which is true in trees that have a proper subtree where φ is true.

A tree t satisfies an EF formula φ if φ holds in the root of the tree t. There are some technical difficulties with generalizing this definition to forests, therefore we will only allow Boolean combinations of formulas of the form $\varphi \vee \mathsf{EF}\varphi$ to describe forests (we call them forest EF formulas; a forest t satisfies such a formula if φ holds in any node of t).

A forest language L is *invariant under EF-bisimulation* if for every forests t_0, t_1 which are EF-bisimilar either both or none belong to L. The relation of EF-bisimilarity is similar to the relation of bisimilarity, but in the game Spoiler chooses an arbitrary node x_i of t_i (not necessarily a root), and Duplicator responds with an arbitrary node x_{1-i} of t_{1-i}. Note that if t_1, t_2 are EF-bisimilar and φ is an forest EF formula then $t_1 \models \varphi$ if and only if $t_2 \models \varphi$.

The following theorem (in a version for general infinite forests) was proved in [4]:

▶ **Theorem 8.** *A regular language of thin forests L can be defined by a forest EF formula if and only if*

1. *it is invariant under EF-bisimulation,*
2. *its syntactic thin-forest algebra satisfies the identity* $v^\omega h = (v + v^\omega h)^\infty$ *for* $v \in V_+$ *and* $h \in H$*.*

For forests that are not necessarily thin, we could not find how to express the first condition in terms of identities. We show how to do it in the case of thin forests:

▶ **Theorem 9.** *A regular language of thin forests L is invariant under EF-bisimulation if and only if its syntactic thin-forest algebra satisfies the identities for $v, u \in V$, $w \in V_+$, $h \in H$:*

$$h + v = v + h, \qquad vh = vh + h, \qquad (w + (wv)^\infty)^\infty = (wv)^\infty, \qquad (wvu)^\infty = (wuv)^\infty.$$

5 Descriptive properties

5.1 Automata

First we show that it is possible to recognize regular languages of thin forests using „simple” automata.

▶ **Theorem 10.** *Every regular language of thin forests can be recognized among all forests by a* $(1, 3)$*-automaton.*

The principal idea is to guess a skeleton of a given forest and use nondeterministic Büchi automata on the branches of this skeleton to verify the types in the syntactic algebra.

The following theorem expresses that the collapse from Theorem 10 is the best we can get from the point of view of the alternating index hierarchy (also known as the Rabin-Mostowski hierarchy).

▶ **Theorem 11.** *There exists a regular language of thin forests L that is not recognizable among all forests by any alternating* $(1, 2)$*-automaton nor any alternating* $(0, 1)$*-automaton.*

The following theorem shows that regular languages of thin forests can be recognized by unambiguous automata *relatively* to thin forests. It is especially interesting, since there are regular languages of forests that are not unambiguous, one of the examples is the language „exists a node labelled by the letter a” (see [8]). The following theorem implies that the language of thin forests containing a letter a is unambiguous among thin forests.

▶ **Theorem 12.** *For every regular language of thin forests L there exists a nondeterministic forest automaton $\mathcal{A}$ such that* $\mathrm{L}(\mathcal{A}) \cap A^{\mathsf{ThinFor}} = L$ *and for every thin forest $t \in L$ there exists exactly one accepting run of $\mathcal{A}$ on t.*

The proof is based on a modification of a technique (called algebraic automata) proposed by Marcin Bilkowski [1]. The idea is the following: we construct an automaton $\mathcal{A}$ that guesses a marking τ of nodes of the given forest t by types in the syntactic algebra of L. Then $\mathcal{A}$ runs on top of τ a deterministic top-down automaton verifying the following property:

For every node x and every infinite branch π that goes through x, the type guessed in x is consistent with the guessed types of nodes that are off π and letters of t on π.

5.2 Languages that are WMSO-definable among all forests

In this section we consider a nonstandard approach to restricting the family of all forests to thin ones. In this setting we show that it is decidable whether a given regular language of thin forests is WMSO-definable. The difference between the standard approach and the one used in this section is that we do not implicitly restrict our universe to thin forests.

▶ **Definition 13.** Let L be a regular language of thin forests and φ be a formula of WMSO. We say that φ *defines L among all forests* if $L = \{t \in A^{\mathsf{For}} : \ t \models \varphi\}$.

Note that the class of languages definable in WMSO among all forests is not closed under complement with respect to thin forests: the relative complement of the empty language $\emptyset \subseteq A^{\mathsf{ThinFor}}$ is A^{ThinFor} which is not WMSO-definable among all forests.

The following fact says that even in this restricted setting we can define languages as complicated as in the general case.

▶ Fact 14. The examples of WMSO-definable languages lying arbitrarily high on the finite levels of the Borel hierarchy (see [20]) can be encoded into thin forests in a way WMSO-definable among all forests.

The main result of this section is the following characterization.

▶ **Theorem 15.** *Let L be a regular language of thin forests. The following conditions are equivalent:*

1. *there exists $M \in \mathbb{N}$ such that every forest $t \in L$ satisfies* $\mathrm{rank}^{CB}(t) \leq M$*,*
2. *L is WMSO-definable among all forests,*
3. *L is not $\mathbf{\Pi}^1_1(A^{\mathsf{For}})$-hard,*
4. *the syntactic morphism for L satisfies the following condition:*

$$\begin{aligned}&\textit{if } h = v(w+h)^\infty \textit{ or } h = v(h+w)^\infty \textit{ for some } v \in V, w \in V_+,\\ &\qquad\qquad \textit{then } h \textit{ is the bottom element for } L.\end{aligned} \tag{2}$$

The following list presents a sketch of the argumentation.

From 1 to 2. A direct construction of a formula.

From 2 to 3. Folklore.

From 3 to 4. A pumping argument: a counterexample to the equations can be used to construct a continuous function f from the space of trees over ω to A^{For}. If a given tree t is well-founded (does not contain an infinite branch) then the result $f(t)$ is in L. Otherwise the result $f(t)$ is not thin, therefore does not belong to L. Since the set of well-founded trees over ω is $\mathbf{\Pi}^1_1$-hard then so is L (f is a continuous reduction).

From 4 to 1. Estimating: condition (2) introduces an order on types in H. The height of this order bounds the maximal CB-rank of forests in the language L.

Note that the last condition in the theorem is effective, therefore we obtain the following corollary.

▶ **Corollary 16.** *It is decidable whether a given regular language of thin forests L is WMSO-definable among all forests.*

▶ Proposition 17. Assume that L is a regular language of forests that is recognized by a nondeterministic (or equivalently alternating) $(1, 2)$-automaton. Assume additionally that L contains only thin forests. Then L can be defined in WMSO among all forests.

Proof. Since L is recognizable by a $(1, 2)$-automaton so L is an analytic subset of A^{For}. Therefore, L cannot be $\mathbf{\Pi}^1_1$-hard, thus L satisfies the condition 3 in Theorem 15. ◀

5.3 Topological properties

In this section we give a couple of results showing that regular languages of thin forests are topologically simpler then generic regular languages of forests.

▶ **Theorem 18.** *Every regular language of thin forests L is co-analytic as a set of forests.*

Note that despite the fact that the space of thin forests A^{ThinFor} is co-analytic among all forests, it contains arbitrarily complicated subsets. In fact, already the family of forests of CB-rank equal 1 is an uncountable Polish topological space, so the whole boldface hierarchy (see Section 2.3) can be constructed using only such forests.

Theorems 15 and 18 imply the following dichotomy or *gap property* in the spirit of [15].

▶ Remark. For every regular language of thin forests L exactly one of the following possibilities holds, it can be effectively decided which one:

- L is WMSO-definable among all forests and lies on a finite level of the Borel hierarchy,
- L is $\mathbf{\Pi}^1_1(A^{\mathsf{For}})$-complete.

The following theorem shows that, when treating thin forests as our universe, there are no topologically hard regular languages.

▶ **Theorem 19.** *Let X be a Polish topological space, $f \colon X \to A^{\mathsf{ThinFor}}$ be continuous and L be a regular language of thin forests. Then $f^{-1}(L)$ is Borel in X.*

The following theorem can be seen as complementing Theorem 19.

▶ **Theorem 20.** *There exists a regular language of thin forests L_W over an alphabet A_W that is* Borel-hard*: for every Polish topological space X and every Borel set $B \subseteq X$ there exists a continuous function $f \colon X \to {A_W}^{\mathsf{ThinFor}}$ such that $f^{-1}(L_W) = B$.*

The principal concept of the above language is based on a construction proposed in [9]. Using the structure of the language L_W one can deduce the following corollary.

▶ **Corollary 21.** *The language L_W cannot be defined in WMSO among thin forests.*

This statement holds true even if we provide with every forest $t \in {A_W}^{\mathsf{ThinFor}}$ its canonical skeleton $\sigma(t)$: there is no WMSO formula φ over the alphabet $A_W \times \{0, 1\}$ such that

$$L_W = \left\{ t \in {A_W}^{\mathsf{ThinFor}} : \ (t, \sigma(t)) \models \varphi \right\}.$$

Acknowledgements

The authors would like to thank Henryk Michalewski for posing a number of motivating problems and questions on the subject. Additionally, the authors thank the referees for suggestions and comments.

References

1 M. Bilkowski. Algebraic automata. Private communication, 2011.

2 A. Blumensath. Recognisability for algebras of infinite trees. *Theor. Comput. Sci.*, 412(29):3463–3486, 2011.

3 M. Bojańczyk. Effective characterizations of tree logics. In *PODS*, pages 53–66, 2008.

4 M. Bojańczyk and T. Idziaszek. Algebra for infinite forests with an application to the temporal logic EF. In *CONCUR*, pages 131–145, 2009.

5 M. Bojańczyk and T. Place. Regular languages of infinite trees that are boolean combinations of open sets. In *ICALP*, pages 104–115, 2012.

6 M. Bojańczyk and I. Walukiewicz. Forest algebras. In *Logic and Automata*, pages 107–132, 2008.

7 J.R. Büchi. On a decision method in restricted second-order arithmetic. In *Proc. 1960 Int. Congr. for Logic, Methodology and Philosophy of Science*, pages 1–11, 1962.

8 A. Carayol, Ch. Löding, D. Niwiński, and I. Walukiewicz. Choice functions and well-orderings over the infinite binary tree. *CEJM*, 8:662–682, 2010.

9 S. Hummel, H. Michalewski, and D. Niwiński. On the Borel inseparability of game tree languages. In *STACS*, pages 565–575, 2009.

10 T. Idziaszek. *Algebraic methods in the theory of infinite trees.* PhD thesis, University of Warsaw, 2013. Unpublished.

11 A. Kechris. *Classical descriptive set theory.* Springer-Verlag, New York, 1995.

12 M. Kufleitner and A. Lauser. Languages of dot-depth one over infinite words. In *LICS*, pages 23–32, 2011.

13 S. Lifsches and S. Shelah. Uniformization and skolem functions in the class of trees. *J. Symb. Log.*, 63(1):103–127, 1998.

14 F. Murlak. The Wadge hierarchy of deterministic tree languages. *LMCS*, 4(4), 2008.

15 D. Niwiński and I. Walukiewicz. A gap property of deterministic tree languages. *TCS*, 1(303):215–231, 2003.

16 D. Perrin and J.-É. Pin. *Infinite Words: Automata, Semigroups, Logic and Games.* Elsevier, 2004.

17 M.O. Rabin. Decidability of second-order theories and automata on infinite trees. *Bull. Amer. Math. Soc.*, 74:1025–1029, 1968.

18 M.P. Schützenberger. On finite monoids having only trivial subgroups. *Inf. and Cont.*, 8(2):190–194, 1965.

19 I. Simon. Piecewise testable events. In *Automata Theory and Formal Languages*, pages 214–222, 1975.

20 J. Skurczyński. The Borel hierarchy is infinite in the class of regular sets of trees. *TCS*, 112(2):413–418, 1993.

21 D. Thérien and T. Wilke. Over words, two variables are as powerful as one quantifier alternation. In *STOC*, pages 234–240, 1998.

22 T. Wilke. Classifying discrete temporal properties. Habilitationsschrift, Universitat Kiel, apr. 1998.

23 Thomas Wilke. An algebraic theory for regular languages of finite and infinite words. *Int. J. Alg. Comput.*, 3:447–489, 1993.

Approximate comparison of distance automata*

Thomas Colcombet and Laure Daviaud

Université Sorbonne Paris Cité, CNRS, LIAFA

Abstract

Distance automata are automata weighted over the semiring $(\mathbb{N} \cup \{\infty\}, \min, +)$ (the tropical semiring). Such automata compute functions from words to $\mathbb{N} \cup \{\infty\}$ such as the number of occurrences of a given letter. It is known that testing $f \leqslant g$ is an undecidable problem for f, g computed by distance automata. The main contribution of this paper is to show that an approximation of this problem becomes decidable.

We present an algorithm which, given $\varepsilon > 0$ and two functions f, g computed by distance automata, answers "yes" if $f \leqslant (1-\varepsilon)g$, "no" if $f \not\leqslant g$, and may answer "yes" or "no" in all other cases. This result highly refines previously known decidability results of the same type.

The core argument behind this quasi-decision procedure is an algorithm which is able to provide an approximated finite presentation to the closure under products of sets of matrices over the tropical semiring.

We also provide another theorem, of affine domination, which shows that previously known decision procedures for cost-automata have an improved precision when used over distance automata.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases Distance automata, tropical semiring, decidability, cost functions

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.574

1 Introduction

One way to see language theory, and in particular the theory of regular languages, is as a toolbox of constructions and decision procedures allowing high level handling of languages. These high level operations can then be used as black-boxes in various decision procedures, such as in verification. Since the early times of automata theory, the need for the effective handling of functions rather than sets (as languages) was already apparent. Schützenberger proposed already in the sixties models of finite state machines used for computing functions. These are now known as weighted automata [11] and are the subject of much attention from the research community. In general, weighted automata are non-deterministic automata, weighted over some semiring $(S, \oplus, \otimes)$. The value computed by such an automaton over a given word is then the sum (for $\oplus$) over every run over this word of the product (for $\otimes$) of the weights along the run.

Several instances of this model are very relevant for modelling the behaviour of systems, and henceforth attract much attention. This is in particular the case of probabilistic automata (over the semiring $(\mathbb{R}^+, +, \times)$ with some additional constraints enforcing weights to remain in $[0, 1]$), and distance automata which are automata weighted over the semiring $(\mathbb{N} \cup \{\infty\}, \min, +)$. In such an automaton, each transition is labelled with a non-negative integer (usually 0 or 1), and the weight of a word is the minimum over all possible paths of

* The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n°259454.

the sum of the weights. These automata naturally capture some optimisation problems since computing the value amounts to find the path of minimal weight.

The subject of this paper is to develop algorithmic tools for distance automata, and more precisely to develop the question of comparing distance automata. We know from the beginning that exact comparison is beyond reach.

▶ **Theorem 1** (Krob [7]). *The problem to determine, given two functions f, g computed by distance automata, whether $f = g$ or not is undecidable. The problem whether $f \leqslant g$ or not is also undecidable, even if g is deterministic.*

Despite this, some positive results exist but for a comparison relation less precise than inequality, namely *domination*. Given two functions $A^* \to \mathbb{N} \cup \{\infty\}$, *$f$ is dominated* by g (and we note $f \preccurlyeq g$) if there is a function $\alpha : \mathbb{N} \to \mathbb{N}$, extended with $\alpha(\infty) = \infty$, such that

$$f \leqslant \alpha \circ g \ .$$

Moreover, if α is a polynomial, we say that *f is polynomially dominated by g*. The following theorem shows the good properties of the domination relation.

▶ **Theorem 2** ([2] extending results and techniques from [4, 9, 13, 6, 1]). *Given two functions computed by distance automata, domination is decidable. Furthermore, if a function dominates another, then it polynomially dominates it*[1].

The motivation of this work is to improve Theorem 2 and to answer the following question:

Is it possible to decide “approximations” of the inequality of functions computed by distance automata that are finer than domination ?

We answer positively this question in two ways. We first show:

▶ **Theorem 3** (affine domination). *Given two functions f and g computed by distance automata, if f is dominated by g then f is affinely dominated by g, i.e., $f \leqslant \alpha \circ g$ for some polynomial α of degree* 1.

A consequence of this theorem is that the decision procedure provided by Theorem 2 in fact decides the affine domination, which is finer than the polynomial domination[2].

Our second, and main contribution is an even more accurate decision-like procedure. One says that an algorithm, given two functions f and g and some real $\varepsilon > 0$, *ε-approximates the inequality* if:

- if $f \leqslant (1-\varepsilon)g$, the output is “yes”,
- if $f \nleqslant g$, the output is “no”,
- otherwise the output can be either “yes” or “no”.

Hence, if such an algorithm answers “yes”, one has a guaranty that $f \leqslant g$. Conversely if f is ε-inferior to g (meaning $f \leqslant (1-\varepsilon)g$), one is sure that the algorithm answers “yes”. Our second and main result reads as follows:

▶ **Theorem 4** (approximate comparison). *There is an EXPSPACE algorithm which ε-approximates the inequality of functions computed by distance automata.*

---

[1] Technically, this is not stated in [2] , but can be derived directly from the proofs which explicitly compute the function α using operations preserving polynomials.

[2] Theorem 2 holds for more general classes of automata, cost automata, for which affine domination does not hold. Affine domination is specific to distance automata.

This result is in fact a consequence of a theorem – called the core theorem below – stating that it is possible, given a set of matrices X in the tropical semiring, to approximate (in a suitable way) the set

$$\left\{\frac{1}{\ell}(M_1 \otimes \cdots \otimes M_\ell) \;:\; M_1,\ldots,M_\ell \in X\right\} ,$$

where $\otimes$ denotes the product of matrices. More precisely, the core theorem states that it is possible to approximate the upper envelope of the set of pairs

$$\{(M_1 \otimes \cdots \otimes M_\ell, \ell) \;:\; M_1,\ldots,M_\ell \in X\}$$

for a suitable notion of approximation. This core theorem, Theorem 6, will be described precisely in the first section of this paper.

In Section 2 we present some classical definitions and formally state our core theorem. Section 3 is devoted to the proof of the core theorem. Section 4 applies the core theorem for answering our original motivation, and shows the decidability of the approximate comparison between distance automata. We prove on the way our result of affine domination, Theorem 3. Section 5 concludes the paper.

2 Description of the core theorem

In this first section, we introduce the basic definitions, and define sufficient material for stating our core theorem 6. Its proof is the subject of Section 3 and its application to the comparison of distance automata is the subject of Section 4. We first introduce some classical algebraic definitions in Section 2.1, and finally state our core theorem in Section 2.2.

2.1 Classical definitions

A *semigroup* $(S,\cdot)$ is a set S equipped with an associative binary operation "$\cdot$". If the product has furthermore a neutral element, it is called a *monoid.* The monoid is said *commutative* when $\cdot$ is commutative. An *idempotent* in a monoid is an element e such that $e \cdot e = e$. Given a subset A of a semigroup, $\langle A\rangle$ denotes the closure of A under product, *i.e.,* the least sub-semigroup that contains A. Given two subsets X, Y of a semigroup, $X \cdot Y$ denotes the set $\{a \cdot b \;:\; a \in X,\; b \in Y\}$.

A *semiring* is a set S equipped with two binary operations $\oplus$ and $\otimes$ such that $(S,\oplus)$ is a commutative monoid of neutral element 0, $(S,\otimes)$ is a monoid of neutral element 1, 0 is absorbing for $\otimes$ (i.e., $x \otimes 0 = 0 \otimes x = 0$) and $\otimes$ distributes over $\oplus$. We will consider three semirings: $(\mathbb{R}^+ \cup \{\infty\}, \min, +)$, denoted $\overline{\mathbb{R}^+}$, its restriction to $\mathbb{N} \cup \{\infty\}$, denoted $\overline{\mathbb{N}}$, and its restriction to $\{0,\infty\}$ denoted $\mathbb{B}$. The third, finite semiring is called the *Boolean semiring*, since if we identify 0 with "true" and ∞ with "false", then $\oplus$ is the disjunction and $\otimes$ the conjunction. Remark that in the three cases, the "0" is ∞, and the "1" is 0.

Let S be $\overline{\mathbb{R}^+}$, $\overline{\mathbb{N}}$ or $\mathbb{B}$. The set of matrices with m rows and n columns over S is denoted $\mathcal{M}_{m,n}(S)$. For $M \in \mathcal{M}_{m,n}(S)$, we denote by $\widetilde{M}$ the matrix over $\mathbb{B}$ in which all entries of M different from ∞ are changed into 0. We define the multiplication $A \otimes B$ of two matrices A, B (provided the number n of columns of A equals the number of rows of B) as usual by:

$$(A \otimes B)_{i,j} = \bigoplus_{0<k\leqslant n} (A_{i,k} \otimes B_{k,j}) = \min_{0<k\leqslant n} (A_{i,k} + B_{k,j}) .$$

For a positive integer k, we also use the notation $M^k = \underbrace{M \otimes \cdots \otimes M}_{k \text{ times}}$.

For $\lambda \in S$, we denote by λA the matrix such that $(\lambda A)_{i,j} = \lambda A_{i,j}$, with the convention $\lambda\infty = \infty$ (the standard product is used here, not the one of the semiring). We denote also by $B + \lambda$ the matrix such that $(B + \lambda)_{i,j} = B_{i,j} + \lambda$. Finally, we write $A \leqslant B$ if for all i, j, $A_{i,j} \leqslant B_{i,j}$.

2.2 Weighted matrices and the core theorem

In this section we state our core approximation result, Theorem 6. This theorem states that given a set of weighted matrices, it is possible to compute a finite presentation of its closure under product up to some approximation. Hence we have to introduce weighted matrices, the approximation, and what are finite presentations before disclosing the statement. This requires some specific definitions that we present beforehand. We fix now a positive integer n, and all matrices implicitly belong to $\mathcal{M}_{n,n}(\overline{\mathbb{R}^+})$.

As already mentioned in the introduction, our goal is to approximate a set of pairs (M, ℓ) where M is a matrix and ℓ is a positive integer. We call such pairs weighted matrices. A *weighted matrix* is an ordered pair (M, ℓ) where $M \in \mathcal{M}_{n,n}(\overline{\mathbb{R}^+})$ and ℓ is a positive integer. The positive integer ℓ is called the *weight* of the weighted matrix. The set of weighted matrices is denoted by $\mathcal{W}_{n,n}$. Weighted matrices have a semigroup structure $(\mathcal{W}_{n,n}, \otimes)$, where $(M, \ell) \otimes (M', \ell')$ stands for $(M \otimes M', \ell + \ell')$. Given A, B subsets of $\mathcal{W}_{n,n}$, one denotes by $A \otimes B$ the set $\{M \otimes N \ : \ M \in A,\ N \in B\}$, and by $\langle A\rangle$ the closure under $\otimes$ of A. With this terminology, our goal is, given a finite set of weighted matrices X, to approximate $\langle X\rangle$. (Intuitively, if (M, ℓ) is a weighted matrix, M represents the behaviour of a distance automaton that computes a function f, over a word w, while ℓ stands for the lengths of w. So, weighted matrices let us compare $f(w)$ with $|w|$ which is exactly what we want. The operation $\otimes$ between two weighted matrices matches with the concatenation of words, i.e. the product in the tropical semiring for matrices and the sum for lengths.)

We describe now the notion of approximation that we use. Given some $\varepsilon > 0$ and two weighted matrices (M, ℓ) and (M', ℓ'), one writes

$$(M, \ell) \preccurlyeq_\varepsilon (M', \ell') \qquad \text{if} \qquad \ell \geqslant \ell', \widetilde{M} = \widetilde{M'} \text{ and } M \leqslant M' + \varepsilon\ell\ .$$

Remark that in particular, this implies $\frac{1}{\ell}M \leqslant \frac{1}{\ell'}M' + \varepsilon$, which is the intention behind this definition. The definition of $\preccurlyeq_\varepsilon$ is more constraining: this is mandatory for having better properties with respect to the product of matrices, such as in Lemma 5 below. This definition extends to sets of weighted matrices as follows. Given two such sets X, X', $X \preccurlyeq_\varepsilon X'$ if for all $(M, \ell) \in X$, there exists $(M', \ell') \in X'$ such that $(M, \ell) \preccurlyeq_\varepsilon (M', \ell')$. One writes $X \approx_\varepsilon X'$ if $X \preccurlyeq_\varepsilon X'$ and $X' \preccurlyeq_\varepsilon X$ (and says X is ε-equivalent to X').

The following lemma establishes some simple properties of the $\preccurlyeq_\varepsilon$ relations (as a consequence, the same properties hold for $\approx_\varepsilon$).

▶ **Lemma 5.** *Given $X, X', Y, Y', Z \subseteq \mathcal{W}_{n,n}$ and $\varepsilon, \eta > 0$,*
- *if $X \preccurlyeq_\varepsilon Y$ and $Y \preccurlyeq_\eta Z$ then $X \preccurlyeq_{\varepsilon+\eta} Z$,*
- *if $X \preccurlyeq_\varepsilon X'$ and $Y \preccurlyeq_\varepsilon Y'$ then $X \otimes Y \preccurlyeq_\varepsilon X' \otimes Y'$,*
- *if $X \preccurlyeq_\varepsilon X'$ then $\langle X\rangle \preccurlyeq_\varepsilon \langle X'\rangle$.*

Proof. *First item.* If $(M, \ell) \preccurlyeq_\varepsilon (M', \ell') \preccurlyeq_\eta (M'', \ell'')$, then $\ell \geqslant \ell' \geqslant \ell''$, $\widetilde{M} = \widetilde{M'} = \widetilde{M''}$ and $M \leqslant M' + \varepsilon\ell \leqslant M'' + \eta\ell' + \varepsilon\ell \leqslant M'' + (\varepsilon + \eta)\ell$. This easily extends to sets of weighted matrices.

Second item. Assume $(M, \ell) \preccurlyeq_\varepsilon (M', \ell')$ and $(N, t) \preccurlyeq_\varepsilon (N', t')$. Then, $\ell + \ell' \geqslant t + t'$, $\widetilde{M \otimes N} = \widetilde{M' \otimes N'}$ and $M \otimes N \leqslant (M' + \varepsilon\ell) \otimes (N' + \varepsilon t) \leqslant M' \otimes N' + \varepsilon(\ell + t)$. This naturally extends to sets of weighted matrices.

Third item. By induction, applying the second item. ◀

The last ingredient required is to describe how to represent (infinite) sets of weighted matrices. Call a set of weighted matrices $W \subseteq \mathcal{W}_{n,n}$ *finitely presented* if it is a finite union of singleton sets, and of sets of the form $\{(kM, k) : \ k \geqslant \ell\}$ where $M \in \mathcal{M}_{n,n}(\overline{\mathbb{R}^+})$ and ℓ is a positive integer. Our algorithm manipulates finitely presented sets of weighted matrices.

The core technical contribution of this paper can now be stated, as follows.

▶ **Theorem 6** (core theorem). *Given $X \subseteq \mathcal{W}_{n,n}$ finitely presented and $\varepsilon > 0$, one can compute effectively $Y \subseteq \mathcal{W}_{n,n}$ finitely presented such that:*

$$Y \approx_\varepsilon \langle X \rangle \ .$$

A sketch of the proof of this result will be the subject of Section 3. The application of this theorem to the comparison of distance automata is presented in Section 4. The two sections are independent.

3 Proof of the core theorem

In this section we describe the key arguments involved in the proof of Theorem 6. It is the combination of several arguments. The first one is the use of the forest factorisation theorem of Simon.

3.1 The main induction: the forest factorization theorem of Simon

The forest factorization theorem of Simon [12] is a powerful combinatorial tool for understanding the structure of finite semigroups. In this short abstract, we will not describe the original statement of this theorem, in terms of trees of factorisations, but rather a direct consequence of it which is central in our proof.

▶ **Theorem 7** (equivalent to the forest factorization theorem [12][3]). *Given a semigroup morphism φ from $(S, \otimes)$ (possibly infinite) to a finite semigroup $(T, \cdot)$, and some $X \subseteq S$, set $X_0 = X$ and for all $k \geqslant 0$,*

$$X_{k+1} = X_k \cup X_k \otimes X_k \cup \bigcup_{e \text{ is idempotent } \in T} \langle X_k \cap \varphi^{-1}(e) \rangle \ ,$$

then $\langle X \rangle = X_N$ for $N = 3|T| - 1$.

This proposition teaches us that, for computing the closure under product in the semigroup S, it is sufficient to know how to compute (a) the union of sets, (b) the product of sets, and (c) the restriction of a set to the inverse image of an idempotent by φ, and (d) the closure under product of sets of elements that all have the same idempotent image under φ. Of course, this proposition is interesting when the semigroup T is cleverly chosen.

In our case, we are going to use the above proposition with $(S, \otimes) = \mathcal{W}_{n,n}$, $(T, \cdot) = \mathcal{M}_{n,n}(\mathbb{B})$, and φ the morphism which maps (M, ℓ) to $\widetilde{M}$. Our algorithm will compute, given a finitely presented set of weighted matrices X, an approximation of $\langle X \rangle$ following the same inductive construction as in the forest factorisation theorem. This is justified by the two following lemmas, for which we provide the sketch of a proof.

[3] Modern proofs of this theorem can be found in [8, 3], in particular with the exact bound of $N = 3|T| - 1$ (Simon's original proof only provides $N = 9|T|$).

▶ **Lemma 8.** *For all $\varepsilon > 0$ and all finitely presented sets $X, Y \subseteq \mathcal{W}_{n,n}$ there exists effectively a finitely presented set* $\mathtt{product}(\varepsilon, X, Y) \subseteq \mathcal{W}_{n,n}$ *such that*

$$\mathtt{product}(\varepsilon, X, Y) \approx_\varepsilon X \otimes Y \ .$$

Given a set X of weighted matrices, let us set $\widetilde{X} = \{\widetilde{M} \mid (M, \ell) \in X\}$.

▶ **Lemma 9.** *For all $\varepsilon > 0$ and all finitely presented set $X \subseteq \mathcal{W}_{n,n}$ such that $\widetilde{X} = \{e\}$ for an idempotent e, there exists effectively a finitely presented set* $\mathtt{idempotent}(\varepsilon, X) \subseteq \mathcal{W}_{n,n}$ *such that*

$$\mathtt{idempotent}(\varepsilon, X) \approx_\varepsilon \langle X \rangle \ .$$

Assuming that Lemmas 8 and 9 hold, it is easy to provide an algorithm which, given $X \subseteq \mathcal{W}_{n,n}$ finitely presented, computes $X' \subseteq \mathcal{W}_{n,n}$ finitely presented such that $X' \approx_\varepsilon \langle X \rangle$. The principle of the algorithm is to implement Theorem 7, using finitely presented sets that approximate the X_k's.

- Set $Y_0 = X$ and $N = 3(2^{n^2}) - 1 = 3|\mathcal{M}_{n,n}(\mathbb{B})| - 1$.
- For all $0 \leqslant k \leqslant N$, set $\varepsilon(k) = \frac{\varepsilon}{2^{N-k}}$ and

$$Y_{k+1} = Y_k \cup \mathtt{product}(\varepsilon(k), Y_k, Y_k) \cup \bigcup_{e \otimes e = e \in \mathcal{M}_{n,n}(\mathbb{B})} \mathtt{idempotent}(\varepsilon(k), Y_k \cap \varphi^{-1}(e)) \ .$$

- output Y_N

Correction can be justified as follows: one proves by induction that $Y_k \approx_{\varepsilon(k)} X_k$ for all $k = 0, \ldots, N$ where X_k is defined as in Theorem 7 (with $S = \mathcal{W}_{n,n}$, $T = \mathcal{M}_{n,n}(\mathbb{B})$ and $\varphi(M, \ell) = \widetilde{M}$). For $k = 0$, one has $X_k = X = Y_k$. Let $k \geqslant 0$, suppose that $Y_k \approx_{\varepsilon(k)} X_k$, then by Lemma 8, Lemma 5 and the induction hypothesis,

$$\mathtt{product}(\varepsilon(k), Y_k, Y_k) \approx_{\varepsilon(k)} Y_k \otimes Y_k \approx_{\varepsilon(k)} X_k \otimes X_k \ .$$

Finally, by Lemma 5, $\mathtt{product}(\varepsilon(k), Y_k, Y_k) \approx_{2\varepsilon(k)} X_k \otimes X_k$. Similarly, by Lemma 9, for all idempotent e, $\mathtt{idempotent}(\varepsilon(k), Y_k \cap \varphi^{-1}(e)) \approx_{2\varepsilon(k)} \langle X_k \cap \varphi^{-1}(e) \rangle$. Thus $Y_{k+1} \approx_{\varepsilon(k+1)} X_{k+1}$.

Hence, what remains to be done is to establish Lemmas 8 and 9.

3.2 Approximate products of sets

In this part, we give the main ideas of the proof of Lemma 8. It shows explicit examples of the approximation arguments that are used in a more advanced way for proving Lemma 9.

Proof of Lemma 8. Since the finitely presented sets of weighted matrices are closed under union, it is sufficient to prove Lemma 8 for the atomic blocks of the finite presentation. Namely, it is sufficient to consider the case $X = \{(M, x)\}$ or $X = \{(\ell M, \ell) \mid \ell \geqslant x\}$ together with $Y = \{(N, y)\}$ or $Y = \{(\ell N, \ell) \mid \ell \geqslant y\}$. This results in four possibilities, among which only three remain up to symmetry: (a) $X = \{(M, x)\}$ and $Y = \{(N, y)\}$, (b) $X = \{(M, x)\}$ and $Y = \{(\ell N, \ell) \mid \ell \geqslant y\}$, and finally (c) $X = \{(\ell M, \ell) \mid \ell \geqslant x\}$ and $Y = \{(\ell N, \ell) \mid \ell \geqslant y\}$.

Let us explain the most interesting case, case (c). Let a be the maximum absolute value of a non-infinite entry of M or N. Choose some z such that $2ax \leqslant \varepsilon z$ and $2ay \leqslant \varepsilon z$, and let Z be the set $Z_1 \cup Z_2$ defined by:

$$Z_1 = \{(x'M \otimes y'N, x' + y') \mid x' + y' < z\} \ ,$$

and $$Z_2 = \{(\ell(\lambda M \otimes (1 - \lambda)N), \ell) \mid \ell \geqslant z,\ \lambda \in [0, 1]\} \ .$$

The set Z_1 is finite, and merely lists all weighted matrices of weight less than z in $X \otimes Y$. The set Z_2 (which is not finitely presented) takes all barycentres of M and N, and produces corresponding weighted matrices for all possible weights greater or equal to z. We need to prove two things. First that $Z \approx_{\frac{\varepsilon}{2}} X \otimes Y$, and second that one can further approximate Z_2 by a finitely presented set $Z_3 \approx_{\frac{\varepsilon}{2}} Z_2$. By Lemma 5 we can then conclude that $X \otimes Y \approx_{\varepsilon} Z_1 \cup Z_3$, and that $Z_1 \cup Z_3$ is finitely presented and computable from X and Y.

Let us prove that $Z \approx_{\frac{\varepsilon}{2}} X \otimes Y$. Remark first that $X \otimes Y \subseteq Z$. For the converse direction, consider $(W, \ell) \in Z$. Clearly, if $\ell < z$, then $(W, \ell) \in Z_1 \subseteq X \otimes Y$. Otherwise, $W = (\lambda \ell M) \otimes ((1 - \lambda)\ell N)$. It is sufficient for us to find $x' \geqslant x$ and $y' \geqslant y$ such that $x' + y' = \ell$, and $\left|\lambda - \frac{x'}{\ell}\right| \leqslant \frac{\varepsilon}{2a}$: indeed, assuming the existence of such x', y', the matrix $W' = (x'M \otimes y'N, \ell)$ is such that $(W, \ell) \approx_{\frac{\varepsilon}{2}} (W', \ell)$, and furthermore $(W', \ell) \in X \otimes Y$. For proving the existence of such x', y', consider the evolution of the value $\frac{x'}{\ell}$ when x' ranges from x to $\ell - y$. Since $\ell \geqslant z$, $\frac{x}{\ell} \leqslant \frac{\varepsilon}{2a}$, and similarly $\frac{\ell - y}{\ell} \geqslant 1 - \frac{\varepsilon}{2a}$. Furthermore when x' increases of 1, the quantity $\frac{x'}{\ell}$ increases of at most $\frac{1}{z} \leqslant \frac{\varepsilon}{2a}$. As a consequence, $\frac{x'}{\ell}$ gets to be $\frac{\varepsilon}{2a}$-close of any $\lambda \in [0, 1]$ when x' ranges from x to $\ell - y$. Consider x' witnessing this fact and set $y' = \ell - x'$. The pair x', y' satisfies the requirement.

One now needs defining a set $Z_3 \approx_{\frac{\varepsilon}{2}} Z_2$ which is finitely presented. The set Z_3 is defined as the set Z_2, but for the fact that λ is discretized by steps of $\frac{\varepsilon}{4a}$. This can be written as:

$$Z_3 = \bigcup_{\lambda \in \left([0,1] \cap \frac{\varepsilon}{4a}\mathbb{N}\right)} \{(\ell(\lambda M \otimes (1 - \lambda)N), \ell) \mid \ell \geqslant z\} \ .$$

Clearly, this set is finitely presented. It is also simple to prove that $Z_3 \approx_{\frac{\varepsilon}{2}} Z_2$. ◀

4 Comparing distance automata

In this section, we consider the problem of comparing the functions computed by distance automata. In particular, we establish Theorem 3, and we reduce Theorem 4 to our core theorem, Theorem 6. We start by describing distance automata, and their relationship with matrices over the tropical semiring (Section 4.1).

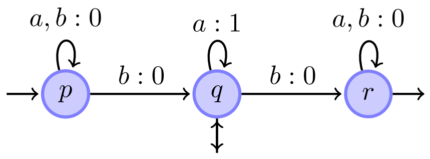
4.1 Distance automata

An *alphabet* is a finite set of symbols. The set of *words* over an alphabet $\mathbb{A}$ is denoted $\mathbb{A}^*$. A *distance automaton* is a tuple $(\mathbb{A}, Q, I, F, T)$, where Q is a finite set of *states* (that we can assume to be $\{1, \ldots, n\}$) where I (resp. T) is a row-vector (*resp.* column-vector) indexed by Q, and F is a morphism from words to $\mathcal{M}_{n,n}(\overline{\mathbb{N}})$. The function f computed by a distance automaton $(\mathbb{A}, Q, I, F, T)$ over an input word u is:

$$\begin{aligned} f : \mathbb{A}^* &\to \overline{\mathbb{N}} \\ u &\mapsto I \otimes F(u) \otimes T \ . \end{aligned}$$

We assume from now on that the initial and final vectors I, T of distance automata only range over $\{0, \infty\}$. The theorems are equally true without this assumption, but this simplifies slightly the proof. In practice the theorems without this restriction can be obtained by simple reductions to this case.

We have defined so far distance automata in terms of matrices. One can see this object in a more "automaton" form as follows. There is a transition labelled (a, x) from state p to state q if $x < \infty$ and $x = F(a)_{p,q}$. A state p is *initial* if $I_{1,p} = 0$. It is *final* if $T_{i,1} = 0$. An example of distance automaton is as follows:



One can redefine the function computed by a distance automaton as follows. A *run* of an automaton over a word $a_1 \dots a_k$ is a sequence $p_0, \dots, p_k$ of states. The *weight of a run* is the sum of the weights of its transitions, *i.e.*, $F(a_1)_{p_0,p_1} + \cdots + F(a_k)_{p_{k-1},p_k}$. Remark that if there is some non-existing transition in this sequence, say from p_{i-1} to p_i, this means that $F(a_i)_{p_{i-1},p_i} = \infty$, and as a consequence the run has an infinite weight. A run is *accepting* if p_0 is initial and p_k is final. One defines the function accepted by the automaton as:

$$\begin{aligned} f : \mathbb{A}^* &\to \overline{\mathbb{N}} \\ u &\mapsto \inf\{\text{weight}(\rho) \ : \ \rho \text{ accepting run over } u\} \ . \end{aligned}$$

This definition is equivalent to the matrix version presented above.

For instance, the function computed by the above automaton associates to each word $u = a^{n_0}ba^{n_1} \dots ba^{n_k}$ the value $\min(n_0, \dots, n_k)$.

4.2 Superior limits

In this section, we present Theorem 10 which allows us to compute the superior limit of some infinite set of matrices.

In order to define the superior limit of a set of matrices, a topology is required. The matrices over $\overline{\mathbb{N}}$ are equipped with the following metric. When two matrices are distinct, their distance is $1/n$ where n is the maximal positive integer such that the entries that carry values at most n are the same in both matrices. If no such integer exists, the distance is 1.

Given $X \subseteq \mathcal{M}_{n,n}(\overline{\mathbb{N}})$, a matrix N (which may not be in X) belongs to the *superior limit* of X if:

- N is the limit of some sequence of matrices from X,
- there exists no $M \in X$ such that $M > N$.

Let us denote $\limsup(X)$ the set of matrices in the superior limit of S.

▶ **Theorem 10** (consequence of [5]). *Given a set $X \subseteq \mathcal{M}_{n,n}(\overline{\mathbb{N}})$, the set $\limsup(X)$ is finite. Furthermore, there is a PSPACE algorithm which, given a morphism F from $\mathbb{A}^*$ to $\mathcal{M}_{n,n}(\overline{\mathbb{N}})$, and a non-deterministic automaton for a language $L \subseteq \mathbb{A}^*$, enumerates $\limsup(F(L))$.*

The first part of the statement is a consequence of Higman's lemma. The second part relies on a result of Hashiguchi [5] (improved by Leung and Podolskiy [10]) which implies that the non-infinite entries in the matrices in $\limsup(F(L))$ are at most exponential. This is crucial for representing matrices in polynomial space, and hence exploring the state space in PSPACE.

4.3 A first reduction: the theorem of affine domination

Our goal in this section is to establish the theorem of affine domination (Theorem 3). This will be the opportunity to introduce some notations used in the subsequent section.

Let us fix ourselves two distance automata over the same alphabet $\mathbb{A}$. The first one, $\mathcal{A}_f = (\mathbb{A}, Q_f, F, I_f, T_f)$ calculates a function f. The second one, $\mathcal{A}_g = (\mathbb{A}, Q_g, G, I_g, T_g)$ calculates a function g.

Define $R_{p,0,q} \subseteq \mathbb{A}^*$ to be the set of words over which there is a run of $\mathcal{A}_g$ of weight 0 from state p to state q. Let ℓ be a non-null weight occurring in some transition of $\mathcal{A}_g$, and p, q be states in Q_g. Define $R_{p,\ell,q} \subseteq \mathbb{A}^*$ to contain the words over which there is a run of $\mathcal{A}_g$ from state p to state q which uses one transition of weight ℓ, and otherwise only transitions of weight 0. We will reuse this languages in the next section.

Proof of theorem 3. Let K be the largest number that occurs in one of $\limsup(F(R_{p,\ell,q}))$ for some states p, q and weight of a transition ℓ (such a number exists since by Theorem 10 it is the maximum of finitely many numbers). Given a matrix M, call an *m-expansion* of M a matrix $M' \geqslant M$ such that for all i, j, $M_{i,j} > K$ implies $M'_{i,j} \geqslant m$. We first show a claim concerning expansions.

Claim. For all $M \in F(R_{p,\ell,q})$ and for all m there exists an m-expansion $M' \in F(R_{p,\ell,q})$ of M.
Indeed, by definition of the superior limit, there is some $L \in \limsup(F(R_{p,\ell,q}))$ such that $L \geqslant M$. Furthermore, by choice of K, whenever $M_{i,j} > K$, $L_{i,j} = \infty$. Finally, still by definition of the superior limit, L is the limit of a sequence of matrices in $F(R_{p,\ell,q})$. Hence, for all m, there exists a matrix M' in this sequence which is sufficiently close to L that it is an m-expansion of M. This proves the claim.

Let us turn now to the core of the proof. Our goal is to prove that if f is dominated by g, (*i.e.,* there exists $\alpha : \mathbb{N} \to \mathbb{N}$ extended with $\alpha(\infty) = \infty$ such that $f \leqslant \alpha \circ g$), then $f \leqslant K(1 + g)$. The proof is by contraposition. Thus, assume $f \nleqslant K(1 + g)$. This means $f(u) > Kg(u) + K$ for some word u. We have to prove that f is not dominated by g.

The first case is $g(u) = 0$. This means that $u \in R_{p,0,q}$ with p initial and q final. Using the above claim, one can choose for all m a word $v^{(m)} \in R_{p,0,q}$ such that $F(v^{(m)})$ is an m-expansion of $F(u)$. Since $f(u) > K$, this means that for all initial state r and all final state s of $\mathcal{A}_f$, $F(u)_{r,s} > K$. This means that for all such r, s, $F(v^{(m)})_{r,s} \geqslant m$. It follows that $f(v^{(m)}) \geqslant m$. Hence over the sequence $(v^{(m)})_m$, g is bounded and f tends to infinity. This forbids the existence of a function α such that $f \leqslant \alpha \circ g$, f is not dominated by g.

Assuming $g(u) \neq 0$, the argument is similar. Remark first that $g(u)$ is finite since $f(u) > Kg(u) + K$. This means one can find $p_0, \ldots, p_k$ with p_0 initial, p_k final, and such that:

$$u = u_1 \ldots u_k, \quad u_1 \in R_{p_0,\ell_1,p_1}, \ldots, u_k \in R_{p_{k-1},\ell_k,p_k} ,$$

where $\ell_1, \ldots, \ell_k$ are all non-null and of sum $g(u)$. By the above claim, for all $i = 1 \ldots k$, and all m, one can select $v_i^{(m)}$ in R_{p_{i-1},ℓ_i,p_i} such that $F(v_i^{(m)})$ is an m-expansion of $F(u_i)$. Consider now the word $v^{(m)} = v_1^{(m)} \ldots v_k^{(m)}$. Clearly $g(v^{(m)}) = g(u)$. For the sake of contradiction, assume now that $f(v^{(m)}) < m$ for some m. This means that there exists $q_0, \ldots, q_k$ such that q_0 is initial, q_k is final, and $F(v_i^{(m)})_{q_{i-1},q_i} < m$ for all $i = 1 \ldots k$. Since $F(v_i^{(m)})$ is an m -expansion of $F(u_i)$, this implies $F(u_i)_{q_{i-1},q_i} \leqslant K$. It follows that $f(u) \leqslant Kk \leqslant Kg(u)$. A contradiction. Hence $f(v^{(m)}) \geqslant m$. Thus, g is bounded over $(v^{(m)})_m$ while f is not. As a consequence, f is not dominated by g. ◀

4.4 The reduction construction

We reuse definitions and notations of automata $\mathcal{A}_f$ and $\mathcal{A}_g$ given in the preceding section. In particular, we use the sets $R_{p,\ell,q}$ again.

Our goal is to construct a finite set of weighted matrices X that captures the relationship between f and g. The key ideas behind this reduction are the following. Each matrix (M, ℓ) in X corresponds to a set of runs of g, that start in a given state p and end in a given state

q, and use exactly one transition of non-null weight ℓ. The corresponding matrix M is in charge of (a) simulating the behaviour of F over some word corresponding to such a run (there may be infinitely many such runs, but only the finitely many matrices of the superior limit need to be considered), and (b) keeping information concerning the first and last state of the run of $\mathcal{A}_g$ for being able to check that pieces of run of g are correctly concatenated.

One also needs to define the part of the matrix in charge of controlling the validity of the run of $\mathcal{A}_g$. The construction behind Lemma 11 below is the one of a deterministic automaton, that reads words over the alphabet Q_g^2, and accepts a word $(p_1, q_1) \dots (p_k, q_k)$ if, either p_1 is not initial, or q_k is not final, or if $q_{i-1} \neq p_i$ for some i. One can verify that this language is accepted by a deterministic and complete automaton of states $Q_g \uplus \{i, \perp\}$. The unique initial state is i, and, when reading the word $(p_1, q_1) \dots (p_k, q_k)$, the automaton reaches state $\perp$ if p_1 is not initial or $q_{i-1} \neq p_i$ for some i, otherwise it reaches state q_k. The final states are the ones not in T_g plus $\perp$ plus possibly i if there are no states that are both initial and final in g. Translated in matrix form, this yields Lemma 11.

▶ **Lemma 11.** *There are $(|Q_g| + 2, |Q_g| + 2)$-matrices $(C^{p,q})_{p,q \in Q_g}$ over $\mathbb{B}$ and vectors I_C and T_C such that for all $p_1, q_1, \dots, p_k, q_k \in Q_g$,*

$$I_C \otimes C^{p_1,q_1} \otimes \cdots \otimes C^{p_k,q_k} \otimes T_C = \begin{cases} \infty & \text{if } p_1 \in I_g,\ q_1 = p_2,\ \dots,\ q_{k-1} = p_k \text{ and } q_k \in T_g, \\ 0 & \text{otherwise.} \end{cases}$$

Proof. This is implemented in matrix form as follows. For each p, q where $p, q \in Q_g$, set the matrix $C^{p,q}$ that has indices in $Q_g \cup \{i, \perp\}$, to be such that:

$$(C^{p,q})_{p',q'} = \begin{cases} 0 & \text{if } p' = i,\ p \in I_g \text{ and } q' = q, \\ 0 & \text{if } p' = i,\ p \notin I_g \text{ and } q' = \perp, \\ 0 & \text{if } p' = p \text{ and } q' = q, \\ 0 & \text{if } p' \neq i \text{ and } p' \neq p \text{ and } q' = \perp, \\ \infty & \text{otherwise.} \end{cases}$$

Define furthermore I_C be the vector with all entries ∞ but i which is 0, and let T_C be the vector with all entries equal to 0 except T_g and i if there is a state both initial and final in $\mathcal{A}_g$. ◀

We can now construct the set X as follows:

$$X = \left\{ \left(\begin{pmatrix} M & \infty \\ \infty & C^{p,q} \end{pmatrix}, \ell \right) \ : \ M \in \limsup(F(R_{p,\ell,q})) \right\} \tag{1}$$

and the vectors

$$I = (I_f \ I_C) \qquad \text{and} \qquad T = \begin{pmatrix} T_f \\ T_C \end{pmatrix} . \tag{2}$$

The following lemma shows the validity of the construction, and more particularly how it relates the comparison of distance automata to the computation of the closure of a set of weighted matrices.

▶ **Lemma 12.** *For all $\beta > 0$, $f \leqslant \beta g$ if and only if for all $(W, \ell) \in \langle X \rangle$, $I \otimes W \otimes T \leqslant \beta\ell$.*

Proof. Assume first $f \not\leqslant \beta g$, which means $f(u) > \beta g(u)$ for some u. Then clearly, $g(u)$ is finite and hence, there is an accepting run ρ of g over u. This means that one can find $p_0, \ldots, p_k$ with p_0 initial, p_k final, such that:

$$u \in R_{p_0,\ell_1,p_1} R_{p_1,\ell_2,p_2} \ldots R_{p_{k-1},\ell_k,p_k} \ ,$$

where $\ell_1, \ldots, \ell_k$ are all non-null and of sum $\ell = g(u)$. For all $i = 1 \ldots k$, set M_i to be some matrix in $\limsup(F(R_{p_{i-1},\ell_i,p_i}))$ such that $F(u_i) \leqslant M_i$. Let also C_i be C^{p_{i-1},p_i}. Clearly, the weighted matrix

$$(W_i, \ell_i) \qquad \text{with} \quad W_i = \begin{pmatrix} M_i & \infty \\ \infty & C_i \end{pmatrix}$$

belongs to X. Hence (W, ℓ) belongs to $\langle X \rangle$, where $W = W_1 \otimes \cdots \otimes W_k$. We then have $I \otimes W \otimes T = \min(x_f, x_C)$ with

$$x_f = I_f \otimes M_1 \otimes \cdots \otimes M_k \otimes T_f \qquad \text{and} \qquad x_C = I_C \otimes C_1 \otimes \cdots \otimes C_k \otimes T_C \ .$$

By choice of the M_i's, $x_f \geqslant I_f \otimes F(u) \otimes T_f = f(u)$. Furthermore, by Lemma 11, $x_C = \infty$. It follows that $I \otimes W \otimes T \geqslant f(u) > \beta g(u) = \beta\ell$.

Assume now that $f \leqslant \beta g$. Consider some $(W, \ell) \in \langle X \rangle$, it is obtained as $(W, \ell) = (W_1, \ell_1) \otimes \cdots \otimes (W_k, \ell_k)$ with $(W_i, \ell_i) \in X$ for all i. By definition of X, each of the W_i's can be written, for some $p_i, q_i \in Q_g$, as

$$W_i = \begin{pmatrix} M_i & \infty \\ \infty & C^{p_i,q_i} \end{pmatrix} \qquad \text{with} \quad M_i \in \limsup F(R_{p_i,\ell_i,q_i}).$$

Once more, one has $I \otimes W \otimes T = \min(x_f, x_C)$ with

$$x_f = I_f \otimes M_1 \otimes \cdots \otimes M_k \otimes T_f \qquad \text{and} \qquad x_C = I_C \otimes C_1 \otimes \cdots \otimes C_k \otimes T_C \ .$$

Remark first that if $x_C = 0$, clearly, $I \otimes W \otimes T = 0 \leqslant \beta\ell$. Hence, let us assume that $x_C = \infty$. This means by Lemma 11 that p_1 is initial, q_k is final, and $p_i = q_{i-1}$ for all $i = 2 \ldots k$. One needs to prove $x_f \leqslant \beta\ell$.

Assume for the sake of contradiction that $x_f > \beta\ell$. By continuity of the product, and using the definition of the superior limit, there exist words $u_1, \ldots, u_k$ such that for all $i = 1 \ldots k$, $u_i \in R_{p_i,\ell_i,q_i}$, and $I_f \otimes F(u_1) \otimes \cdots \otimes F(u_k) \otimes T_f > \beta\ell$. Furthermore, by definition of the sets R_{p_i,ℓ_i,q_i}, the fact that p_1 is initial, that q_k is final, and that $q_{i-1} = p_i$ for all $i = 2 \ldots k$, it follows that $g(u_1 \ldots u_k) = \ell$. It follows that $f(u_1 \ldots u_k) > \beta g(u_1 \ldots u_k)$. A contradiction. ◀

We are now ready to establish the main theorem of the paper.

Proof of Theorem 4. Let us consider two functions f and g computed by distance automata and some $\varepsilon > 0$. The algorithm works as follows. It computes the set X of weighted matrices as defined in this section (1), as well as the corresponding vectors I, T (2). Using Theorem 6, it computes a finitely presented set Y of weighted matrices such that $Y \approx_{\frac{\varepsilon}{2}} \langle X \rangle$. Then it tests the existence in Y of a weighted matrix (M, ℓ) such that $I \otimes \frac{1}{\ell} M \otimes T > 1 - \frac{\varepsilon}{2}$. This is easy to do for finitely presented sets. If such a weighted matrix exists, the algorithm answers "no". It answers "yes" otherwise. Let us show the correctness of this approach.

- Assume $f \leqslant (1-\varepsilon)g$, and that, for the sake of contradiction, the algorithm answers "no". This means that $I \otimes \frac{1}{\ell} M \otimes T > 1 - \frac{\varepsilon}{2}$ for some weighted matrix $(M, \ell) \in Y$. Furthermore, there exists $(M', \ell') \in \langle X \rangle$ such that $(M, \ell) \preccurlyeq_{\frac{\varepsilon}{2}} (M', \ell')$. This implies $\frac{1}{\ell} M \leqslant \frac{1}{\ell'} M' + \frac{\varepsilon}{2}$. It follows that $I \otimes M' \otimes T > (1-\varepsilon)\ell'$. This contradicts Lemma 12.

- Assume $f \not\preccurlyeq g$, then by Lemma 12, there exists a matrix $M \in \langle X \rangle$ such that $I \otimes \frac{1}{\ell} M \otimes T > 1$. Furthermore, there exists $M' \in Y$ such that $(M, \ell) \preccurlyeq_{\frac{\varepsilon}{2}} (M', \ell')$. This implies $\frac{1}{\ell} M \leqslant \frac{1}{\ell'} M' + \frac{\varepsilon}{2}$, and hence $I \otimes \frac{1}{\ell'} M' \otimes T > 1 - \frac{\varepsilon}{2}$. The algorithm answers "no". ◀

5 Conclusion and further remarks

In this paper, we provided an algorithm for deciding the approximate comparison of distance automata. This algorithm involves the computation of the closure under product of sets of weighted matrices, a result of independent interest.

The main open question is the complexity of the problem. It is clear that the problem is at least PSPACE hard. A correct implementation of the arguments in this paper shows that EXSPACE is an upper bound. We do not know the exact complexity.

Acknowledgments

We would like to thank Jean Mairesse for numerous enlightening discussions as well as the anonymous referees for their helpful comments.

References

1 Mikolaj Bojańczyk and Thomas Colcombet. Bounds in ω-regularity. In *LICS 06*, pages 285–296, 2006.
2 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *Automata, languages and programming. Part II*, volume 5556 of *Lecture Notes in Comput. Sci.*, pages 139–150. Springer, Berlin, 2009.
3 Thomas Colcombet. Green's relations and their use in automata theory. In Adrian Horia Dediu, Shunsuke Inenaga, and Carlos Martín-Vide, editors, *LATA*, volume 6638 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2011. Invited lecture.
4 Kosaburo Hashiguchi. Limitedness theorem on finite automata with distance functions. *J. Comput. Syst. Sci.*, 24(2):233–244, 1982.
5 Kosaburo Hashiguchi. New upper bounds to the limitedness of distance automata. *Theor. Comput. Sci.*, 233(1–2):19–32, 2000.
6 Daniel Kirsten. Distance desert automata and the star height problem. *RAIRO*, 3(39):455–509, 2005.
7 Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *Internat. J. Algebra Comput.*, 4(3):405–425, 1994.
8 Manfred Kufleitner. A proof of the factorization forest theorem. Technical Report Nr. 2007/05, Universität Stuttgart, Germany, 2007.
9 Hing Leung. On the topological structure of a finitely generated semigroup of matrices. *Semigroup Forum*, 37:273–287, 1988.
10 Hing Leung and Viktor Podolskiy. The limitedness problem on distance automata: Hashiguchi's method revisited. *Theoretical Computer Science*, 310(1-3):147–158, 2004.
11 Marcel-Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.
12 Imre Simon. Piecewise testable events. In H. Brackage, editor, *Proc. 2nd GI Conf.*, volume 33, pages 214–222. Springer, 1975.
13 Imre Simon. On semigroups of matrices over the tropical semiring. *RAIRO ITA*, 28(3-4):277–294, 1994.

The Rank of Tree-Automatic Linear Orderings

Martin Huschenbett

Institut für Theoretische Informatik, Technische Universität Ilmenau, Germany
martin.huschenbett@tu-ilmenau.de

Abstract

A tree-automatic structure is a structure whose domain can be encoded by a regular tree language such that each relation is recognisable by a finite automaton processing tuples of trees synchronously. The finite condensation rank (FC-rank) of a linear ordering measures how far it is away from being dense. We prove that the FC-rank of every tree-automatic linear ordering is below ω^ω. This generalises Delhommé's result that each tree-automatic ordinal is less than ω^{ω^ω}. Furthermore, we show an analogue for tree-automatic linear orderings where the branching complexity of the trees involved is bounded.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases tree-automatic structures, linear orderings, finite condensation rank, computable model theory

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.586

1 Introduction

The fundamental idea of automatic structures can be traced back to the 1960s when Büchi, Elgot, Rabin, and others used finite automata to provide decision procedures for the first-order theory of Presburger arithmetic $(\mathbb{N}; +)$ and several other logical problems. Hodgson generalised this idea to the concept of *automaton decidable* first-order theories. Independently of Hodgson and inspired by the successful employment of finite automata and their methods in group theory, Khoussainov and Nerode [8] initiated the systematic investigation of *automatic structures*. Recalling the efforts from the 1960s, Blumensath [2] extended this notion beyond finite automata to finite automaton models recognising, e.g., finite trees.

Basically, a countable relational structure is *tree-automatic* or *tree-automatically presentable* if its elements can be encoded by finite trees in such a way that its domain and its relations are recognisable by finite automata processing either single trees or tuples of trees synchronously. *String-automatic* structures can be regarded as a special case where only specific simple trees—which effectively represent strings—are used. In contrast to the more general concept of *computable structures* and based on the strong closure properties of recognisability, automatic structures provide pleasant algorithmic features. In particular, they possess decidable first-order theories.

Due to this latter circumstance, the concept of automatic structures gained a lot attention which led to noticeable progress (cf. [1, 13]). Automatic presentations were found for many structures and others like the random graph were shown not to be automatic at all. Some structures are provably on an intermediate level, they are tree-automatic but not string-automatic, for instance Skolem arithmetic $(\mathbb{N}; \times)$. For the classes of ordinals and Boolean algebras it was even possible to characterise their (string-)automatic members. Certain extensions of first-order logic which preserve decidability of the corresponding theory were detected. The question whether two automatic structures are isomorphic turned out to be highly undecidable in general as well as for some restricted classes of structures. In contrast, the isomorphism problem for string-automatic ordinals was proven to be decidable. Recently, some classes of structures for which string-automaticity of their tree-automatic members is decidable were identified. Last but not least, different classes of automatic structures were characterised by means of logical interpretations in universal structures.

The characterisation of automatic ordinals was provided by Delhommé [4]. An ordinal is string-automatic if, and only if, it is less than ω^ω. The respective bound for tree-automatic ordinals is ω^{ω^ω}. To obtain these results, Delhommé developed and employed a *decomposition technique* for automatic structures. Later, Khoussainov, Rubin, and Stephan generalised the only-if-implication of the string-automatic case by proving that the finite condensation rank (FC-rank) of any string-automatic linear ordering is below ω [9]. Roughly speaking, the FC-rank is an ordinal indicating how far a linear ordering is away from being dense. Basically, they applied the decomposition technique for string-automatic structures to the class of (scattered) linear orderings. Since that time, it is presumed that the FC-rank of every tree-automatic linear ordering is below ω^ω. However, this conjecture has not been verified yet.[1] We close this gap by our first main result.

▶ **Theorem 4.6.** *The* FC*-rank of every tree-automatic linear ordering is strictly below ω^ω.*

Again, the proof is an application of the decomposition technique to the class of (scattered) linear orderings. Unfortunately, Delhommé never provided a proof of his decomposition theorem for tree-automatic structures. As his wording of the theorem is also too weak for our purposes, we state and prove a refined version (Theorem 3.7).[2] However, the main difficulty in showing Theorem 4.6 is to substantiate that scattered linear orderings are accessible to the decomposition technique for *tree-automatic* structures (Proposition 4.3 and Corollary 4.5).

In the last section, we demonstrate how to adapt the (refined) decomposition technique to finite-rank tree-automatic structures (cf. [1, Section 1.3.7]). Roughly speaking, the rank of a tree-automatic structure describes the branching complexity of the trees involved and is measured in terms of the Cantor-Bendixson rank (cf. [9]). Our second main result is the following analogue of Theorem 4.6 for finite-rank tree-automatic linear orderings.

▶ **Theorem 5.2.** *Let $k \in \mathbb{N}_+$. The* FC*-rank of every rank-k tree-automatic linear ordering is strictly below ω^k.*

In the very end, we briefly sketch how to apply these results to show upper bounds on the Cantor-Bendixson rank of (finite-rank) tree-automatic finitely branching order trees, i.e., partial orderings which happen to be trees.

2 Background

2.1 Tree-Automatic Structures

This section recalls the basic notions of tree-automatic structures (cf. [1, 2]).

Let $\mathbf{2} = \{0, 1\}$ be the binary alphabet. The set of all *strings* over $\mathbf{2}$ is denoted by $\mathbf{2}^\star$ and the *empty string* by ε. A *tree domain* is a non-empty, finite, prefix-closed subset $D \subseteq \mathbf{2}^\star$. The *boundary* of D is the set $\partial D = \{\, ud \mid u \in D, d \in \mathbf{2}, ud \notin D \,\}$. Let Σ be an alphabet. A *finite Σ-labelled tree* (or just *tree*) is a map $t \colon D \to \Sigma$ where $\operatorname{dom}(t) = D$ is a tree

[1] Due to personal communication, S. Jain, B. Khoussainov, P. Schlicht, and F. Stephan recently verified the conjecture for scattered linear orderings. This implies that the FC-rank of arbitrary tree-automatic linear orderings is at most ω^ω (including).

[2] A similar refinement was used to bound the ordinal height of well-founded order trees [7].

domain. The set of all finite Σ-labelled trees is denoted by T_Σ. Its subsets are called *(tree) languages*. For $t \in T_\Sigma$ and a node $u \in \operatorname{dom}(t)$ the *subtree* $t\restriction u \in T_\Sigma$ rooted at u is defined by $\operatorname{dom}(t\restriction u) = \{\, v \in \mathbf{2}^\star \mid uv \in \operatorname{dom}(t)\,\}$ and $(t\restriction u)(v) = t(uv)$. For nodes $u_1, \dots, u_n \in \operatorname{dom}(t)$ which form an anti-chain in t, i.e., they are mutually not prefixes of each other, and trees $t_1, \dots, t_n \in T_\Sigma$, we consider the tree $t[u_1/t_1, \dots, u_n/t_n] \in T_\Sigma$ which is obtained from t by simultaneously replacing for each $i \in [1, n]$ the subtree rooted at u_i by t_i.

A *(deterministic bottom-up) tree automaton* $\mathcal{M} = (Q, \iota, \delta, F)$ over Σ consists of a finite set Q of *states*, a *start state* $\iota \in Q$, a *transition function* $\delta\colon \Sigma \times Q \times Q \to Q$, and a set $F \subseteq Q$ of *accepting* states. For all $t \in T_\Sigma$, $u \in \operatorname{dom}(t) \cup \partial \operatorname{dom}(t)$, and maps $\rho\colon U \to Q$ with $U \subseteq \partial \operatorname{dom}(t)$ a state $\mathcal{M}(t, u, \rho) \in Q$ is defined recursively by

$$\mathcal{M}(t,u,\rho) = \begin{cases} \delta\big(t(u), \mathcal{M}(t,u0,\rho), \mathcal{M}(t,u1,\rho)\big) & \text{if } u \in \operatorname{dom}(t), \\ \rho(u) & \text{if } u \in U, \\ \iota & \text{if } u \in \partial\operatorname{dom}(t) \setminus U. \end{cases}$$

We omit the parameter u (resp. U) if $u = \varepsilon$ (resp. $U = \emptyset$). Notice that $\mathcal{M}(t, u) = \mathcal{M}(t\restriction u)$. The tree language *recognised* by $\mathcal{M}$ is the set $L(\mathcal{M}) = \{\, t \in T_\Sigma \mid \mathcal{M}(t) \in F\,\}$. A language $L \subseteq T_\Sigma$ is *regular* if it can be recognised by some tree automaton.

Let $\diamond \notin \Sigma$ be a new symbol and $\Sigma_\diamond = \Sigma \cup \{\diamond\}$. The *convolution* of an n-tuple $\bar{t} = (t_1, \dots, t_n) \in (T_\Sigma)^n$ of trees is the tree $\otimes\bar{t} \in T_{\Sigma_\diamond^n}$ defined by

$$\operatorname{dom}(\otimes\bar{t}) = \operatorname{dom}(t_1) \cup \dots \cup \operatorname{dom}(t_n) \quad \text{and} \quad (\otimes\bar{t})(u) = \big(t'_1(u), \dots, t'_n(u)\big),$$

where $t'_i(u) = t_i(u)$ if $u \in \operatorname{dom}(t_i)$ and $t'_i(u) = \diamond$ otherwise. If $n = 2$, we also write $t_1 \otimes t_2$ for $\otimes(t_1, t_2)$. A relation $R \subseteq (T_\Sigma)^n$ is *automatic* if the tree language $\otimes R = \{\, \otimes\bar{t} \mid \bar{t} \in R\,\} \subseteq T_{\Sigma_\diamond^n}$ is regular. We say a tree automaton *recognises* R if it recognises $\otimes R$.

A relational structure $\mathfrak{A} = \big(A; R_1^{\mathfrak{A}}, \dots, R_n^{\mathfrak{A}}\big)$ is called *tree-automatic* if its domain A is a regular tree language and each relation $R_i^{\mathfrak{A}}$ is automatic.[3] In this situation, a *tree-automatic presentation* of $\mathfrak{A}$ is a tuple of tree automata recognising A and the $R_i^{\mathfrak{A}}$. Abusing notation, we sometimes call any structure *tree-automatic* (in a wider sense) which is isomorphic to some tree-automatic structure (in the narrow sense). The following theorem lays out the main motivation for investigating tree-automatic structures.

▶ **Theorem 2.1** (Blumensath [2]). *Let $\mathfrak{A}$ be a tree-automatic structure. For every first-order formula $\phi(\bar{x})$ in the signature of $\mathfrak{A}$ the relation $\phi^{\mathfrak{A}}$ defined by ϕ is automatic and one can compute a tree automaton recognising $\phi^{\mathfrak{A}}$ from a tree-automatic presentation of $\mathfrak{A}$ and the formula ϕ. In particular, the first-order theory of $\mathfrak{A}$ is decidable.*

2.2 Linear Orderings

This section recalls the necessary background on linear orderings (cf. [12]).

A *linear ordering* is a structure $\mathfrak{A} = \big(A; \leq^{\mathfrak{A}}\big)$ where $\leq^{\mathfrak{A}}$ is a *non-strict* linear order on A. The corresponding *strict* linear order is denoted by $<^{\mathfrak{A}}$. If $\mathfrak{A}$ is clear from the context we omit the superscript $\mathfrak{A}$. For $n \in \mathbb{N}$ the (isomorphism type of the) linear ordering with exactly n elements is denoted by $\mathbf{n} = \big(\{0, \dots, n-1\}; \leq\big)$. Let $\mathfrak{I}$ and $\mathfrak{A}_i$ for each $i \in I$ be linear orderings. The $\mathfrak{I}$-*sum* of the $\mathfrak{A}_i$ is the linear ordering $\mathfrak{A} = \sum_{i \in \mathfrak{I}} \mathfrak{A}_i$ defined by $A = \biguplus_{i \in I} A_i$ and $x \leq^{\mathfrak{A}} y$ iff either $x, y \in A_i$ and $x \leq^{\mathfrak{A}_i} y$ for some $i \in I$ or $x \in A_i$ and $y \in A_j$ for some $i, j \in I$ with $i <^{\mathfrak{I}} j$. In case that $\mathfrak{I}$ is finite, say $\mathfrak{I} = \mathbf{n}$, we also write $\mathfrak{A}_0 + \dots + \mathfrak{A}_{n-1}$.

[3] By convention, structures are named in Fraktur and their domains by the same letter in Roman.

A linear ordering $\mathfrak{A}$ is *dense* if for all $x, y \in A$ with $x < y$ there exists a $z \in A$ such that $x < z < y$. Up to isomorphism, there are only five countable dense linear orderings, namely $\mathbf{1}$, η, $\mathbf{1} + \eta$, $\eta + \mathbf{1}$, and $\mathbf{1} + \eta + \mathbf{1}$, where $\eta = (\mathbb{Q}; \leq)$ are the rational numbers ordered as usual. At the opposite extreme, $\mathfrak{A}$ is *scattered* if η cannot be embedded into $\mathfrak{A}$. Examples of scattered linear orderings include the natural numbers $\omega = (\mathbb{N}; \leq)$, the reversed natural numbers $\omega^* = (\mathbb{N}; \geq)$, the integers $\zeta = (\mathbb{Z}; \leq)$, and the finite linear ordering $\mathbf{n}$ for each $n \in \mathbb{N}$. Moreover, all well-orderings and scattered sums of scattered linear orderings are scattered.

For two subsets $X, Y \subseteq A$ of a linear ordering $\mathfrak{A}$ we write $X \ll Y$ if $x < y$ for all $x \in X$ and $y \in Y$. A *condensation (relation)* on $\mathfrak{A}$ is an equivalence relation $\sim$ on A such that each $\sim$-class is a (possibly non-closed) interval of $\mathfrak{A}$. In this situation, the set of all $\sim$-classes is (strictly) linearly ordered by $\ll$ and we denote this linear ordering by $\mathfrak{A}/{\sim}$. An example of a condensation is given by $x \sim y$ if x and y are at finite distance in $\mathfrak{A}$. Transfinitely iterating this process leads to the inductive definition of a condensation $\sim_\alpha$ on $\mathfrak{A}$ for each ordinal α:

1. $\sim_0$ is the identity relation on $\mathfrak{A}$,
2. for successor ordinals $\alpha = \beta + 1$ let $x \sim_\alpha y$ iff there are only finitely many elements between the $\sim_\beta$-classes of x and y in $\mathfrak{A}/{\sim_\beta}$, and
3. for limit ordinals α let $x \sim_\alpha y$ iff $x \sim_\beta y$ for some $\beta < \alpha$.

There exists an α such that $\sim_\alpha$ and $\sim_\beta$ coincide for each $\beta \geq \alpha$. The least such α is called *finite condensation rank* (FC-rank) of $\mathfrak{A}$ and denoted by $\mathrm{FC}(\mathfrak{A})$. In particular, if $\mathfrak{A}$ is countable then $\mathrm{FC}(\mathfrak{A})$ is also countable [12, Theorem 5.9]. Moreover, each $\sim_\alpha$-class is a scattered interval of $\mathfrak{A}$ and $\mathfrak{A}/{\sim_\alpha}$ is dense, proving the following theorem.

▶ **Theorem 2.2** (Hausdorff [12, Theorem 4.9]). *Every linear ordering $\mathfrak{A}$ is a dense sum of scattered linear orderings, i.e., there are a dense linear ordering $\mathfrak{I}$ and scattered linear orderings $\mathfrak{A}_i$ for each $i \in I$ such that $\mathfrak{A} = \sum_{i \in \mathfrak{I}} \mathfrak{A}_i$.*

Due to Hausdorff, there is a valuable inductive construction of the class of countable scattered linear orderings. For each countable ordinal α a class $\mathcal{VD}_\alpha$ is defined as follows:

1. $\mathcal{VD}_0 = \{\mathbf{0}, \mathbf{1}\}$ and
2. for $\alpha > 0$ the class $\mathcal{VD}_\alpha$ contains all ζ-sums of elements from $\mathcal{VD}_{<\alpha} = \bigcup_{\beta<\alpha} \mathcal{VD}_\alpha$.

The class $\mathcal{VD}$ of *very discrete* linear orderings is the union of all classes $\mathcal{VD}_\alpha$ and the VD-*rank* of some $\mathfrak{A} \in \mathcal{VD}$, denoted by $\mathrm{VD}(\mathfrak{A})$, is the least ordinal α with $\mathfrak{A} \in \mathcal{VD}_\alpha$.

▶ **Theorem 2.3** (Hausdorff [12, Theorem 5.24]). *A countable linear ordering $\mathfrak{A}$ is scattered if, and only if, it is contained in $\mathcal{VD}$. In case that $\mathfrak{A}$ is scattered,* $\mathrm{FC}(\mathfrak{A}) = \mathrm{VD}(\mathfrak{A})$.

3 Delhommé's Decomposition Technique

3.1 Augmentations and the Decomposition Theorem

In this section, we present the decomposition technique Delhommé developed and employed to show that every tree-automatic ordinal is less than ω^{ω^ω} [4]. As we want to apply this technique to linear orderings, we restrict our attention to structures whose signature contains only a single binary relation symbol $\leq$, i.e., (directed) graphs. First, we introduce the central notions of sum- and box-augmentations. For a graph $\mathfrak{A}$ and a subset $B \subseteq A$ we denote by $\mathfrak{A}{\restriction}B$ the subgraph induced by B.

▶ **Definition 3.1.** A graph $\mathfrak{A}$ is a *sum-augmentation* of graphs $\mathfrak{B}_1, \ldots, \mathfrak{B}_n$ if there exists a finite partition $A = \biguplus_{i \in [1,n]} A_i$ of $\mathfrak{A}$ such that $\mathfrak{A}{\restriction}A_i \cong \mathfrak{B}_i$ for each $i \in [1, n]$.

▶ **Example 3.2.** Let $\mathfrak{B}_1, \ldots, \mathfrak{B}_n$ be graphs.

1. Suppose that the $\mathfrak{B}_i$ are linear orderings and let $\mathfrak{A}$ be a linearisation of the partial ordering $\mathfrak{B}_1 \amalg \cdots \amalg \mathfrak{B}_n = \big(\biguplus_{i\in[1,n]} B_i; \preceq\big)$ with $x \preceq y$ iff $x, y \in B_i$ and $x \leq^{\mathfrak{B}_i} y$ for some $i \in [1, n]$. Then $\mathfrak{A}$ is a sum-augmentation of $\mathfrak{B}_1, \ldots, \mathfrak{B}_n$.
2. Let $\mathfrak{A}$ be a linear ordering which is a sum-augmentation of $\mathfrak{B}_1, \ldots, \mathfrak{B}_n$. First, each $\mathfrak{B}_i$ can be embedded into $\mathfrak{A}$ and is hence a linear ordering, which is scattered in case $\mathfrak{A}$ is scattered. Second, $\mathfrak{A}$ is isomorphic to a linearisation of $\mathfrak{B}_1 \amalg \cdots \amalg \mathfrak{B}_n$.

▶ **Definition 3.3.** A graph $\mathfrak{A}$ is a *box-augmentation* of graphs $\mathfrak{B}_1, \ldots, \mathfrak{B}_n$ if there exists a bijection $f \colon B_1 \times \cdots \times B_n \to A$ such that for all $j \in [1, n]$ and $\bar{x} \in \prod_{i\in[1,n], i\neq j} B_i$ the map $f^j_{\bar{x}} \colon B_j \to A$ defined by $f^j_{\bar{x}}(b) = (x_1, \ldots, x_{j-1}, b, x_{j+1}, \ldots, x_n)$ is an embedding of $\mathfrak{B}_j$ into $\mathfrak{A}$.

▶ **Example 3.4.** Let $\mathfrak{B}_1, \ldots, \mathfrak{B}_n$ be graphs.

1. Suppose that the $\mathfrak{B}_i$ are linear orderings and let $\mathfrak{A}$ be a linearisation of the partial ordering $\mathfrak{B}_1 \times \cdots \times \mathfrak{B}_n = \big(B_1 \times \cdots \times B_n; \preceq\big)$ with $\bar{x} \preceq \bar{y}$ iff $x_i \leq^{\mathfrak{B}_i} y_i$ for all $i \in [1, n]$. The identity map $B_1 \times \cdots \times B_n \to A$ witnesses that $\mathfrak{A}$ is a box-augmentation of $\mathfrak{B}_1, \ldots, \mathfrak{B}_n$.
2. Let $\mathfrak{A}$ be a linear ordering which is a box-augmentation of $\mathfrak{B}_1, \ldots, \mathfrak{B}_n$. First, each $\mathfrak{B}_i$ can be embedded into $\mathfrak{A}$ and is hence a linear ordering, which is scattered in case $\mathfrak{A}$ is scattered. Second, the bijection f from Definition 3.3 above is an isomorphism between a linearisation of $\mathfrak{B}_1 \times \cdots \times \mathfrak{B}_n$ and $\mathfrak{A}$.

In order to make the class of linear orderings accessible to the decomposition technique, we have to study the connection between box-augmentations and the FC-rank. More precisely, given some linear orderings $\mathfrak{B}_1, \ldots, \mathfrak{B}_n$ we want to establish a bound on the FC-rank of any linear ordering which is a box-augmentation of $\mathfrak{B}_1, \ldots, \mathfrak{B}_n$ in terms of the FC-ranks of the $\mathfrak{B}_i$. However, the following example indicates that this is impossible in general.

▶ **Example 3.5.** Consider the partial ordering $\omega \times \omega^* = (\mathbb{N} \times \mathbb{N}, \preceq)$, where $\preceq$ is defined as above. For each $i \in \mathbb{Z}$ the elements $(m, n) \in \mathbb{N} \times \mathbb{N}$ with $m - n = i$ form an anti-chain, i.e., they are mutually incomparable by $\preceq$. Therefore, any ζ-sum of countably infinite linear orderings is (isomorphic to) a linearisation of $\omega \times \omega^*$. In particular, for any countable ordinal $\alpha > 1$ there exists a (scattered) linear ordering $\mathfrak{A}$ with $\mathrm{FC}(\mathfrak{A}) = \alpha$ which is a box-augmentation of ω, ω^*. Compare this to the fact that $\mathrm{FC}(\omega) = \mathrm{FC}(\omega^*) = 1$.

Owing to this observation, we introduce a restricted notion of box-augmentations. Therein, a *finite colouring* of a graph $\mathfrak{A}$ is a map $\sigma \colon A \times A \to Q$ into a finite set Q such that $\sigma(x, y) = \sigma(x', y')$ and $x \leq y$ imply $x' \leq y'$ for all $x, y, x', y' \in A$.

▶ **Definition 3.6.** The box-augmentation in Definition 3.3 is called *tame* if for each $i \in [1, n]$ there exists a finite colouring $\sigma_i \colon B_i \times B_i \to Q_i$ of $\mathfrak{B}_i$ such that the map

$$f(\sigma_1, \ldots, \sigma_n) \colon A \times A \to Q_1 \times \ldots \times Q_n, \big(f(\bar{x}), f(\bar{y})\big) \mapsto \big(\sigma_1(x_1, y_1), \ldots, \sigma_n(x_n, y_n)\big)$$

is a finite colouring of $\mathfrak{A}$.

▶ Remark. In the situation of Definition 3.6, assume that all Q_i are the same set, say $\{1, \ldots, m\}$. For each $i \in [1, n]$ consider the structure $\mathfrak{C}_i = \big(B_i; R^{\mathfrak{C}_i}_1, \ldots, R^{\mathfrak{C}_i}_m\big)$ with $R^{\mathfrak{C}_i}_q = \sigma_i^{-1}(q)$. Due to the definition of a finite colouring, the $R^{\mathfrak{C}_i}_q$ form a finite partition of $B_i \times B_i$ which is compatible with $\leq^{\mathfrak{B}_i}$. Therefore, the graph $\mathfrak{A}$ is a generalised product—in the sense of Feferman and Vaught—of the structures $\mathfrak{C}_1, \ldots, \mathfrak{C}_n$ using only atomic formulae.

We will see later, in Lemma 4.4 and its proof, that every linear ordering $\mathfrak{A}$ which is a tame box-augmentation of ω, ω^* is scattered and satisfies $\mathrm{FC}(\mathfrak{A}) \leq 3$. We conclude this section by providing our refined version of Delhommé's decomposition theorem. For a structure

$\mathfrak{A}$, a first-order formula $\phi(x, y_1, \dots, y_r)$ in the signature of $\mathfrak{A}$, and a tuple $\bar{s} \in A^r$ we let $\phi^{\mathfrak{A}}(\cdot, \bar{s}) = \{\, t \in A \mid \mathfrak{A} \models \phi(t, \bar{s}) \,\}$ and abbreviate $\mathfrak{A}{\restriction}\phi^{\mathfrak{A}}(\cdot, \bar{s})$ by $\mathfrak{A}{\restriction}\phi, \bar{s}$.

▶ **Theorem 3.7.** *Let $\mathfrak{A}$ be a tree-automatic graph and $\phi(x, y_1, \dots, y_r)$ a first-order formula in the signature of graphs. Then there exists a finite set $\mathcal{S}^{\mathfrak{A}}_{\phi}$ of tree-automatic graphs such that for all tuples $\bar{s} \in A^r$ the graph $\mathfrak{A}{\restriction}\phi, \bar{s}$ is a sum-augmentation of* tame *box-augmentations of elements from $\mathcal{S}^{\mathfrak{A}}_{\phi}$.*

▶ Remark. The only, but very important, difference to Delhommé's decomposition theorem is our addition of the word *tame*. Since by Example 3.5 there is no reasonable connection between the FC-rank and arbitrary box-augmentations, the version without *tame* cannot be used to investigate bounds on the FC-rank of tree-automatic (scattered) linear orderings.

Proof of Theorem 3.7. Suppose that $A \subseteq T_\Sigma$ for some alphabet Σ. Let $\mathcal{M}_\leq$ and $\mathcal{M}_\phi$ be tree automata recognising $\leq^{\mathfrak{A}}$ and $\phi^{\mathfrak{A}}$ and $Q_\leq$ and Q_ϕ be their state sets, respectively. In order to simplify notation, for $t \in T_\Sigma$ we put $t^{\leq} = t \otimes t$ and define $t^\phi \in T_{\Sigma_\diamond^{1+r}}$ by $\mathrm{dom}\big(t^\phi\big) = \mathrm{dom}(t)$ and $t^\phi(u) = \big(t(u), \diamond, \dots, \diamond\big)$, where the number of $\diamond$-symbols is r.

As a first step, we construct the set $\mathcal{S}^{\mathfrak{A}}_{\phi}$. Therefore, consider the set $\Gamma = Q_\leq \times Q_\phi \times 2^{Q_\leq}$. For each $\gamma = \big(q_\leq, q_\phi, P_\leq\big) \in \Gamma$ we define a graph $\mathfrak{S}_\gamma$ by

$$S_\gamma = \big\{\, t \in T_\Sigma \;\big|\; \mathcal{M}_\leq\big(t^{\leq}\big) = q_\leq \;\&\; \mathcal{M}_\phi\big(t^\phi\big) = q_\phi \,\big\} \quad \text{and} \quad t_1 \leq^{\mathfrak{S}_\gamma} t_2 \text{ iff } \mathcal{M}_\leq(t_1 \otimes t_2) \in P_\leq \,.$$

Clearly, $\mathfrak{S}_\gamma$ is tree-automatic. Finally, we put $\mathcal{S}^{\mathfrak{A}}_{\phi} = \{\, \mathfrak{S}_\gamma \mid \gamma \in \Gamma \,\}$.

We have to show that for each $\bar{s} = (s_1, \dots, s_r) \in A^r$ the subgraph $\mathfrak{A}{\restriction}\phi, \bar{s}$ is a sum-augmentation of box-augmentations of elements from $\mathcal{S}^{\mathfrak{A}}_{\phi}$. Therefore, we fix a tuple $\bar{s}$, put $\mathfrak{B} = \mathfrak{A}{\restriction}\phi, \bar{s}$, and consider the tree domain $D = \bigcup_{1 \leq i \leq n} \mathrm{dom}(s_i)$. The *$\bar{s}$-type* of a tree $t \in T_\Sigma$ is defined as $\mathrm{tp}_{\bar{s}}(t) = \big(t{\restriction}D, U, \rho_\leq, \rho_\phi\big)$, where $t{\restriction}D \in T_\Sigma$ is the restriction of t to the tree domain $\mathrm{dom}(t) \cap D$, $U = \mathrm{dom}(t) \cap \partial D$, and $\rho_R \colon U \to Q_R, u \mapsto \mathcal{M}_R\big((t{\restriction}u)^R\big)$ for $R \in \{<, \phi\}$. Observe that

$$\mathcal{M}_\phi\big(\otimes(t, \bar{s})\big) = \mathcal{M}_\phi\big(\otimes(t{\restriction}D, \bar{s})\big[\big(u/(t{\restriction}u)^\phi\big)_{u \in U}\big]\big) = \mathcal{M}_\phi\big(\otimes(t{\restriction}D, \bar{s}), \rho_\phi\big) \,. \tag{1}$$

Hence, $\mathrm{tp}_{\bar{s}}(t)$ completely determines whether $t \in B$. Since D is finite, there are only finitely many different $\bar{s}$-types. Thus, the equivalence relation $\sim_{\bar{s}}$ on T_Σ defined by $t_1 \sim_{\bar{s}} t_2$ iff $\mathrm{tp}_{\bar{s}}(t_1) = \mathrm{tp}_{\bar{s}}(t_2)$ has finite index. Due to the mentioned consequence of Eq. (1), B is a union of $\sim_{\bar{s}}$-classes. Say $B = C_1 \uplus \dots \uplus C_m$ is the corresponding partition of B into $\sim_{\bar{s}}$-classes, then $\mathfrak{B}$ is a sum-augmentation of $\mathfrak{B}{\restriction}C_1, \dots, \mathfrak{B}{\restriction}C_m$.

As a final step, we fix a single $\sim_{\bar{s}}$-class $C \subseteq B$ and provide a tuple of graphs from $\mathcal{S}^{\mathfrak{A}}_{\phi}$ of whom $\mathfrak{C} = \mathfrak{B}{\restriction}C$ is a box-augmentation. Let $\tau = \big(t_0, U, \rho_\leq, \rho_\phi\big)$ be the $\bar{s}$-type corresponding to C. For $u \in U$ we put $\gamma(u) = \big(\rho_\leq(u), \rho_\phi(u), P_\leq(u)\big) \in \Gamma$, where $P_\leq(u)$ is the set of all $q \in Q_\leq$ for which $\mathcal{M}_\leq\big(t_0^{\leq}, \rho_\leq[u \mapsto q]\big)$ is an accepting state in $\mathcal{M}_\leq$.

It is easy to show that the map $f \colon \prod_{u \in U} S_{\gamma(u)} \to C$ with $f\big((x_u)_{u \in U}\big) = t_0[(u/x_u)_{u \in U}]$ is a bijection witnessing that $\mathfrak{C}$ is a box-augmentation of the collection of the $\mathfrak{S}_{\gamma(u)}$. To see that this box-augmentation is tame, consider for each $u \in U$ the finite colouring σ_u of $\mathfrak{S}_{\gamma(u)}$ which is given by $\sigma_u(x, y) = \mathcal{M}_\leq(x \otimes y)$ and let $\sigma = f\big((\sigma_u)_{u \in U}\big)$. Then $\mathcal{M}_\leq(x \otimes y)$ is completely determined by $\sigma(x, y)$ for all $x, y \in C$ and hence σ is a finite colouring of $\mathfrak{C}$. ◀

3.2 Indecomposability and Tree-Automatic Ordinals

According to Delhommé's approach [4], we introduce the notion of indecomposable ordinals and provide a refined version of his result on indecomposability as a corollary of Theorem 3.7.

Therefore, suppose that $\mathcal{C}$ is a class of graphs and an ordinal $\mathrm{rk}(\mathfrak{A})$ is assigned to each $\mathfrak{A} \in \mathcal{C}$ in an isomorphism invariant way. We say that $\mathcal{C}$ is *ranked* by rk. An ordinal α is rk-*sum-indecomposable* if for any graph $\mathfrak{A} \in \mathcal{C}$ with $\mathrm{rk}(\mathfrak{A}) = \alpha$ and all graphs $\mathfrak{B}_1, \dots, \mathfrak{B}_n$ which $\mathfrak{A}$ is a sum-augmentation of, there exists an $i \in [1, n]$ with $\mathfrak{B}_i \in \mathcal{C}$ and $\mathrm{rk}(\mathfrak{B}_i) = \alpha$. Analogously, rk-*box-indecomposable* and rk-*tame-box-indecomposable* ordinals are defined. Although Example 3.5 shows that neither FC- nor VD-box-indecomposability are useful, Corollary 4.5 indicates that VD-*tame*-box-indecomposability is indeed a reasonable notion.

▶ **Corollary 3.8.** *Let $\mathcal{C}$ be a class of graphs ranked by* rk*, $\mathfrak{A}$ a tree-automatic graph, and $\phi(x, y_1, \dots, y_r)$ a first-order formula in the signature of graphs. Then there are only finitely many ordinals α which are simultaneously* rk*-sum- and* rk*-tame-box-indecomposable and admit a tuple $\bar{s} \in A^r$ with $\mathfrak{A}{\restriction}\phi, \bar{s} \in \mathcal{C}$ and $\mathrm{rk}\big(\mathfrak{A}{\restriction}\phi, \bar{s}\big) = \alpha$.*

Proof. Let $\mathcal{S}^{\mathfrak{A}}_{\phi}$ be the finite set of graphs which exists by Theorem 3.7. Consider an ordinal α satisfying the condition above, witnessed by $\bar{s} \in A^r$. There exist box-augmentations $\mathfrak{B}_1, \dots, \mathfrak{B}_m$ of elements from $\mathcal{S}^{\mathfrak{A}}_{\phi}$ such that $\mathfrak{A}{\restriction}\phi, \bar{s}$ is a sum-augmentation of them. Then there is an $i \in [1, m]$ such that $\mathfrak{B}_i \in \mathcal{C}$ and $\mathrm{rk}(\mathfrak{B}_i) = \alpha$. Now, there exist $\mathfrak{C}_1, \dots, \mathfrak{C}_n \in \mathcal{S}^{\mathfrak{A}}_{\phi}$ of which $\mathfrak{B}_i$ is a tame box-augmentation. Again, there is a $j \in [1, n]$ with $\mathfrak{C}_j \in \mathcal{C}$ and $\mathrm{rk}(\mathfrak{C}_j) = \alpha$. Altogether, α belongs to the finite set $\{\, \mathrm{rk}(\mathfrak{B}) \mid \mathfrak{B} \in \mathcal{S}^{\mathfrak{A}}_{\phi} \,\}$. ◀

To illustrate the general idea behind the proof of Theorem 4.6, we demonstrate how to show Delhommé's characterisation of the tree-automatic ordinals. For the purpose of later reuse, the proof of the if-part is slightly more involved than actually necessary. For the converse implication, let $\mathcal{C}$ be the class of ordinals and $\mathrm{rk}(\alpha) = \alpha$. Results by Caruth [3] imply that ω^{α} is rk-sum-indecomposable and $\omega^{\omega^{\alpha}}$ is rk-box-indecomposable for each ordinal α.

▶ **Corollary 3.9** (Delhommé [4]). *An ordinal α is tree-automatic if, and only if, $\alpha < \omega^{\omega^{\omega}}$.*

Proof. We first show the if-part. There exists a $k \in \mathbb{N}$ such that $\alpha < \omega^{\omega^{k}}$. For $k = 0$ the claim is trivial. Suppose $k \geq 1$. Then $\alpha < \omega^{\omega^{k-1} n}$ for some $n \in \mathbb{N}$. We show that $\omega^{\omega^{k-1}}$ is tree-automatic by induction on k. The case $k = 1$ is obvious. For $k > 1$ we combine the fact $\omega^{\omega^{k-1}} = \big(\omega^{\omega^{k-2}}\big)^{\omega}$ with the general idea behind showing that the class of tree-automatic structures is closed under direct ω-sums. Encoding $\bar{\beta} \in \big(\omega^{\omega^{k-1}}\big)^{n}$ by $\otimes\bar{\beta}$ shows that $\omega^{\omega^{k-1} n}$ is also tree-automatic. Finally, $\big(\omega^{\omega^{k-1} n}\big){\restriction}\phi, \alpha = \alpha$ with $\phi(x, y) = x < y$ proves the claim.

For the sake of a contradiction to the only-if-implication, assume that $\alpha \geq \omega^{\omega^{\omega}}$ is tree-automatic. For each $d \in \mathbb{N}$ we have $\alpha{\restriction}\phi, \omega^{\omega^{d}} = \omega^{\omega^{d}}$, contradicting Corollary 3.8. ◀

4 Tree-Automatic Linear Orderings

The ultimate goal of this section is to prove Theorem 4.6, stating that the FC-rank of every tree-automatic linear ordering is below ω^{ω}. Owing to the fact that every linear ordering is a dense sum of scattered linear orderings, the strategy is to apply Corollary 3.8 to the class $\mathcal{VD}$ of scattered linear orderings ranked by VD_*, a slight variation of the VD-rank. Since it is already known that every ordinal is VD_*-sum-indecomposable [9], the main difficulty is to identify the VD_*-tame-box-indecomposable ordinals.

▶ **Definition 4.1.** The VD_**-rank* of a scattered linear ordering $\mathfrak{A}$, denoted by $\mathrm{VD}_*(\mathfrak{A})$, is the least ordinal α such that $\mathfrak{A}$ is a finite sum of elements from $\mathcal{VD}_{\alpha}$.

The VD_*-rank of a scattered linear ordering $\mathfrak{A}$ is closely related to its VD-rank by the inequality $\mathrm{VD}_*(\mathfrak{A}) \leq \mathrm{VD}(\mathfrak{A}) \leq \mathrm{VD}_*(\mathfrak{A}) + 1$. For each $B \subseteq A$ we have $\mathrm{VD}(\mathfrak{A}{\restriction}B) \leq \mathrm{VD}(\mathfrak{A})$

[12, Lemma 5.14]. In particular, this implies $\mathrm{VD}_*(\mathfrak{A}{\upharpoonright}B) \leq \mathrm{VD}_*(\mathfrak{A})$ for all $B \subseteq A$. Remember that whenever a scattered linear ordering $\mathfrak{A}$ is a sum- or box-augmentation of $\mathfrak{B}_1, \ldots, \mathfrak{B}_n$ the $\mathfrak{B}_i$ are also scattered linear orderings (cf. Examples 3.2 and 3.4). The following proposition essentially states that every countable ordinal is VD_*-sum-indecomposable.

▶ **Proposition 4.2** (Khoussainov, Rubin, Stephan [9, Proposition 4.4]). *Let a scattered linear ordering $\mathfrak{A}$ be a sum-augmentation of $\mathfrak{B}_1, \ldots, \mathfrak{B}_n$. Then*

$$\mathrm{VD}_*(\mathfrak{A}) = \max\bigl\{\mathrm{VD}_*(\mathfrak{B}_1), \ldots, \mathrm{VD}_*(\mathfrak{B}_n)\bigr\} .$$

Our main tool for identifying the VD_*-tame-box-indecomposable ordinals is Proposition 4.3 below. Let α and β be two ordinals. Due to Cantor normal form, there are ordinal exponents $\gamma_1 > \cdots > \gamma_n \geq 0$ and coefficients $k_i, \ell_i \in \mathbb{N}$, which are possibly 0, such that $\alpha = \sum_{i=1}^{i=n} \omega^{\gamma_i} k_i$ and $\beta = \sum_{i=1}^{i=n} \omega^{\gamma_i} \ell_i$. The *natural sum* of α and β is $\alpha \oplus \beta = \sum_{i=1}^{i=n} \omega^{\gamma_i}(k_i + \ell_i)$. Compared to the usual addition of ordinals, this operation is commutative and strictly monotonic in both arguments.

▶ **Proposition 4.3.** *Let the scattered linear ordering $\mathfrak{A}$ be a tame box-augmentation of $\mathfrak{B}_1, \ldots, \mathfrak{B}_n$. Then*

$$\mathrm{VD}_*(\mathfrak{A}) \leq \mathrm{VD}_*(\mathfrak{B}_1) \oplus \cdots \oplus \mathrm{VD}_*(\mathfrak{B}_n) .$$

The proof below reveals the main benefit of tameness: box-augmentations are opened to arguments using *Ramsey's theorem.* It proceeds by induction on n, reducing to case $n = 2$.

▶ **Lemma 4.4.** *Let the scattered linear ordering $\mathfrak{A}$ be a tame box-augmentation of $\mathfrak{B}, \mathfrak{C}$. Then*

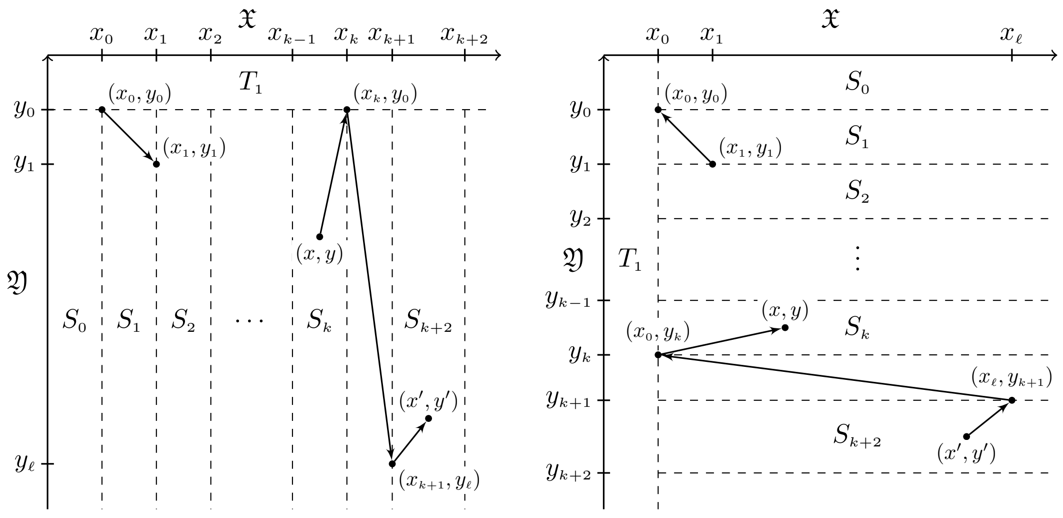
$$\mathrm{VD}_*(\mathfrak{A}) \leq \mathrm{VD}_*(\mathfrak{B}) \oplus \mathrm{VD}_*(\mathfrak{C}) .$$

Proof. We proceed by induction on $\beta = \mathrm{VD}_*(\mathfrak{B})$ and $\gamma = \mathrm{VD}_*(\mathfrak{C})$. If $\beta = 0$, then $\mathfrak{B}$ is finite, $\mathfrak{A}$ a sum-augmentation of $|B|$ many copies of $\mathfrak{C}$, and $\mathrm{VD}_*(\mathfrak{A}) = \mathrm{VD}_*(\mathfrak{C})$ by Proposition 4.2. Similarly, $\mathrm{VD}_*(\mathfrak{A}) = \mathrm{VD}_*(\mathfrak{B})$ whenever $\gamma = 0$. Thus, suppose $\beta > 0$ and $\gamma > 0$.

Due to Example 3.4, we may assume that $\mathfrak{A}$ is a linearisation of $\mathfrak{B} \times \mathfrak{C}$. In particular, $A = B \times C$. By definition, $\mathfrak{B} = \mathfrak{B}_1 + \cdots + \mathfrak{B}_m$ and $\mathfrak{C} = \mathfrak{C}_1 + \cdots + \mathfrak{C}_n$ for some $\mathfrak{B}_i \in \mathcal{VD}_\beta$ and $\mathfrak{C}_j \in \mathcal{VD}_\gamma$. Since every ζ-sum can be split into an ω^*-sum and an ω-sum, we can assume that none of the $\mathfrak{B}_i$ or $\mathfrak{C}_j$ is constructed as a ζ-sum. By Proposition 4.2, it suffices to show $\mathrm{VD}_*\bigl(\mathfrak{A}{\upharpoonright}(B_i \times C_j)\bigr) \leq \beta \oplus \gamma$ for all $i \in [1, m]$ and $j \in [1, n]$. Therefore, fix i and j, and let $\mathfrak{Z} = \mathfrak{A}{\upharpoonright}(B_i \times C_j)$, $\mathfrak{X} = \mathfrak{B}_i$, and $\mathfrak{Y} = \mathfrak{C}_j$. Notice that $\mathfrak{Z}$ is a tame box-augmentation of $\mathfrak{X}, \mathfrak{Y}$.

If $\mathfrak{X}$ is a finite sum of elements from $\mathcal{VD}_{<\beta}$, then $\mathrm{VD}_*(\mathfrak{X}) < \beta$ and the claim follows by induction. The case of a finite sum $\mathfrak{Y}$ is analogous. Thus, we assume that $\mathfrak{X}$ and $\mathfrak{Y}$ are ω- or ω^*-sums. We only demonstrate the case $\mathfrak{X} = \sum_{k\in\omega} \mathfrak{X}_k$ with $\mathfrak{X}_k \in \mathcal{VD}_{<\beta}$ and $\mathfrak{Y} = \sum_{\ell\in\omega^*} \mathfrak{Y}_\ell$ with $\mathfrak{Y}_\ell \in \mathcal{VD}_{<\gamma}$, for the remaining cases are very similar.

There are finite colourings σ of $\mathfrak{X}$ and σ' of $\mathfrak{Y}$ inducing a finite colouring of $\mathfrak{Z}$. Using *Ramsey's theorem*, we find an unbounded infinite sequence $x_0 < x_1 < x_2 < \cdots$ in $\mathfrak{X}$ which is *monochromatic* w.r.t. σ, i.e., $\sigma(x_k, x_{k'})$ is the same colour for all $k < k'$. Similarly, we find an unbounded infinite sequence $y_0 > y_1 > y_2 > \cdots$ in $\mathfrak{Y}$ which is monochromatic w.r.t. σ'. Depending on how (x_0, y_0) and (x_1, y_1) are ordered in $\mathfrak{Z}$, we distinguish two cases. As they are very similar we only deal with the case $(x_0, y_0) < (x_1, y_1)$, whose treatment is sketched in Figure 1(a). The horizontal axis depicts $\mathfrak{X}$ and increases from left to right, whereas the vertical axis outlines $\mathfrak{Y}$ and grows upwards. Within the grid, arrows point from lesser to greater elements. Figure 1(b) sketches the case $(x_0, y_0) > (x_1, y_1)$.


$\mathfrak{X}$
x_0 x_1 x_2 x_{k-1} x_k x_{k+1} x_{k+2}
$\mathfrak{Y}$
y_0 y_1 y_ℓ
T_1
(x_0, y_0) (x_1, y_1) (x_k, y_0) (x, y) (x', y') (x_{k+1}, y_ℓ)
S_0 S_1 S_2 $\cdots$ S_k S_{k+2}
$\mathfrak{X}$
x_0 x_1 x_ℓ
$\mathfrak{Y}$
y_0 y_1 y_2 y_{k-1} y_k y_{k+1} y_{k+2}
T_1
(x_0, y_0) (x_1, y_1) (x, y) (x_0, y_k) (x_ℓ, y_{k+1}) (x', y')
S_0 S_1 S_2 $\vdots$ S_k S_{k+2}

(a) The case $(x_0, y_0) < (x_1, y_1)$. **(b)** The case $(x_0, y_0) > (x_1, y_1)$.

Figure 1 Proof sketch for the inductive step of Lemma 4.4.

We partition the set $Z = X \times Y$ into sets $S_0, S_1, S_2, \ldots$ and T_1 as indicated in Figure 1(a).[4] For each $k \in \mathbb{N}$ there exists an $k' \in \mathbb{N}$ such that $S_k \subseteq (X_0 \cup \cdots \cup X_{k'}) \times Y$. Since $\mathrm{VD}_*(\mathfrak{X}_0 + \cdots + \mathfrak{X}_{k'}) < \beta$, the induction hypothesis yields $\mathrm{VD}_*(\mathfrak{Z}{\restriction}S_k) < \beta \oplus \gamma$. Similarly, we obtain $\mathrm{VD}_*(\mathfrak{Z}{\restriction}T_1) < \beta \oplus \gamma$. The right part of Figure 1(a) sketches how to show $S_k \ll S_{k+2}$ for all $k \in \mathbb{N}$. Therein $(x_k, y_0) < (x_{k+1}, y_\ell)$ follows from $(x_0, y_0) < (x_1, y_1)$ due to *monochromaticity and tameness*. As a consequence, we obtain $\mathfrak{Z}{\restriction}T_2 = \sum_{k \in \omega} \mathfrak{Z}{\restriction}S_{2k}$ for $T_2 = \bigcup_{k \in \mathbb{N}} S_{2k}$. Since every $\mathfrak{Z}{\restriction}S_{2k}$ is a finite sum of elements from $\mathcal{VD}_{<\beta\oplus\gamma}$, $\mathfrak{Z}{\restriction}T_2$ is an ω-sum of elements from $\mathcal{VD}_{<\beta\oplus\gamma}$. Thus, $\mathrm{VD}_*(\mathfrak{Z}{\restriction}T_2) \leq \beta \oplus \gamma$. Analogously, $\mathrm{VD}_*(\mathfrak{Z}{\restriction}T_3) \leq \beta \oplus \gamma$ for $T_3 = \bigcup_{k \in \mathbb{N}} S_{2k+1}$. Finally, Proposition 4.2 and $Z = T_1 \uplus T_2 \uplus T_3$ imply $\mathrm{VD}_*(\mathfrak{A}) \leq \beta \oplus \gamma$. ◀

Proof of Proposition 4.3. We proceed by induction on n. The case $n = 1$ is obvious. Thus, consider $n > 1$. We assume that $\mathfrak{A}$ is a linearisation of $\mathfrak{B}_1 \times \cdots \times \mathfrak{B}_n$. There are finite colourings $\sigma_i \colon B_i \times B_i \to Q_i$ of each $\mathfrak{B}_i$ which induce a finite colouring of $\mathfrak{A}$. For each $q \in Q_1$ we put $X_q = \{\, x \in B_1 \mid \sigma_1(x, x) = q \,\}$, fix some $x_q \in X_q$, and let $\mathfrak{C}_q = \mathfrak{A}{\restriction}(\{x_q\} \times Y)$, where $Y = B_2 \times \cdots \times B_n$. Straightforward arguments show that $\mathfrak{C}_q$ is a tame box-augmentation of $\mathfrak{B}_2, \ldots, \mathfrak{B}_n$ and $\mathfrak{A}{\restriction}(X_q \times Y)$ is a tame box-augmentation of $\mathfrak{B}_1{\restriction}X_q, \mathfrak{C}_q$. Lemma 4.4 and the induction hypothesis yield

$$\mathrm{VD}_*\big(\mathfrak{A}{\restriction}(X_q \times Y)\big) \leq \mathrm{VD}_*(\mathfrak{B}_1{\restriction}X_q) \oplus \mathrm{VD}_*(\mathfrak{C}_q) \leq \mathrm{VD}_*(\mathfrak{B}_1) \oplus \bigoplus_{i \in [2,n]} \mathrm{VD}_*(\mathfrak{B}_i)\,.$$

Finally, $A = \biguplus_{q \in Q_1} X_q \times Y$ and Proposition 4.2 imply the claim. ◀

▶ **Corollary 4.5.** *Every countable ordinal of the shape ω^α is VD_*-tame-box-indecomposable.*

Proof. Let a scattered linear ordering $\mathfrak{A}$ with $\mathrm{VD}_*(\mathfrak{A}) = \omega^\alpha$ be a tame box-augmentation of $\mathfrak{B}_1, \ldots, \mathfrak{B}_n$. Each $\mathfrak{B}_i$ can be embedded into $\mathfrak{A}$ and thus $\mathrm{VD}_*(\mathfrak{B}_i) \leq \omega^\alpha$. If this inequality were strict for each $i \in [1, n]$, the definition of $\oplus$ would imply $\mathrm{VD}_*(\mathfrak{B}_1) \oplus \cdots \oplus \mathrm{VD}_*(\mathfrak{B}_n) < \omega^\alpha$, contradicting Proposition 4.3. ◀

[4] It does not matter to which of the adjacent sets the dashed lines belong.

Using the previous results, we prove our main result on tree-automatic linear orderings.

▶ **Theorem 4.6.** *The* FC*-rank of every tree-automatic linear ordering is strictly below* ω^ω.

Proof. For the sake of a contradiction, assume there exists a tree-automatic linear ordering $\mathfrak{A}$ with $\mathrm{FC}(\mathfrak{A}) \geq \omega^\omega$. Consider the formula $\phi(x, y_1, y_2) = y_1 \leq x \wedge x \leq y_2$. Due to the proof of [9, Proposition 4.5], for each $d \in \mathbb{N}$ there is a $\bar{s} \in A^2$ such that the closed interval $\mathfrak{I} = \mathfrak{A}{\restriction}\phi, \bar{s}$ is scattered and $\mathrm{VD}(\mathfrak{I}) = \omega^d + 1$. As $\mathfrak{I}$ contains least and greatest element, it is a finite sum of elements from $\mathcal{VD}_{<\omega^d+1} = \mathcal{VD}_{\omega^d}$ and hence $\mathrm{VD}_*(\mathfrak{I}) = \omega^d$. Since ω^d is VD_*-sum- and VD_*-tame-box-indecomposable, this contradicts Corollary 3.8. ◀

5 Finite-Rank Tree-Automatic Linear Orderings

In this section we reintroduce finite-rank tree-automatic structures [1] and investigate the linear orderings among them. The highlight is Theorem 5.2 which states that the FC-rank of every rank-k tree-automatic linear ordering is below ω^k.

5.1 Finite-Rank Tree-Automatic Structures

A *binary tree* is a (possibly empty or infinite) prefix-closed subset $T \subseteq \mathbf{2}^\star$ whose elements are considered to be ordered by the prefix relation $\preceq$. The (isomorphism type of the) *subtree* rooted at $u \in T$ is the binary tree $T{\restriction}u = \{\, v \in \mathbf{2}^\star \mid uv \in T \,\}$. We call T *regular* if it is a regular language, or due to the Myhill-Nerode theorem equivalently, if there are only finitely many distinct subtrees $T{\restriction}u$. To every tree language $L \subseteq T_\Sigma$ we assign the binary tree $T(L) = \bigcup_{t\in L} \mathrm{dom}(t)$, which is effectively regular when L is regular.

An *infinite branch* of T is a prefix-closed infinite subset $P \subseteq T$ which is linearly ordered by $\preceq$. The *derivative* of T is the binary tree $d(T)$ consisting of all $u \in T$ which are contained in at least two distinct infinite branches. This operation effectively preserves regularity. When T is regular, $d^{(n)}(T)$ is finite for some $n \in \mathbb{N}$, where $d^{(n)}$ denotes the n-fold application of d, precisely if the full binary tree $\mathbf{2}^\star$ cannot be embedded into T [9, Section 7]. If these equivalent conditions are satisfied, the *rank* of T is the least such $n \in \mathbb{N}$.[5]

▶ **Definition 5.1.** Let $k \in \mathbb{N}$. A tree-automatic structure $\mathfrak{A}$ is *rank-k tree-automatic* if the rank of $T(A)$ is at most k and *finite-rank tree-automatic* if the rank of $T(A)$ exists.[6]

▶ Remark. For a tree-automatic structure $\mathfrak{A}$ the rank of $T(A)$ is not an isomorphism invariant property, but depends on its specific representation as a tree-automatic structure. The rank of $T(A)$ is computable from a tree automaton recognising A. It can be shown that the rank-1 tree-automatic structures are precisely those which are isomorphic to a string-automatic structure.

5.2 Linear Orderings

Theorem 5.2 is our main result on finite-rank tree-automatic linear orderings. Basically, it is shown by adapting Theorem 4.6's proof. The key idea behind this adaption is provided by Lemma 5.3 below.

[5] This rank is a slight variation of the Cantor-Bendixson rank for trees introduced in [9].
[6] The definition of rank-k tree-automatic structures in [1, Section 1.3.7] is different, but equivalent.

▶ **Theorem 5.2.** *Let $k \in \mathbb{N}_+$. The* FC*-rank of every rank-k tree-automatic linear ordering is strictly below ω^k.*

▶ **Lemma 5.3.** *Let T be a regular binary tree of rank $k \in \mathbb{N}$. Then there exists a constant $K \in \mathbb{N}$ such that any anti-chain (w.r.t. $\preceq$) $A \subseteq T$ contains at most K elements u such that $T{\restriction}u$ has rank k.*

Proof. We proceed by induction on k. If $k = 0$, then T is finite and the claim is obvious. Thus, suppose $k > 0$. Let $n \in \mathbb{N}$ be the *index* of T, i.e., the size of the set $\{\, T{\restriction}u \mid u \in T \,\}$.

For the sake of a contradiction, assume there is an anti-chain A consisting of $2^n + 1$ elements $u \in T$ such that $T{\restriction}u$ has rank k. The set B of all $v \in T$ which are the longest common prefix of two distinct elements from A contains exactly 2^n elements. For every $u \in A$ the binary tree $d^{(k-1)}(T{\restriction}u) = d^{(k-1)}(T){\restriction}u$ is infinite and hence, by König's lemma, there is an infinite branch of $d^{(k-1)}(T)$ containing u. Thus, $B \subseteq d^{(k)}(T)$ and $\left|d^{(k)}(T)\right| \geq 2^n$. Since $d^{(k)}(T){\restriction}v = d^{(k)}(T{\restriction}v)$ for all $v \in d^{(k)}(T)$, the index of $d^{(k)}(T)$ is at most n. Finally, a pumping argument shows that $d^{(k)}(T)$ is infinite, contradicting the choice of k. ◀

Proof of Theorem 5.2. We proceed by induction on $k \geq 1$, using an artificial base case $k = 0$. A rank-0 tree-automatic scattered linear ordering $\mathfrak{A}$ is finite and hence satisfies $\mathrm{VD}_*(\mathfrak{A}) < \omega^0$. Thus, consider $k \geq 1$.

For the sake of a contradiction, assume there exists a rank-k tree-automatic linear ordering $\mathfrak{A}$ with $\mathrm{FC}(\mathfrak{A}) \geq \omega^k$. Let $\mathcal{S}_\phi^{\mathfrak{A}}$ be the set constructed in Theorem 3.7's proof from $\mathfrak{A}$ and the formula $\phi(x, y_1, y_2) = y_1 \leq x \wedge x \leq y_2$. We show that $\mathcal{S}_\phi^{\mathfrak{A}}$ contains for each $n \in \mathbb{N}$ a scattered linear ordering $\mathfrak{B}$ with $\omega^{k-1}n < \mathrm{VD}_*(\mathfrak{B}) < \omega^k$, contradicting the finiteness of $\mathcal{S}_\phi^{\mathfrak{A}}$.

Consider $n \in \mathbb{N}$ and let K be the constant which exists by Lemma 5.3 applied to $T(A)$. Like in Theorem 4.6's proof there is a $\bar{s} \in A^2$ such that $\mathfrak{A}{\restriction}\phi, \bar{s}$ is scattered and $\mathrm{VD}_*(\mathfrak{A}{\restriction}\phi, \bar{s}) = \omega^{k-1}(nK + 1)$. We delve into the details of Theorem 3.7's proof, supposing we have fixed $\bar{s}$. Since $\omega^{k-1}(nK + 1)$ is VD_*-sum-indecomposable, there exists a $\sim_{\bar{s}}$-class $C \subseteq B$ such that $\mathrm{VD}_*(\mathfrak{C}) = \omega^{k-1}(nK + 1)$. Let $\tau = \left(t_0, U, q_\leq, q_\phi\right)$ be the corresponding $\bar{s}$-type. For each $u \in U$ we have $T\left(S_{\gamma(u)}\right) \subseteq T(A){\restriction}u$ and hence the rank of $T\left(S_{\gamma(u)}\right)$ is at most k. Let V be the set of those $u \in U$ for which the rank equals k. Due to Lemma 5.3, $|V| \leq K$. The induction hypothesis yields $\mathrm{VD}_*\left(\mathfrak{S}_{\gamma(u)}\right) < \omega^{k-1}$ for $u \in U \setminus V$. If we had $\mathrm{VD}_*\left(\mathfrak{S}_{\gamma(u)}\right) \leq \omega^{k-1}n$ for each $u \in V$, this would imply

$$\underbrace{\bigoplus_{u \in V} \mathrm{VD}_*\left(\mathfrak{S}_{\gamma(u)}\right)}_{\leq \omega^{k-1}n|V|} \oplus \underbrace{\bigoplus_{u \in U \setminus V} \mathrm{VD}_*\left(\mathfrak{S}_{\gamma(u)}\right)}_{< \omega^{k-1}} < \omega^{k-1}n|V| \oplus \omega^{k-1} \leq \omega^{k-1}(nK + 1)\,,$$

contradicting Proposition 4.3. Hence, there exists a $u \in V$ with $\mathrm{VD}_*\left(\mathfrak{S}_{\gamma(u)}\right) > \omega^{k-1}n$. Since $\mathfrak{S}_{\gamma(u)}$ can be embedded into $\mathfrak{C}$, we also have $\mathrm{VD}_*\left(\mathfrak{S}_{\gamma(u)}\right) \leq \omega^{k-1}(nK + 1) < \omega^k$. ◀

For ordinals α and β we have $\mathrm{FC}(\alpha) \leq \beta$ precisely if $\alpha \leq \omega^\beta$. Consequently, Theorem 5.2 implies that every rank-k tree-automatic ordinal is less than ω^{ω^k}. In fact, the construction in Corollary 3.9's proof shows that the converse implication holds as well. Therefore, we obtain the following analogue to Corollary 3.9.

▶ **Corollary 5.4.** *Let $k \in \mathbb{N}$. An ordinal α is rank-k tree-automatic if, and only if, $\alpha < \omega^{\omega^k}$.*

6 Discussion

As an application of the FC-rank of string-automatic linear orderings being finite, it was shown that the Cantor-Bendixson rank of string-automatic order trees is also finite [9]. The proof uses that $\Sigma^\star$ admits an automatic linear order isomorphic to ω, but results in [5] imply this to fail for T_Σ. However, the arguments in [9] carry over for *finitely branching* order trees since T_Σ admits at least some automatic linear order. Thus, the Cantor-Bendixson rank of every (rank-k) tree-automatic finitely branching order tree is below ω^ω respectively ω^k.

Another application of Theorem 4.6 was pointed out by Kuske [10]. Results in [11] can be adapted to show that the isomorphism problem for tree-automatic scattered linear orderings, that he proved to be Π^0_1-hard, belongs to level $\Delta^0_{\omega^\omega}$ of the hyperarithmetical hierarchy.

As a sideline, Corollary 5.4 reproves that the hierarchy of finite-rank tree-automatic structures is strict, a fact whose proof yet depended on deep logical insights [1, Section 1.3.7]. Moreover, it implies that any tree-automatic well-ordering is already finite-rank tree-automatic. However, the respective question for arbitrary linear orderings remains open.

All results present upper bounds on the FC-rank. Unfortunately, for each $k \in \mathbb{N}$ there exists a tree-automatic well-ordering $\mathfrak{A}$ isomorphic to ω^k such that $T(A)$ has rank k. This renders it impossible to provide useful lower bounds in terms of the rank of $T(A)$. Using another approach, a first step in this direction provides a decidable characterisation of the tree-automatic scattered linear orderings of FC-rank at least ω [6].

Finally, it is known that the ordinal height of string-automatic well-founded partial orderings is below ω^ω [4] and of tree-automatic well-founded order trees below ω^{ω^ω} [7]. Regretfully, even the refined decomposition technique seems too weak to verify the resulting conjecture that the height of tree-automatic well-founded partial orderings is below ω^{ω^ω}.

Acknowledgement I thank Alexander Kartzow for valuable discussions on Delhommé's decomposition technique.

References

1 V. Bárány, E. Grädel, and S. Rubin. Automata-based presentations of infinite structures. In *Finite and Algorithmic Model Theory*. Cambridge University Press, 2011.

2 A. Blumensath. Automatic structures. Diploma thesis, RWTH Aachen, 1999.

3 P. W. Carruth. Arithmetic of ordinals with applications to the theory of ordered abelian groups. *Bull. Amer. Math. Soc.*, 48:262–271, 1942.

4 C. Delhommé. Automaticité des ordinaux et des graphes homogènes. *Comptes Rendus Mathematique*, 339(1):5–10, 2004.

5 Y. Gurevich and S. Shelah. Rabin's uniformization problem. *Journal of Symbolic Logic*, 48(4):1105–1119, 1983.

6 M. Huschenbett. Word automaticity of tree automatic scattered linear orderings is decidable. In *CiE 2012*, volume 7318 of *LNCS*, pages 313–322. Springer, 2012.

7 A. Kartzow, J. Liu, and M. Lohrey. Tree-automatic well-founded trees. In *CiE 2012*, volume 7318 of *LNCS*, pages 363–373. Springer, 2012.

8 B. Khoussainov and A. Nerode. Automatic presentations of structures. In *LCC 1994*, volume 960 of *LNCS*, pages 367–392. Springer, 1994.

9 B. Khoussainov, S. Rubin, and F. Stephan. Automatic linear orders and trees. *ACM Transactions on Computional Logic*, 6(4):675–700, 2005.

10 D. Kuske. Isomorphisms of scattered automatic linear orders. In *CSL 2012*, pages 455–469, 2012.

11 D. Kuske, J. Liu, and M. Lohrey. The isomorphism problem on classes of automatic structures with transitive relations. *Transactions of the AMS*, 2011. accepted.

12 J. G. Rosenstein. *Linear Orderings*. Academic Press, 1982.

13 S. Rubin. Automata presenting structures: A survey of the finite string case. *Bulletin of Symbolic Logic*, 14(2):169–209, 2008.

A general framework for the realistic analysis of sorting and searching algorithms. Application to some popular algorithms*

Julien Clément, Thu Hien Nguyen Thi, and Brigitte Vallée

Université de Caen / ENSICAEN / CNRS - GREYC - Caen, France

Abstract

We describe a general framework for realistic analysis of sorting and searching algorithms, and we apply it to the average-case analysis of five basic algorithms: three sorting algorithms (*QuickSort*, *InsertionSort*, *BubbleSort*) and two selection algorithms (*QuickMin* and *SelectionMin*). Usually, the analysis deals with the mean number of key comparisons, but, here, we view keys as words produced by the same source, which are compared via their symbols in the lexicographic order. The "realistic" cost of the algorithm is now the total number of symbol comparisons performed by the algorithm, and, in this context, the average–case analysis aims to provide estimates for the mean number of symbol comparisons used by the algorithm. For sorting algorithms, and with respect to key comparisons, the average-case complexity of *QuickSort* is asymptotic to $2n \log n$, *InsertionSort* to $n^2/4$ and *BubbleSort* to $n^2/2$. With respect to symbol comparisons, we prove that their average-case complexity becomes $\Theta(n \log^2 n), \Theta(n^2), \Theta(n^2 \log n)$. For selection algorithms, and with respect to key comparisons, the average-case complexity of *QuickMin* is asymptotic to $2n$, of *SelectionMin* is $n - 1$. With respect to symbol comparisons, we prove that their average-case complexity remains $\Theta(n)$. In these five cases, we describe the dominant constants which exhibit the probabilistic behaviour of the source (namely, entropy, and various notions of coincidence) with respect to the algorithm.

1998 ACM Subject Classification F2.2: Pattern matching, sorting and searching – G2.1: Generating functions, permutations – G4: Algorithm design and analysis – H1.1: Information theory – I1.2: Analysis of algorithms

Keywords and phrases Probabilistic analysis of algorithms – Sorting and searching algorithms – Pattern matching – Permutations – Information theory – Rice formula – Asymptotic estimates

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.598

Introduction

There are two main classes of sorting and searching algorithms: the first class gathers the algorithms which deal with keys, while the algorithms of the second class deal with words (or strings). Of course, any data is represented inside a computer as a sequence of bits (that is a binary string). However, the point of view is different: the key is viewed as a "whole", and its precise representation is not taken into account, whereas the structure of a word, as a sequence of symbols, is essential in text algorithms. Hence, for basic algorithms of the first class (sorting, searching), the unit operation is the comparison between keys, whereas for text algorithms of the second class, comparisons between symbols are considered.

* Thanks to the two ANR Projects: ANR BOOLE (ANR 2009 BLAN 0011) and ANR MAGNUM (ANR 2010 BLAN 0204).

There exist two important drawbacks to this usual point of view. First, it is difficult to compare algorithms belonging to these two different classes, since they are analyzed with respect to different costs. Second, when the keys are complex items, not reduced to single machine words, it is not realistic to consider the total cost of their comparison as unitary. This is why Sedgewick proposed in 1998 to analyze basic algorithms (sorting and searching) when dealing with words rather than with "atomic" keys; in this case, the realistic cost for comparing two words is the number of symbols comparisons needed to distinguish them in the lexicographic order and is closely related to the length of their longest common prefix, called here the *coincidence*. There are two factors which influence the efficiency of such an algorithm: the strategy of the algorithm itself (*which words are compared?*) and the mechanism which produces words, called the source (*what makes two words distinguishable?*).

The first results in the area are due to Fill and Janson [5], Fill and Nakama [6], who dealt with data composed of random uniform bits. Then, in the paper [18], a general framework towards a realistic analysis based on the number of symbol comparisons is provided, when the source which emits symbols is (almost completely) general. Furthermore, these principles are applied to two algorithms, QuickSort and QuickSelect. Later on, a study of the distribution of the complexity was performed in the same framework [4, 7].

Main results. The present paper follows the lines of the article [18], and works within the same general framework, with four specific aims:

(a) The general method has been already described in [18]: it was shown that a Dirichlet series denoted by $\varpi(s)$ characterizes the behavior of an algorithm with respect to the source. We wish here to highlight the main principles, in order to make easier its application to various algorithms. As it is often the case in analytical combinatorics, there are two main phases in the method, a first phase where the series $\varpi(s)$ is built, and a second phase where it is analyzed. We note here that the first phase may mostly be performed in an "automatic" way.

(b) We apply the method to three other popular algorithms: InsertionSort, BubbleSort and SelectionMinimum, respectively denoted in the sequel by the short names `InsSort`, `BubSort`, `SelMin` (see for instance the book [15] for a thorough description of these algorithms). With this approach we also easily recover the results about algorithms `QuickSort` and `QuickMin` already obtained in [18]. Thus we provide an unified framework for the analysis of these five algorithms in Section 2.2.

(c) We exhibit in each case the probabilistic features of the source which play a role in the analysis: each algorithm of interest is related to a particular constant of the source, which describes the interplay between the algorithm and the source, and explains how the efficiency of the algorithm depends on the source, via various notions of coincidence between words (See Proposition 5). This type of coincidence provides a good characterization of the algorithm, and our study is a tool for a better understanding of the algorithmic strategy.

(d) We discuss the robustness of the algorithms, i.e., the possible changes in the complexity behaviors, due to the change in the complexity measure, from the number of key comparisons to the number of symbol comparisons (see Discussion p. 607).

Plan of the paper. Section 1 first presents the general method, with its main steps. Then, Section 2 states the main results.

Most of the proofs and technical details are omitted due to space constraints. The full version of this paper will include them.

1 Main steps for the "realistic" analysis of a sorting algorithm

Here, we describe our general framework, already provided in [18]. We insist on the main steps, and the notions developed here are somewhat different from the previous paper. We first characterize in Section 1.1 the strategy of the algorithm (which keys are compared? with which probability?), then we describe the source, and the central notion of coincidence (Sections 1.2 and 1.3). We obtain an exact formula for the mean number of symbol comparisons, which involves the mixed Dirichlet series $\varpi(s)$ (depending on the source *and* the algorithm) introduced in Section 1.4 and 1.5. In order to obtain asymptotic estimates, we deal with tameness properties of the source, which entail tameness for the series $\varpi(s)$, and finally the asymptotic estimates (Sections 1.6 and 1.7).

1.1 The classical probabilistic model: permutations and arrival times

Consider a totally ordered set of keys $\mathcal{U} = \{U_1 < U_2 < \cdots < U_n\}$ and any algorithm $\mathcal{A}$ which only performs comparisons and exchanges between keys. The initial input is the sequence $(V_1, V_2, \ldots, V_n)$ defined from $\mathcal{U}$ by the permutation $\sigma \in \mathfrak{S}_n$ via the equalities $V_i = U_{\sigma(i)}$. The execution of the algorithm does not actually depend on the input sequence, but only on the permutation σ which defines the input sequence from the final (ordered) sequence. Then, the permutation σ is the actual input of the algorithm and the set of all possible inputs is the set $\mathfrak{S}_n$ (usually endowed with the uniform distribution).

The strategy of the algorithm $\mathcal{A}$ defines, for each pair (i, j), with $1 \le i < j \le n$, the subset of $\mathfrak{S}_n$ which gathers the permutations σ (or the arrival times) for which U_i and U_j are compared by the algorithm $\mathcal{A}$, when the input sequence is $(U_{\sigma(1)}, U_{\sigma(2)}, \ldots, U_{\sigma(n)})$. For efficient algorithms, the two keys U_i and U_j are compared only once, but there exist other algorithms (the `BubSort` algorithm for instance) where U_i and U_j may be compared several times. In all cases, $\pi(i, j)$ denotes the mean number of comparisons between U_i and U_j. The computation of $\pi(i, j)$ is the first step, described in Section 2.1. These mean numbers $\pi(i, j)$ are computed with direct probabilistic arguments. A remarkable feature is that the expectations $\pi(i, j)$ are always expressed as sums of rational functions depending on i, j or $j - i$.

1.2 General sources

Here, we consider that the keys are words produced by a general source. By convention, we denote open and closed intervals of real numbers $]a, b[$ and $[a, b]$, whereas (a, b) denotes a pair of real numbers.

▶ **Definition 1.** Let Σ be a totally ordered alphabet of cardinality r. A *general source* produces infinite words of $\Sigma^{\mathbb{N}}$, and is specified by the set $\{p_w, w \in \Sigma^\star\}$ of *fundamental probabilities* p_w, where p_w is the probability that an infinite word begins with the finite prefix w. It is (only) assumed that $\sup\{p_w : w \in \Sigma^k\}$ tends to 0, as $k \to \infty$.

For any prefix $w \in \Sigma^\star$, we denote by $|w|$ the length of w (i.e., the number of the symbols that it contains) and a_w, b_w, p_w the probabilities that a word produced by the source begins with a prefix α of the same length as w, which satisfies $\alpha < w$, $\alpha \le w$, or $\alpha = w$, meaning

$$a_w := \sum_{\substack{\alpha, |\alpha|=|w|, \\ \alpha < w}} p_\alpha, \qquad b_w := \sum_{\substack{\alpha, |\alpha|=|w|, \\ \alpha \le w}} p_\alpha, \qquad p_w = b_w - a_w. \tag{1}$$

Denote by $\mathcal{L}(\mathcal{S})$ the set of (infinite) words produced by the source $\mathcal{S}$, ordered via the lexicographic order. Given an infinite word $X \in \mathcal{L}(\mathcal{S})$, denote by w_k its prefix of length k.

The sequence (a_{w_k}) is increasing, the sequence (b_{w_k}) is decreasing, and $b_{w_k} - a_{w_k} = p_{w_k}$ tends to 0. Thus a unique real $P(X) \in [0,1]$ is defined as the common limit of (a_{w_k}) and (b_{w_k}), and $P(X)$ can be viewed as the probability that an infinite word Y be smaller than X. The mapping $P : \mathcal{L}(\mathcal{S}) \to [0,1]$ is strictly increasing outside the exceptional set formed with words of $\mathcal{L}(\mathcal{S})$ which end with an infinite sequence of the smallest symbol or with an infinite sequence of the largest symbol.

Conversely, almost everywhere, except on the set $\{a_w, w \in \Sigma^\star\}$, there is a mapping M which associates, to a number u of the interval $\mathcal{I} := [0,1]$, a word $M(u) \in \mathcal{L}(\mathcal{S})$. Hence the probability that a word Y be smaller than $M(u)$ equals u. The lexicographic order on words is then compatible with the natural order on the interval $\mathcal{I}$. The interval $\mathcal{I}_w := [a_w, b_w]$, of length p_w, gathers (up to a denumerable set) all the reals u for which $M(u)$ begins with the finite prefix w. This is the fundamental interval of the prefix w.

1.3 Coincidence

Here, we are interested in a more realistic cost related to the number of symbol comparisons performed by these algorithms, when the keys are words independently produced by the same source. The words are ordered with respect to the lexicographic order, and the cost for comparing two words (measured as the number of symbol comparisons needed) is closely related to the coincidence, defined as follows.

▶ **Definition 2.** The *coincidence function* $\gamma(u,t)$ is the length of the largest common prefix of $M(u)$ and $M(t)$.

More precisely, the realistic cost of the comparison between $M(u)$ and $M(t)$ equals $\gamma(u,t)+1$. The coincidence $\gamma(u,t)$ is at least ℓ if and only if $M(u)$ and $M(t)$ have the same common prefix w of length ℓ, so that the parameters u and t belong to the same fundamental interval $\mathcal{I}_w$ relative to a prefix w of length ℓ. We thus introduce the triangles

$$\mathcal{T} := \{(u,t) : 0 \le u \le t \le 1\}, \quad \mathcal{T}_w = (\mathcal{I}_w \times \mathcal{I}_w) \cap \mathcal{T} = \{(u,t) : a_w \le u \le t \le b_w\}. \tag{2}$$

Using the two relations

$$\mathcal{T} \cap [\gamma \ge \ell] = \bigcup_{w \in \Sigma^\ell} \mathcal{T}_w, \qquad \sum_{\ell \ge 0} \mathbf{1}_{[\gamma \ge \ell]} = \sum_{\ell \ge 0} (\ell+1) \mathbf{1}_{[\gamma = \ell]},$$

the following equality holds, for any integrable function g on the unit triangle $\mathcal{T}$, and will be extensively used in the sequel,

$$\int_{\mathcal{T}} [\gamma(u,t)+1] g(u,t)\, du\, dt = \sum_{w \in \Sigma^\star} \int_{\mathcal{T}_w} g(u,t)\, du\, dt. \tag{3}$$

1.4 Average-case analysis – various models

The purpose of average–case analysis of structures (or algorithms) is to characterize the mean value of their parameters under a well-defined probabilistic model that describes the initial distribution of its inputs.

Here, we adopt the following general model for the set of inputs: we consider a finite sequence $\mathcal{V} = (V_1, \ldots, V_n)$ of infinite words independently produced by the same source $\mathcal{S}$. Such a sequence $\mathcal{V}$ is obtained by n independent drawings $v_1, v_2, \ldots, v_n$ in the interval $\mathcal{I}$ via the mapping M, and we set $V_i := M(v_i)$. We assume moreover that $\mathcal{V}$ contains two given words $M(u)$ and $M(t)$, with $u < t$. The variables $N_{[0,u[}$, $N_{[0,t[}$ respectively denote the number of words of $\mathcal{V}$ strictly less than $M(u)$, strictly less than $M(t)$. These variables define the ranks of $M(u)$ and $M(t)$ inside the set $\mathcal{V}$, via the relations, valid for $u < t$,

$$\operatorname{Rank} M(u) = N_{[0,u[} + 1, \operatorname{Rank} M(t) = N_{[0,t[} + 2,$$

where the respective translations of 1 and 2 express that $M(u)$ and $M(t)$ belong to $\mathcal{V}$.

We first consider the number of key comparisons between $M(u)$ and $M(t)$, and deal with the mean number $\widehat{\pi}(u,t)$ of key comparisons performed by the algorithm between $M(u)$ and $M(t)$, where the mean is taken with respect to all the permutations of $\mathcal{V}$. The mean number $\widehat{\pi}(u,t)$ is related to the mean number $\pi(i,j)$ via the equality

$$\widehat{\pi}(u,t) = \pi(N_{[0,u[} + 1, N_{[0,t[} + 2). \tag{4}$$

In our framework, expressions obtained for $\pi(i,j)$ ensure that $\widehat{\pi}(u,t)$ is always a sum of rational functions in variables $N_{[0,u[}$, $N_{[0,t[}$ and $N_{[u,t[}$, (with the relation $N_{[0,t[} = N_{[0,u[} + N_{]u,t[} + 1$).

When the cardinality n of $\mathcal{V}$ is fixed, and words $V_i \in \mathcal{V}$ are independently emitted by the source $\mathcal{S}$, this is the Bernoulli model denoted by $(\mathcal{B}_n, \mathcal{S})$. However, it proves technically convenient to consider that the sequence $\mathcal{V}$ has a variable number N of elements that obeys a Poisson law of rate Z,

$$\Pr\{N = k\} = e^{-Z}\frac{Z^k}{k!}. \tag{5}$$

In this model, called the Poisson model of rate Z, the rate Z plays a role much similar to the cardinality of $\mathcal{V}$. When it is relative to probabilistic source $\mathcal{S}$, the model, denoted by $(\mathcal{P}_Z, \mathcal{S})$, is composed with two main steps:

(a) The number N of words is drawn according to the Poisson law of rate Z;
(b) Then, the N words are independently drawn from the source $\mathcal{S}$.

Note that, in the Poisson model, the variables $N_{[0,u[}, N_{]u,t[}$ are themselves independent Poisson variables of parameters Zu and $Z(t-u)$ (respectively). The expectation $\widehat{\pi}(u,t)$ is itself a random variable which involves these variables.

1.5 Exact formula for the mean number of symbol comparisons

The density of the algorithm in the Poisson model, denoted by $\phi_Z(u,t)$ and defined as

$$\phi_Z(u,t)\,du\,dt = Z^2 \cdot \mathbb{E}_Z[\widehat{\pi}(u,t)]\,du\,dt = (Z\,du)\cdot(Z\,dt)\cdot\mathbb{E}_Z[\widehat{\pi}(u,t)],$$

is the mean number of key comparisons between two words $M(u')$ and $M(t')$ for $u' \in [u-du, u]$ and $t' \in [t, t+dt]$. In the model $(\mathcal{P}_Z, \mathcal{S})$, this is a main tool for computing, not only the mean number of key comparisons K_Z performed by the algorithm, but also the mean number of symbol comparisons S_Z via the formulae

$$K_Z = \int_{\mathcal{T}} \phi_Z(u,t)\,du\,dt, \qquad S_Z = \int_{\mathcal{T}} [\gamma(u,t)+1]\phi_Z(u,t)\,du\,dt.$$

To return to the Bernoulli model $(\mathcal{B}_n, \mathcal{S})$, the coefficients $\varphi(n,u,t)$ in the series expansion of $\phi_Z(u,t)$ defined as

$$\varphi(n,u,t) := (-1)^n\, n![Z^n]\phi_Z(u,t), \tag{6}$$

are computed in an "automatic way" from the mean numbers $\widehat{\pi}(u,t)$, themselves closely related to $\pi(i,j)$. This is the second step leading to results in Table 1 p. 608. Using Eq. (3), the sequence $\varphi(n)$ is now defined for any $n \geq 2$,

$$\varphi(n) := \int_{\mathcal{T}} (\gamma(u,t)+1)\,\varphi(n,u,t)\,du\,dt = \sum_{w\in\Sigma^\star}\int_{\mathcal{T}_w} \varphi(n,u,t)\,du\,dt, \tag{7}$$

and is easy to obtain via computations of the integral of $\varphi(n,u,t)$ on the triangles $\mathcal{T}_w$. Now, the mean number $S(n)$ of symbol comparisons used by the algorithm when it deals with n words independently drawn from the same source is related to $\varphi(n)$ by the equality

$$S(n) = \sum_{k=2}^{n} (-1)^k \binom{n}{k} \varphi(k), \tag{8}$$

which provides an exact formula for $S(n)$, described in Section 2.2. The expression of $S(n)$ is obtained in an "automatic" way, from the expectations $\pi(i,j)$.

1.6 Asymptotic estimates for the mean number of symbol comparisons

However, the previous formula does not give an easy or straightforward access to the asymptotic behaviour of $S(n)$ (when $n \to \infty$). In order to get asymptotic estimates, we first need an analytic lifting $\varpi(s,u,t)$ of the coefficients $\varphi(k,u,t)$, that is an analytic function $\varpi(s,u,t)$ which coincides with $\varphi(k,u,t)$ at integer values $s = k$ in the summation of Eq. (8). This analytic lifting gives rise to the mixed Dirichlet series itself,

$$\varpi(s) := \int_{\mathcal{T}} [\gamma(u,t)+1]\varpi(s,u,t)\, du\, dt = \sum_{w\in\Sigma^\star} \int_{\mathcal{T}_w} \varpi(s,u,t)\, du\, dt,$$

which depends both on the algorithm (via $\varpi(s,u,t)$) and the source (via the fundamental triangles $\mathcal{T}_w$). For each algorithm, the existence of this analytic lifting is granted in a domain $\Re s > \sigma_0$. However, the value of σ_0 depends on the algorithm. One has $\sigma_0 = 1$, except for the algorithms `InsSort` and `BubSort` where σ_0 equals 2. This is due to constant term 1/2 appearing in the expectation $\pi(i,j)$, as seen in Table 1 p. 608 (see also Section 2.2).

The Rice Formula [12, 13] transforms a binomial sum into an integral in the complex plane. For any real $\sigma_1 \in]\sigma_0, \sigma_0+1[$, one has

$$T(n) = \sum_{k=1+\sigma_0}^{n} (-1)^k \binom{n}{k} \varpi(k) = \frac{(-1)^{n+1}}{2i\pi} \int_{\Re s=\sigma_1} G(s)\, ds, \text{ with } G(s) = \frac{n!\,\varpi(s)}{s(s-1)\dots(s-n)}. \tag{9}$$

Then, along general principles in analytic combinatorics [9, 10], the integration line can be pushed to the left, as soon as $G(s)$ (closely related to $\varpi(s)$) has good analytic properties: we need a region $\mathcal{R}$ on the left of $\Re s = \sigma_0$, where $\varpi(s)$ is of polynomial growth (for $\Im s \to \infty$) and meromorphic. With a good knowledge of its poles, we finally obtain a residue formula

$$T(n) = (-1)^{n+1} \left[\sum_s \operatorname{Res}[G(s)] + \frac{1}{2i\pi} \int_{\mathcal{C}_2} G(s)\, ds \right],$$

where $\mathcal{C}_2$ is a curve of class $\mathcal{C}^1$ enclosed in $\mathcal{R}$ and the sum is extended to all poles s of $G(s)$ inside the domain delimited by the vertical line $\Re s = \sigma_1$ and the curve $\mathcal{C}_2$.

The dominant singularities of $G(s)$ provide the asymptotic behaviour of $T(n)$, and the remainder integral is estimated using the polynomial growth of $G(s)$ when $|\Im(s)| \to \infty$. According to Eq. (8) and(9), and in the cases where $\sigma_0 = 2$, we have to add to $T(n)$ the term corresponding to the index $k = 2$, where the analytical lifting ϖ does not coincides with φ. For algorithms `BubSort` and `InsSort`, the additional term is of the form $\varphi(2)\binom{n}{2}$.

1.7 Tameness of sources

We first describe three cases of possible regions $\mathcal{R}$ where good properties of $\varpi(s)$ will make possible such a shifting to the left in the Rice formula.

▶ **Definition 3.** A function $\varpi(s)$ is tame at σ_0 if one of the three following properties holds:

(a) [S–shape] (shorthand for Strip shape) there exists a vertical strip $\Re(s) > \sigma_0 - \delta$ for some $\delta > 0$ where $\varpi(s)$ is meromorphic, has a sole pole (of order $k_0 \geq 0$) at $s = \sigma_0$ and is of polynomial growth as $|\Im s| \to +\infty$.

(b) [H–shape] (shorthand for Hyperbolic shape) there exists an hyperbolic region $\mathcal{R}$, defined as, for some $A, B, \rho > 0$

$$\mathcal{R} := \{s = \sigma + it; \quad |t| \geq B, \quad \sigma > \sigma_0 - \frac{A}{t^\rho}\} \bigcup \{s = \sigma + it; \quad \sigma > \sigma_0 - \frac{A}{B^\rho}, |t| \leq B\},$$

where $\varpi(s)$ is meromorphic, with an only pole (of order $k_0 \geq 0$) at $s = \sigma_0$ and is of polynomial growth in $\mathcal{R}$ as $|\Im s| \to +\infty$.

(c) [P–shape] (shorthand for Periodic shape) there exists a vertical strip $\Re(s) > \sigma_0 - \delta$ for some $\delta > 0$ where $\varpi(s)$ is meromorphic, has only a pole (of order $k_0 \geq 0$) at $s = \sigma_0$ and a family (s_k) (for $k \in \mathbb{Z}, k \neq 0$) of simple poles at points $s_k = \sigma_0 + 2ki\pi t$ with $t \neq 0$, and is of polynomial growth as $|\Im s| \to +\infty$[1].

There are three parameters relative to the tameness: the integer k_0 is the *order*, and, when they exist, the real δ is the *abscissa*, and the real ρ is the *exponent.*

Here, the main Dirichlet series $\varpi(s)$ of interest are closely related to the Dirichlet series of the source, which involve the fundamental probabilities p_w, and the ends a_w, b_w of the fundamental intervals (see Section 1.1), via a function $F : [0,1]^2 \to \mathbb{R}^+$ of class $\mathcal{C}^1$,

$$\Lambda[F](s) := \sum_{w \in \Sigma^\star} F(a_w, b_w)\, p_w^s, \qquad \Lambda_k[F](s) := \sum_{w \in \Sigma^k} F(a_w, b_w)\, p_w^s. \tag{10}$$

For $F \equiv 1$, we omit the reference to F, and we let $\Lambda := \Lambda[1]$. These series satisfy, for $\Re s > 1$, the relation[2] $|\Lambda(F, s)| \leq \|F\| \Lambda(\sigma)$. Since the equality $\Lambda_k(1) = 1$ holds for all k, the series $\Lambda(s)$ is divergent at $s = 1$, and many probabilistic properties of the source can be expressed in terms of the behavior of $\Lambda(s)$, when $\Re s$ is close to 1. For instance, the entropy $h(\mathcal{S})$ of the source $\mathcal{S}$ is defined as the limit (if it exists),

$$h(\mathcal{S}) := \lim_{k\to\infty} \frac{-1}{k} \sum_{w \in \Sigma^k} p_w \log p_w = \lim_{k\to\infty} \frac{-1}{k} \frac{d}{ds} \Lambda_k(s)_{|s=1}. \tag{11}$$

Two types of properties of the source may entail tameness for the mixed series $\varpi(s)$.

▶ **Definition 4** (Tameness of Sources)**.** (a) A source is *weakly tame* if the function $s \mapsto \Lambda(s)$ is analytic on $\Re s > 1$, and of polynomial growth when $\Im s \to \infty$ on any $\Re s \geq \sigma_1 > 1$

(b) Denote by $\mathcal{F}$ the set of functions $F : [0,1]^2 \to \mathbb{R}^+$ of class $\mathcal{C}^1$. A source is Λ–*tame* if $\Lambda(s)$ admits at $s = 1$ a simple pole, with a residue equal to $1/h(\mathcal{S})$, (where $h(\mathcal{S})$ is the entropy of the source)[3] and if one of the following conditions is fulfilled:

1. [S–shape] for any $F \in \mathcal{F}$, the series $\Lambda[F](s)$ is tame at $s = 1$ with a S–shape;

[1] More precisely, this means that $\varpi(s)$ is of polynomial growth on a family of horizontal lines $t = t_k$ with $t_k \to \infty$, and on vertical lines $\Re(s) = \sigma_0 - \delta'$ with some $\delta' < \delta$.

[2] The norm $\|\cdot\|$ is the sup-norm on $[0,1] \times [0,1]$.

[3] Then (proof omitted here) any series $\Lambda[F](s)$ for any $F \in \mathcal{F}, F > 0$, admits at $s = 1$ a simple pole, with a residue equal to

$$\frac{1}{h(\mathcal{S})} \int_0^1 F(x,x)dx.$$

2. [H–shape] for any $F \in \mathcal{F}$, the series $\Lambda[F](s)$ is tame at $s = 1$ with a H–shape;
3. [P–shape] for any $F \in \mathcal{F}$, the series $\Lambda[F](s)$ is tame at $s = 1$, with a P–shape for $F \equiv 1$. For $F \not\equiv 1$, $\Lambda[F](s)$ has either a S–shape, or a P–shape.

This definition is in fact very natural, since it describes various possible behaviors of classical sources. "Most of the time", the simple sources (memoryless sources or aperiodic Markov chains) are Λ–tame. They never have a S–shape, but they may have a H–shape or a P–shape, according to arithmetic properties of their probabilities [8]. Dynamical sources, introduced by Vallée and defined in [17], may have a P–shape only if they are "similar" to simple sources. Adapting deep results of Dolgopyat [2, 3], it is possible to prove that dynamical sources are "most of the time" Λ–tame with a S–shape [1], but they may also have a H–shape [14]. See the cited papers for more details, where all these facts, here described in a informal way, are stated in a formal way and proven.

This definition is also well-adapted to our framework since it describes situations where the mixed series $\varpi(s)$ may be proven tame. Then, the contour of the Rice integral may be shifted to the left, providing an asymptotic expansion for the mean number $S(n)$.

The weak tameness of the source is sufficient to entail the tameness at $s = 1$ (with a S–shape, and an exponent $k_0 = 0$) of series $\varpi(s)$ related to selection algorithms (namely `QuickMin` and `SelMin`). The Λ–tameness of the source is central in the analysis of sorting algorithms, as it ensures the tameness of $\varpi(s)$ related to algorithms `QuickSort`, `InsSort` and `BubSort`); moreover, the tameness shape $\varpi(s)$ is inherited from the one of the source.

2 Summary of our results.

We recall the main steps of the method.
Step 1. Computation of expected values $\pi(i, j)$.
Step 2. Automatic derivation of $\varpi(s, u, t)$; determination of the abscissa σ_0.
Step 3. Expression for the mixed Dirichlet series $\varpi(s)$, and description of the main term of the singular expression of $\varpi(s)/(s - \sigma_0)$. Interpretation of the "dominant" constants.
Step 4. Relation between tameness of the source and tameness of the mixed series $\varpi(s)$. Application of the Rice Formula. Statement of the final results.

This Section presents the results with three tables (found at the end), five propositions and a theorem. Section 2.1 summarizes Steps 1 and 2 with Propositions 2.1 and 2.2, and Table 1. Section 2.2 summarizes Step 3 with Propositions 2.3, 2.4, 2.5, and Table 2. Finally, Section 2.3 states the final result (Theorem 2.6) with Table 3. The proofs are omitted in this short version and defered to a full version of this paper.

2.1 Summary of the results for Steps 1 and 2

We present in the leftmost part of Table 1 the expressions for the mean number $\pi(i, j)$ of key comparisons between U_i and U_j, for each algorithm of interest. With these expressions, it is easy to recover the estimates for the mean number $K(n)$ of key comparisons (recalled in the third column).

▶ Proposition 5. Consider the permutation model described in Section 1.1, and denote by $\pi(i, j)$ the mean number of comparisons between the keys of rank i and j, with $i \leq j$. Then, for any of the five algorithms, the mean numbers $\pi(i, j)$ admit the expressions described in the second column of Table 1 p. 608.

We then obtain the expressions for the analytic lifting $\varpi(s, u, t)$, via an "automatic" derivation taking into account the similar expressions for quantities $\pi(i, j)$.

▶ Proposition 6. Denote by $\varpi(s,u,t)$ the function which provides an analytical lifting of the sequence $\varphi(n,u,t)$ defined in Eq. (6), and by σ_0 the integer which defines the domain $\Re s > \sigma_0$ of validity of this lifting. Then, for any of the five algorithms, the functions $\varpi(s,u,t)$ admit the expressions described in the fifth column of Table 1 p. 608.

2.2 Summary of the results for Step 3 – the mixed Dirichlet series

▶ Proposition 7. Consider any general source, assumed to be weakly tame, together with the fundamental intervals $[a_w, b_w]$ defined in (1) and its Dirichlet series defined in Eq. (10). Then, for any of the five algorithms, the mixed Dirichlet series $\varpi(s)$ (defined in Section 1.6) admit in the domain $\Re s > \sigma_0$, the expressions displayed in the second column of Table 2, together with the values of σ_0 in the third column. Depending on the value of σ_0 the mean number $S(n)$ of symbol comparisons is

$$S(n) = \sum_{k=2}^{n} (-1)^k \binom{n}{k} \varpi(k) \text{ (if } \sigma_0 = 1), \; S(n) = \binom{n}{2}\frac{\Lambda(2)}{2} + \sum_{k=3}^{n} (-1)^k \binom{n}{k} \varpi(k) \text{ (if } \sigma_0 = 2).$$

We now study the relation between tameness of the source and tameness of the mixed Dirichlet series.

▶ Proposition 8. Assume the source $\mathcal{S}$ to be weakly tame. Then, the mixed Dirichlet series $\varpi(s)$ relative to selection algorithms are both tame at $\sigma_0 = 1$ with order $k_0 = 0$ and a S–shape. Moreover, their abscissae δ satisfy

(a) `[QuickMin]` $\delta \geq 1/3$.
(b) `[SelMin]` $\delta > 0$ depends on an exponent a (attached to the source).

Assume the source $\mathcal{S}$ to be Λ–tame. Then, the mixed Dirichlet series $\varpi(s)$ relative to sorting algorithms satisfy the following:

(a) `[QuickSort]` $\varpi(s)$ is tame at $\sigma_0 = 1$ with order $k_0 = 2$.
(b) `[InsSort]` $\varpi(s)$ is tame at $\sigma_0 = 1$ with order $k_0 = 1$.
(c) `[BubSort]` $\varpi(s)$ is tame at $\sigma_0 = 2$ with order $k_0 = 1$.

Moreover, the source $\mathcal{S}$ gives its shape of tameness to the series $\varpi(s)$.

We finally describe the main term of the singular expression of $\varpi(s)/(s-\sigma_0)$ at $s = \sigma_0$.

▶ Proposition 9. The constants of interest which intervene in the main terms displayed in the last column of Table 2 p. 608 are:

(i) The entropy $h(\mathcal{S})$ of the source.
(ii) The coincidence $c(\mathcal{S})$, namely the mean number of symbols needed to compare two random words produced by the source.
(iii) The min–coincidence $a(\mathcal{S})$: this is the mean number of symbols needed to compare a uniform random word and the smallest word of the source.
(iv) The logarithmic coincidence $b(\mathcal{S})$: this is the mean number of symbols needed to compare two words X and Y randomly chosen as follows: the word X is uniformly drawn from the source, and Y is drawn with $Y \geq X$, according to density $1/t$.

The entropy is defined in (11). The constants $a(\mathcal{S}), b(\mathcal{S}), c(\mathcal{S})$ satisfy the inequalities $a(\mathcal{S}) < b(\mathcal{S}), c(\mathcal{S}) < 2b(\mathcal{S})$ and are defined as follows

$$a(\mathcal{S}) = \sum_{\ell \geq 0} q_\ell, \quad b(\mathcal{S}) = \sum_{w \in \Sigma^\star} \int_{\mathcal{T}_w} \frac{1}{t}\, du\, dt \quad c(\mathcal{S}) = 2 \sum_{w \in \Sigma^\star} \int_{\mathcal{T}_w} du\, dt = \sum_{w \in \Sigma^\star} p_w^2 = \Lambda(2).$$

Here q_ℓ is the probability of the prefix of length ℓ of the smallest word of the source, $\mathcal{T}_w$ is the fundamental triangle defined in (2) and $\Lambda(s)$ is defined in (10).

The constants $a(\mathcal{S}), c(\mathcal{S})$ and $h(\mathcal{S})$ are easy to compute for any memoryless source. For the unbiased source $\mathcal{M}_r$, or for the source $\mathcal{B}_p$ on the alphabet $\{0,1\}$, with $p := p_0$, one has: $a(\mathcal{M}_r) = c(\mathcal{M}_r) = \frac{r}{r-1}$, $h(\mathcal{M}_r) = \log r$, $a(\mathcal{B}_p) = \frac{1}{1-p}$, $c(\mathcal{B}_p) = \frac{1}{2p(1-p)}$ and $h(\mathcal{B}_p) = -p\log p - (1-p)\log(1-p)$. The constant $b(\mathcal{S})$ is more difficult to compute even in the memoryless case. But, for the source $\mathcal{M}_r$, one has (see [11] for details)

$$b(\mathcal{M}_r) = \sum_{\ell \geq 0} \Big(1 + \frac{1}{r^\ell} \sum_{k=1}^{r^\ell - 1} \log \frac{k}{r^\ell}\Big), \qquad b(\mathcal{M}_2) \doteq 2.639689120.$$

2.3 Final step

▶ **Theorem 10.** *Consider a general source $\mathcal{S}$. For selection algorithms* `QuickMin`, `SelMin`, *we assume the source to be weakly–tame, and, for sorting algorithms* `QuickSort`, `InsSort`, `BubSort`, *we assume the source to be Λ–tame. Then, the mean number $S(n)$ of symbol comparisons performed by each algorithm on a sequence of n words independently drawn from the same source $\mathcal{S}$ admits the asymptotic behaviour described in Table 3. Here, the constants κ_i in the subdominant terms*[4] *involve the Euler constant γ together with the subdominant constant of the source*[5] *$d(\mathcal{S})$:*

$$\kappa_0 = \frac{2}{h(\mathcal{S})}(\gamma - 2) + 2d(\mathcal{S}), \qquad \kappa_1 = \frac{1}{8h(\mathcal{S})}(2\gamma - 3) + \frac{d(\mathcal{S})}{4}.$$

The errors terms $E(n), F(n)$ are not of the same type for sorting algorithms and selection algorithms.

For selection algorithms, still assuming the source is weakly tame. The error term $F(n)$ is of order $O(n^{1-\delta})$, with $\delta = 1/3$ for `QuickMin`. *For* `SelMin`, *the constant δ depends on the exponent a (if it exists) attached to the source.*

For sorting algorithms, assuming a Λ–tame source with a given shape, we have
- *if the source has a S–shape with abscissa δ, then $E(n) = O(n^{1-\delta})$;*
- *if the source has a H–shape with exponent ρ, then $E(n) = n \cdot\ O\left(\exp[-(\log n)^\rho]\right)$;*
- *if the source has a P–shape with abscissa δ, then $E(n) = n\cdot\Phi(n) + O(n^{1-\delta})$ where $n\cdot\Phi(n)$ is the expansion given by the family of imaginary poles (s_k).*

Discussion. We now compare the asymptotic estimates for the two mean numbers, the mean number $K(n)$ of key–comparisons (column 2 of Table 3) and the mean number $S(n)$ (column 3 of Table 3). There are two types of algorithms

(a) The "robust" algorithms for which $K(n)$ and $S(n)$ are of the same order. This is the case for three algorithms: `InsSort`, `QuickMin` and `SelMin`. Of course, the constants are different for $K(n)$ and $S(n)$, and the ratios $S(n)/K(n)$ involve coincidences of various types always between *two* words, respectively uniform coincidence $c(\mathcal{S})$, logarithmic-coincidence $b(\mathcal{S})$, or min-coincidence $a(\mathcal{S})$.

(b) The algorithms for which $S(n)$ and $K(n)$ are not of the same order, here `QuickSort` and `BubSort`. In both cases, the ratio $S(n)/K(n)$ is asymptotic to $[1/(2h(\mathcal{S}))] \log n$.

(c) This ratio also appears in lower bounds: Combining results due to Seidel [16], with our methods, we obtain a lower bound for the number of symbol comparisons of any sorting

[4] The constant κ_2 is not computed here. Note that the computation of the subdominant term for `InsSort` needs the singular expansion of $\varpi(s)/(s-1)$ at $s=1$.

[5] This constant, defined as the constant term in the singular expansion of $\Lambda(s)$ at $s=1$, is easy to compute for any source $\mathcal{B}_p$: $d(\mathcal{B}_p) = (1/h(\mathcal{B}_p))^2(p\log^2 p + (1-p)\log^2(1-p))$.

algorithm on a source $\mathcal{S}$, asymptotic to $S_0(n) = [1/(2h(\mathcal{S}))]\, n \log^2 n$. Comparing with the well-known lower bound for the number of key comparisons, asymptotic to $K_0(n) = n \log n$, we observe that the ratio $S_0(n)/K_0(n)$ is also asymptotic to $[1/(2h(\mathcal{S}))]\, \log n$.

Algorithms	$\pi(i,j)$	$K(n)$	σ_0	$\varpi(s,u,t)$, $\Re s > \sigma_0$
`QuickSort`	$\dfrac{2}{j-i+1}$	$2n\log n$	1	$2(t-u)^{s-2}$
`InsSort`	$\dfrac{1}{2} + \dfrac{1}{(j-i+1)(j-i)}$	$\dfrac{n^2}{4}$	2	$(s-1)(t-u)^{s-2}$
`BubSort`	$\dfrac{1}{2} + \dfrac{1}{(j-i+1)(j-i)} + {} + \dfrac{2(i-1)}{(j-i+2)(j-i+1)(j-i)}$	$\dfrac{n^2}{2}$	2	$(s-1)(t-u)^{s-3}[t-(s-1)u]$
`QuickMin`	$\dfrac{2}{j}$	$2n$	1	$2t^{s-2}$
`SelMin`	$\dfrac{1}{i(i+1)} + \dfrac{1}{j(j-1)}$	n	1	$(s-1)[u^{s-2}+t^{s-2}]$

(a) Table 1: results for Steps 1 and 2 (Section 2.1)

Algorithms	$\varpi(s)$	σ_0	Main term of $\varpi(s)/(s-\sigma_0)$
`QuickSort`	$\dfrac{2\Lambda(s)}{s(s-1)}$	1	$\dfrac{2}{h(\mathcal{S})}\dfrac{1}{(s-1)^3}$
`InsSort`	$\dfrac{\Lambda(s)}{s}$	2	$\dfrac{c(\mathcal{S})}{2}\dfrac{1}{(s-2)}$
`BubSort`	$-\Lambda[F_0](s-1) = -\sum_{w\in\Sigma^\star} a_w p_w^{s-1}$	2	$-\dfrac{1}{2h(\mathcal{S})}\dfrac{1}{(s-2)^2}$
`QuickMin`	$2\sum_{w\in\Sigma^\star}\int_{a_w}^{b_w}(t-a_w)t^{s-2}dt$	1	$2b(\mathcal{S})\dfrac{1}{s-1}$
`SelMin`	$(s-1)\sum_{w\in\Sigma^\star}(b_w-a_w)\int_{a_w}^{b_w}u^{s-2}du$	1	$a(\mathcal{S})\dfrac{1}{s-1}$

(b) Table 2: results for Step 3 (Section 2.2)

Algorithms	$K(n)$	Dom. term of $S(n)$	Subdominant terms	Rem. term
`QuickSort`	$2n\log n$	$\dfrac{1}{h(S)}n\log^2 n$	$\kappa_0 n\log n + \kappa_2 n$	$E(n)$
`InsSort`	$\dfrac{n^2}{4}$	$\dfrac{c(\mathcal{S})}{4}n^2$	$\dfrac{1}{h(S)}n\log n + \left(\kappa_0 - \dfrac{c(\mathcal{S})}{4}\right)n$	$E(n)$
`BubSort`	$\dfrac{n^2}{2}$	$\dfrac{1}{4h(S)}n^2\log n$	$\left(\kappa_1 + \dfrac{c(\mathcal{S})}{4}\right)n^2$	$nE(n)$
`QuickMin`	$2n$	$2b(\mathcal{S})\,n$		$F(n)$
`SelMin`	n	$a(\mathcal{S})\,n$		$F(n)$

(c) Table 3: results for Theorem 1 (Section 2.3). *Nota.* Rem.: Remainder, Dom.: Dominant.

Figure 1 Tables summarizing results.

Conclusion. We show here the applicability of the method which has been described in the paper [18]. We describe a new point of view on the basic algorithms, and their analysis, which can be (partially) automatized. Our dream is to revisit all standard algorithms from a student book, with this point of view, and perform their realistic analysis.

Acknowledgements. This paper greatly benefited from many discussions we had with Philippe Flajolet, on the topics of the Rice formula and the tameness of sources. For these, we are truly grateful.

References

1 E. Cesaratto and B. Vallée. Gaussian distribution of trie depth for dynamical sources. submitted, 2012.

2 D. Dolgopyat. On decay of correlations in Anosov flows. *Ann. of Math.*, 147(2):357–390, 1998.

3 D. Dolgopyat. Prevalence of rapid mixing in hyperbolic flows. *Ergod. Th. & Dynam. Sys.*, 18:1097–1114, 1998.

4 J. A. Fill. Distributional convergence for the number of symbol comparisons used by Quicksort. Ann. Appl. Probab. (2012), to appear.

5 J. A. Fill and S. Janson. The number of bit comparisons used by quicksort: an average-case analysis. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 300–307, 2004. Long version *Electron. J. Probab.* 17, Article 43, 1-22 (2012).

6 J. A. Fill and T. Nakama. Analysis of the expected number of bit comparisons required by quickselect. *Algorithmica*, 58(3):730–769, 2010.

7 J. A. Fill and T. Nakama. Distributional convergence for the number of symbol comparisons used by Quickselect. *CoRR*, abs/1202.2599, 2012. submitted.

8 P. Flajolet, M. Roux, and B. Vallée. Digital trees and memoryless sources: from arithmetics to analysis. *Proceedings of AofA'10, DMTCS, proc AM*, pages 231–258, 2010.

9 P. Flajolet and R. Sedgewick. Mellin transforms and asymptotics: Finite differences and Rice's integrals. *Theor. Comput. Sci.*, 144(1&2):101–124, 1995.

10 P. Flajolet and R. Sedgewick. *Analytic Combinatorics.* Cambridge University Press, 2009.

11 P. J. Grabner and H. Prodinger. On a constant arising in the analysis of bit comparisons in Quickselect. *Quaestiones Mathematicae*, 31(4):303–306, 2008.

12 N. E. Nörlund. Leçons sur les équations linéaires aux différences finies. In *Collection de monographies sur la théorie des fonctions.* Gauthier-Villars, Paris, 1929.

13 N. E. Nörlund. *Vorlesungen über Differenzenrechnung.* Chelsea Publishing Company, New York, 1954.

14 M. Roux and B. Vallée. Information theory: Sources, dirichlet series, and realistic analyses of data structures. In *Proceedings 8th International Conference Words 2011*, volume 63 of *EPTCS*, pages 199–214, 2011.

15 R. Sedgewick. *Algorithms in C, Parts 1–4.* Addison–Wesley, Reading, Mass., 1998. 3rd ed.

16 R. Seidel. Data-specific analysis of string sorting. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1278–1286, 2010.

17 B. Vallée. Dynamical sources in information theory: Fundamental intervals and word prefixes. *Algorithmica*, 29(1/2):262–306, 2001.

18 B. Vallée, J. Clément, J. A. Fill, and P. Flajolet. The number of symbol comparisons in QuickSort and QuickSelect. In S. A. *et al.*, editor, *Proceedings of ICALP 2009, Part I*, volume 5555 of *Lecture Notes in Computer Science*, pages 750–763. Springer-Verlag, 2009.

Search using queries on indistinguishable items

Mark Braverman[1] and Gal Oshri[2]

1 Princeton University, research partially supported by an Alfred P. Sloan Fellowship, an NSF CAREER award, and a Turing Centenary Fellowship.
2 Princeton University

Abstract

We investigate the problem of determining a set S of k indistinguishable integers in the range $[1,n]$. The algorithm is allowed to query an integer $q \in [1,n]$, and receive a response comparing this integer to an integer randomly chosen from S. The algorithm has no control over which element of S the query q is compared to. We show tight bounds for this problem. In particular, we show that in the natural regime where $k \le n$, the optimal number of queries to attain $n^{-\Omega(1)}$ error probability is $\Theta(k^3 \log n)$. In the regime where $k > n$, the optimal number of queries is $\Theta(n^2 k \log n)$.

Our main technical tools include the use of information theory to derive the lower bounds, and the application of noisy binary search in the spirit of Feige, Raghavan, Peleg, and Upfal (1994). In particular, our lower bound technique is likely to be applicable in other situations that involve search under uncertainty.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Search, Noisy Search, Information Theory, Query Complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.610

1 Introduction

This paper investigates the problem of identifying a set S of indistinguishable items by repeated queries where we know the range of values the items can take. At every query, we gain information based on our query and some random item from the set S we are trying to find (we do not know which item was chosen). The overall simple statement of the problem makes it widely generalizable. The query can be thought of as an experiment in which we apply a measurement on an element of S without knowing which element has been measured. The set of items can refer to a set of DNA strands in a "soup" of DNAs, passwords or any item that we might be interested in finding when we know what possible values the item may take. The queries can be viewed as tests on DNA strands, attempts at guessing a password or any trial we may run that will provide some information about one of the items in question. The specific problem we investigate is where the items are integers. Our queries are guesses of integers which return the result of a comparison with a chosen integer from the set we are trying to find.

As far as we know, this problem has not been investigated in the literature. However, it falls into the rich class of noisy search problems. Since we do not know which number was chosen when we query a number, we have to deal with a lack of information in trying to determine the set of numbers. Due to this missing information, it is not immediately obvious that there exists a solution to the problem.

In this paper we give asymptotically tight upper and lower bounds for the number of queries needed to find a set S of size k of numbers from $\{1,\dots,n\}$, where the queries are comparison queries.

We briefly discuss similar problems that have been previously studied. Feige et al. explored the depth of noisy decision trees, where each node can be wrong with some constant probability, in [2]. One of the problems they investigated is binary search where the result of each query is wrong with a constant probability. They presented an algorithm to solve this with running time $\Theta(\log \frac{n}{Q})$ where n is the input set size and Q is the probability of error of the algorithm. The algorithm we present uses a similar technique to the one used for noisy binary search in [2].

The Renyi-Ulam game is also a related problem. In one variation of this game, we need to discover a chosen integer. To do this, we query a number and are told whether the number we are trying to find is greater than the number we guessed or not. However, some constant number of lies are allowed. In [8], one lie is allowed, which means that one of the responses to our queries can be false. Similarly, Pelc discussed in [5] an algorithm for performing the search when one lie is allowed and concluded that the original question posed by Ulam (finding an integer between one and a million with one lie allowed) requires 25 queries. In [8], [5] and other papers that explore the Renyi-Ulam game, some restriction is placed on the pattern of queries with false results. Ravikumar and Lakshmanan discussed such patterns (and why they are necessary to make the problem solvable) in [7].

The problem we are investigating is motivated by applications that involve a search for several items by repeated queries where we do not know which item was chosen to be compared with our query (i.e. the items are indistinguishable). One interpretation is where the items represent DNA strands in a mixture that we are trying to identify. We can perform tests that give us some information about one of the DNA strands in the mixture, but we do not know which one. Similarly, instead of trying to identify DNA strands, we might be trying to identify passwords where our queries give us some partial information about one password out of several that a particular user often uses (and switches between).

We note that the applications mentioned do not take the exact form as the problem we explore. The items in our problem are integers and the queries are guesses of an integer that result in the response 'less than or equal to' or 'greater than'. In generalizing the problem to other applications, the form of items or queries may change. For example, the queries in the DNA mixture example may describe a property of a particular nucleotide instead of returning one of two possible answers. Therefore, the algorithm will have to be changed. However, a similar framework can be used which allows information to be gained despite the uncertainty regarding query responses due to the indistinguishability of the items. A solution to the problem we have posed can lead to the development of new methods for identifying a set of items where we know these items can only take on a certain range of values. On the lower-bound side, our results show that information-theoretic quantities are very effective at measuring and upper-bounding information learned from queries, even when such information is only a fraction of one bit. We believe that the information-theoretic lower bound technique will generalize to tight lower bounds in other settings.

We now discuss the results and structure of the paper. In Section 2, we formally introduce the problem we are solving with the restriction that the number of chosen integers is significantly smaller than the range of integers available. We prove a lower bound for the problem in Section 3.1 using information theoretic techniques. This involves constructing the hard instances where we split the possible values the chosen integers can take into consecutive clusters of equal size and place one chosen integer in each such cluster. Intuitively, this forces the search algorithm to find the elements one at a time, which turns out to be costly due to the fact that we don't control the sample. To formalize this intuition, we calculate the entropy of the random variable representing a particular chosen integer (it may take values of the integers in one of the clusters described above). We then use the mutual information of this random variable and the random variable representing the responses to the queries we make to find the minimum number of queries required to find that chosen integer. After showing that the same minimum number of queries applies to at least half of the chosen integers, we reach a lower bound of $\Omega(k^3 \log n)$, where k is the size of the set S and the elements of S take integer values between 1 and n (inclusive). Further, this bound extends to all $k < n$, using a slightly different set of hard instances. When $k > n$ we obtain a lower bound of $\Omega(k^2 n \log n)$. In Section 4, we present an optimal algorithm for solving the problem, proving both its correctness and worst case running time of $O(k^3 \log \frac{n}{\delta})$ where δ is the probability of error. This shows that the lower bound is tight. Moreover, while the lower bound applies to finding S even with a constant error probability, we see that the upper bound remains asymptotically the same even if we set the error $\delta = n^{-O(1)}$ to be polynomially small.

Our results show that the problem we describe can be solved in practice when the items we are searching for can take a large number of values. This is because the dependence of the running time on n grows as $\log n$. However, the number of items in S needs to remain small because the dependence of the running time on k grows as k^3.

2 Problem definition

We consider a (multi-)set S of k distinct integers where each is $X_i \in \{1, 2, \ldots, n\}$ for $1 \le i \le k$. Our goal is to discover the set S. The process is to repeat the following three steps:

1. Query an integer $Y \in \{1, 2, \ldots, n\}$.
2. An integer X_i is selected from S uniformly at random.
3. We are told whether $X_i \le Y$ or $X_i > Y$.

These three steps are repeated until we know what the k integers in S are. Our goal is to find the most efficient algorithm for determining S. Our model of computation is that queries are the costly operations. Therefore, by finding the most efficient algorithm we mean finding the algorithm that minimizes the number of queries made. We refer to this as 'the problem' we are solving. Furthermore, for brevity, we refer to the two possible responses to queries as '$\le$' ($X_i \le Y$) and '$>$' ($X_i > Y$) and the k integers in S as 'the chosen integers'.

In this paper we give a complete characterization of the query complexity of this problem. Note that since the X_i is selected at random from S, we cannot hope for a deterministic algorithm, and have to settle for a probabilistic performance guarantee. We focus on the regime where we are required to output the correct set S except with some (possibly constant) probability δ. The answer can be broken down into three main regimes, which will be discussed in the analysis: (1) $k \ll n$, e.g. $k < \sqrt{n}$; (2) $\sqrt{n} < k < n$; and (3) $k \ge n$. The answer is given by the following main theorem:

▶ **Theorem 1.** *The number of queries needed to determine a multi-set $S \subset [n]$ of size k with a given error $n^{-O(1)} < \delta < 1/4$ is $\Theta(k^3 \log n)$ when $k \le n$, and $\Theta(k^2 n \log n)$ when $k \ge n$.*

Note that the distinction between $k < \sqrt{n}$ and $\sqrt{n} < k < n$ only comes up in the analysis, but (asymptotically) makes no difference in the result.

▶ Remark. Because of the way the algorithms work, Theorem 1 remains true even if the comparisons in the query answers are themselves noisy, and output the correct value of $X_i \overset{?}{>} Y$ correctly only with probability $1/2 + \gamma$ for some constant $\gamma > 0$.

▶ Remark. Somewhat surprisingly, same bounds hold for a fairly broad range of error parameters. In particular, the lower bound holds even when the error is constant, while the upper bound holds even for polynomially small errors (the constant in the $\Theta(\cdot)$ may depend on the constant β in $\delta = n^{-\beta}$).

3 The lower bounds

We begin with showing the lower bound. In fact, we break the lower bound into two regimes: $k \le \sqrt{n}$ and $k > \sqrt{n}$. In the former regime, we use information-theoretic techniques to show the lower bound. In the latter, we give a more straightforward proof of the $\Omega(k^3 \log k)$ lower bound when $k < n$, and $\Omega(k^2 n \log n)$ when $k > n$. The $\Omega(k^3 \log k)$ lower bound is weaker in general than $\Omega(k^3 \log n)$ when $k < n$, but is equivalent in the regime where $k > \sqrt{n}$.

3.1 The case $k \le \sqrt{n}$: an information-theoretic lower bound

The main technical ingredient in the lower bound proof is the Kullback-Leibler divergence and mutual information. We first introduce these terms and the lemmas we will use. For a more thorough introduction to these, see [1].

The Kullback-Leibler divergence (KL-divergence) measures the difference between two probability distributions:

▶ Definition 3.1. For discrete random variables P and Q over sample space Ω, the KL-Divergence is defined as:

$$D_{KL}(P||Q) = \sum_{i \in \Omega} P(i) \log \frac{P(i)}{Q(i)}$$

with the convention that the term in the sum is interpreted as 0 when $P(i) = 0$ and $+\infty$ when $P(i) > 0$ and $Q(i) = 0$

We also use mutual information, which we define and arrange into a form we will use:

▶ Definition 3.2. Mutual information is a measure of the correlation between two random variables. The more independent the variables are, the lower the mutual information is.

$$I(X;Y) = D_{KL}(p(x,y)||p(x)p(y))$$

Before we rearrange this definition into a form we will use, we first note (from [1]) that it can also be written in terms of the more familiar Shannon entropy as:

$$I(X;Y) = H(X) - H(X|Y).$$

Since $H(X) \ge H(X|Y)$, $I(X;Y) \ge 0$. If entropy is interpreted as the uncertainty regarding a probability distribution, we see that the mutual information between X and Y represents the reduction in uncertainty of X by knowing Y.

We now return to the original definition given for mutual information. Using the definition of the KL-divergence and conditional probability ($p(x|y) = \frac{p(x,y)}{p(y)}$), we have:

$$\begin{aligned} I(X;Y) = \sum_y p(y) \sum_x p(x|y) \log \frac{p(x|y)}{p(x)} &= \sum_y p(y) D_{KL}(p(x|y)||p(x)) \\ &= E_Y[D_{KL}(p(x|y)||p(x))] \end{aligned}$$

Thus we see that the mutual information is the expectation of the KL-divergence between the probability distribution of X and the probability distribution of X conditioned on Y. If these two distributions have a high KL-divergence, then knowing Y provides us a high amount of information regarding the probability distribution of X. This is equivalent to saying that the mutual information of X and Y is high.

We will use the chain rule for mutual information:

▶ **Lemma 2.** $I(X; Y_1, Y_2, \ldots, Y_k) = I(X; Y_1) + I(X; Y_2|Y_1) + \ldots + I(X; Y_k|Y_{k-1}, \ldots, Y_2, Y_1)$

For a proof of the above lemma, see [1]. We are now done defining the information theory terms we will need. Lastly, we will need the following lemma which describes the KL-divergence between two Bernoulli random variables with a similar probability of success:

▶ **Lemma 3.** *$D_{KL}(B_{p\pm\varepsilon}||B_p) = O(\varepsilon^2)$ where B_p is a Bernoulli random variable with probability of success p, $\frac{1}{4} \leq p \leq \frac{3}{4}$ and $\varepsilon \leq \frac{1}{8}$.*

The proof for this lemma is straightforward and is thus not included here. We are now ready to begin our proof of the lower bound. The approach taken is to show that the information gain from each query is small compared with the total information required to find a certain chosen integer. This will allow us to show that a certain minimum number of queries is required to find each of the k integers.

▶ **Lemma 4.** *The lower bound for the number of queries required to find the k integers between 1 and n in the set S with probability > 0.99, when $8 \leq k \leq \sqrt{n}$, is $\Omega(k^3 \log n)$*

Proof. We choose our input as follows. Split the integers in the range $[1, n]$ into k equally sized clusters. Call these clusters $G_1, G_2, \ldots, G_k$. Let there be one of the k chosen integers in each such cluster. This integer is chosen uniformly at random from the integers in the cluster. Note that the number of integers in each cluster is $\frac{n}{k}$, which, without loss of generality, we will assume is an integer.

We consider individually a cluster G_i where $\frac{k+4}{4} \leq i \leq \frac{3k}{4}$. Let L be the random variable that represents the chosen integer in G_i. Since this number is chosen uniformly at random from $\frac{n}{k}$ elements, the probability of each integer being the chosen integer is $P(x) = \frac{1}{\frac{n}{k}} = \frac{k}{n}$. Therefore, the entropy of L is $H(L) = \sum_x P(x) \log \frac{1}{P(x)} = \sum_{i=1}^{\frac{n}{k}} \frac{k}{n} \log \frac{n}{k} = \log \frac{n}{k}$. We now define Q_j to be a Bernoulli random variable representing the response to the j^{th} query (i.e. either '$\leq$' or '$>$'). We need to make enough queries so that the information gain relevant to L is close to the entropy of L in order to determine the chosen number in G_i with a high degree of accuracy. This is equivalent to saying that the mutual information between L and the queries made $Q_1, Q_2, \ldots, Q_l$ is at least a constant times the entropy of L. Indeed, in the end, we must have determined the point with probability greater than 0.99. Therefore, conditioned on the queries, most of the mass is concentrated on one point and $H(L|Q_1, \ldots, Q_l) < 0.2 \log \frac{n}{k}$. Therefore, $I(L; Q_1, \ldots, Q_l) = H(L) - H(L|Q_1, \ldots, Q_l) = \Omega(\log \frac{n}{k})$. Thus, we need:

$$I(L; Q_1, Q_2, \ldots, Q_l) \geq \Omega(\log \frac{n}{k}), \tag{1}$$

where l is the number of queries made. We want to find the minimum l for which this is true. First, we use Lemma 2 (chain rule) to write:

$$I(L; Q_1, Q_2, \ldots, Q_l) = I(L; Q_1) + I(L; Q_2|Q_1) + \ldots + I(L; Q_l|Q_{l-1}, \ldots, Q_2, Q_1). \tag{2}$$

Take one of these terms and recall that we can express mutual information in terms of KL-divergence:

$$I(L; Q_j|Q_{j-1}, \ldots, Q_1) = E_Q[D_{KL}(p(Q_j|L, Q_{j-1}, \ldots, Q_1)||p(Q_j|Q_{j-1}, \ldots, Q_1))]$$

where $1 \leq j \leq l$. Thus, we need to find the KL-divergence of $Q_j|L, Q_{j-1}, \ldots, Q_1$ and of $Q_j|Q_{j-1}, \ldots, Q_1$. We note that since we chose cluster G_i, there are $i - 1$ of the k chosen integers that are smaller and $k - i$ of the k numbers that are bigger than any element of

G_i. Therefore, for both probability distributions, the probability that the response is '$\leq$' is at least $\frac{i-1}{k}$ and the probability that the response is '$>$' is at least $\frac{k-i}{k}$. Therefore, both probability distributions are Bernoulli with probability of success (taking success to be the response '$\leq$') between $\frac{i-1}{k}$ and $1 - \frac{k-i}{k} = \frac{i}{k}$. Thus, the difference in probabilities of success of the two distributions is at most $\frac{i}{k} - \frac{i-1}{k} = \frac{1}{k}$. Then if we let $Q_j|L, Q_{j-1}, \ldots, Q_1$ be B_p and let $Q_j|Q_{j-1}, \ldots, Q_1$ be $B_{p\pm\varepsilon}$, we know $\frac{1}{4} \leq p \leq \frac{3}{4}$ (because $\frac{k+4}{4} \leq i \leq \frac{3k}{4}$) and $0 \leq \varepsilon \leq \frac{1}{k}$ (because this is the maximum difference in probability of success between the two distributions). By lemma 3, $D_{KL}(p(Q_j|L, Q_{j-1}, \ldots, Q_1)||p(Q_j|Q_{j-1}, \ldots, Q_1)) = O(\varepsilon^2) = O(\frac{1}{k^2})$. So: $E_Q[D_{KL}(p(Q_j|L, Q_{j-1}, \ldots, Q_1)||p(Q_j|Q_{j-1}, \ldots, Q_1))] = O(\frac{1}{k^2})$ and we have:

$$I(L; Q_j|Q_{j-1}, \ldots, Q_1) = O\left(\frac{1}{k^2}\right).$$

Returning to equation 2:

$$I(L; Q_1, Q_2, \ldots, Q_l) = \sum_{j=1}^{l} I(L; Q_j|Q_{j-1}, \ldots, Q_1) = O\left(l\frac{1}{k^2}\right)$$

From (1), we have $O(l\frac{1}{k^2}) \geq \Omega(\log \frac{n}{k})$ so

$$l = \Omega\left(k^2 \log \frac{n}{k}\right) = \Omega(k^2 \log n)$$

since $k \leq \sqrt{n}$. This is the minimum number of queries to find the chosen integer in G_i. This holds in total for $\frac{3k}{4} - \frac{k+4}{4} + 1 = \frac{k}{2}$ of the k chosen numbers (this is the number of clusters G_i with i in the range we considered). Note that to find the chosen number in G_i, queries made in determining the number within G_j with $j \neq i$ provide no information for determining the number in G_i (as all queries are either bigger or smaller than all the numbers in G_i). Then finding $\frac{k}{2}$ of the k chosen numbers requires at least $\Omega\left(\frac{k}{2}k^2 \log n\right) = \Omega(k^3 \log n)$ time. Therefore, finding all k of the chosen numbers requires at least $\Omega(k^3 \log n)$ queries. ◀

3.2 The lower bound when $k > \sqrt{n}$

Next we turn our attention to the lower bound in the regime where $k > \sqrt{n}$. We start with the case $\sqrt{n} < k \leq n - 2$, as the case $k > n - 2$ is treated very similarly. The multi-set S is constructed as follows: we place $k/4$ 1's and $k/4$ n's in S. Partition the rest of the set $\{1, \ldots, n\}$ into bins $B_1 = \{2, 3\}$, $B_2 = \{4, 5\}$, etc. For each bin B_i for $i = 1, 2, \ldots, k/2$, we place exactly one of the elements of B_i in S independently and uniformly at random. We now look at the process of determining which element of B_i has been selected using the queries. Note that only the query with $Y = 2i$ carries any information on which element of B_i has been selected. Thus a set of observations can be specified by a set of pairs of numbers $\{(l_i, h_i)\}_{i=1}^{k/2}$ where l_i represents the number of times we queried $Y = 2i$ and received the '$\leq$' answer, and h_i represents the number of times we received the '$>$' answer. The probability of each answer is between $1/4$ and $3/4$, and varies by $1/k$ depending on whether we selected $2i$ or $2i + 1$ in B_i.

When we output the set S, we need to make $k/2$ decisions of whether to output $2i$ or $2i + 1$ for each B_i. Each of these decisions should depend only on the values of (l_i, h_i), and should maximize the probability that the output is correct. This can only be done by outputting the maximum likelihood value for each B_i. More precisely, we should output $2i$ if $\frac{l_i}{l_i+h_i} > \frac{k/4+i-1/2}{k}$, and $2i+1$ otherwise. We are not particularly concerned with these details, but only with the probability that our output is wrong. Denote by $\varepsilon_i > 0$ the probability that the maximum-likelihood output given (l_i, h_i) is incorrect. We first claim that to have a probability of > 0.9 to be correct in outputting S, we must have a bound on the sum of the ε_i's.

▶ Claim 3.1. If given the values $\{(l_i, h_i)\}_{i=1}^{k/2}$ the output S is correct with probability > 0.5, then $\sum_{i=1}^{k/2} \varepsilon_i < 1$.

Proof. Since the events of being correct on each B_i are independent, the probability of being correct on all B_i's is given by

$$0.5 < \prod_{i=1}^{k/2}(1-\varepsilon_i) < e^{-\sum_{i=1}^{k/2}\varepsilon_i},$$

which implies the statement of the claim. ◀

Next, let us denote by μ_i the *a-priori* expected number of '$\leq$' responses on $l_i + h_i$ queries, and let $d_i := |l_i - \mu_i|$ be the observed deviation from this expected value. Intuitively, the greater this deviation, the greater is our confidence in the answer. In fact, it is not hard to formalize this intuition:

▶ Claim 3.2. For each i, and $k > 25$, $\varepsilon_i > e^{-10d_i/k}/3$.

Proof. Suppose wlog that $l_i > \mu_i$, and thus we are outputting $2i$. Denote $p = \frac{k/4+i-1}{k}$ and $q = \frac{3k/4-i}{k}$. We have by Bayes' rule

$$\varepsilon_i = Pr[2i+1|(l_i,h_i)] = \frac{Pr[(l_i,h_i)|2i+1]}{2Pr[(l_i,h_i)]} \geq \frac{Pr[(l_i,h_i)|2i+1]}{2Pr[(l_i,h_i)|2i]} =$$
$$\frac{p^{l_i}(q+1/k)^{h_i}}{2(p+1/k)^{l_i}q^{h_i}} = \frac{p^{\mu_i-1}(q+1/k)^{l_i+h_i-\mu_i+1}}{2(p+1/k)^{\mu_i-1}q^{l_i+h_i-\mu_i+1}} \cdot \frac{p^{l_i-\mu_i+1}(q+1/k)^{\mu_i-l_i-1}}{(p+1/k)^{l_i-\mu_i+1}q^{\mu_i-l_i-1}} =$$
$$\frac{Pr[(\mu_i-1, l_i+h_i-\mu_i+1)|2i+1]}{2Pr[(\mu_i-1, l_i+h_i-\mu_i+1)|2i]} \cdot \left(1-\frac{1/k}{p+1/k}\right)^{d_i+1} \cdot \left(1+\frac{1/k}{q}\right)^{-d_i-1} \geq$$
$$(1/2)\cdot(1-5/k)^{2d_i+2} \geq e^{-(5/k)(2d_i+2)}/2 > e^{-10d_i/k}/3.$$

The second-to last inequality follows from the fact that the breakdown $(\mu_i - 1, l_i + h_i - \mu_i + 1)$ is more likely under the selection of $2i+1$ than under the selection of $2i$. ◀

Putting Claims 3.1 and 3.2 together we see that assuming the probability that the output S is correct is > 0.5, we must have

$$\sum_{i=1}^{k/2} e^{-10d_i/k} < 3. \tag{3}$$

▶ Claim 3.3. Equation (3) implies $\sum_{i=1}^{k/2} d_i > \frac{k^2}{40}\ln k$, for $k > 40$.

Proof. Denote $\tau_i := e^{-10d_i/k}$, and let $f(x) := -\ln x$. The function $f(x)$ is convex, and thus we have

$$\sum_{i=1}^{k/2}\frac{10d_i}{k} = \sum_{i=1}^{k/2} f(\tau_i) \geq \frac{k}{2}\cdot f\left(\frac{2}{k}\sum_{i=1}^{k/2}\tau_i\right) > \frac{k}{2}\ln\frac{k}{6} > \frac{k}{4}\ln k,$$

since $k > 40$. This implies the claim. ◀

To finish the proof let D_t denote the random variable representing the value of $\sum_{i=1}^{k/2} d_i$ after t queries. Let $Z_t = D_t - \frac{t}{k}$. At each time step, a query to $Y = 2i$ will on average not change d_i if the element from B_i is not selected for comparison with Y. If it is selected, it will change d_i by at most 1. Thus, on average, D_t only grows by at most $\frac{1}{k}$ after each time step. Thus Z_t is a supermartingale. Let T be the random variable representing the time at which we stop and output S. By the optional stopping time theorem, we have $E[Z_T] \leq 0$, which implies $E[T] \geq k \cdot E[D_T]$.

If our overall success probability is > 0.75, it must be the case that with probability $> 1/2$ the probability of the output S being correct conditioned on the observed $\{(l_i, h_i)\}_{i=1}^{k/2}$ is $> 1/2$. Thus by Claims 3.1, 3.2 and 3.3, we have $D_T > \frac{k^2}{40} \ln k$ with probability $> 1/2$. Thus,

$$E[T] \geq k \cdot E[D_T] > k \cdot \frac{1}{2} \cdot \frac{k^2}{40} \ln k = \Omega(k^3 \log k),$$

completing the proof of the lower bound.

▶ Remark. The proof in the regime $k > n - 2$ is very similar. The only difference is that there are $n/2$ bins now, and we'd get $E[D_T] = \Omega(kn \log n)$ instead of $\Omega(k^2 \log n)$, and thus $E[T] = \Omega(k^2 n \log n)$.

We will now study the case where $k \leq n$.

4 Optimal upper bounds

As discussed in the previous sections, it is not immediately clear how to make use of the information gained from queries because we do not know which of the k integers the information corresponds to. In this section, we present an algorithm for solving this problem. The algorithm is optimal when the probability of error required is constant (which means its worst case running time matches the lower bound). Our algorithm finds each of the k numbers individually, without attempting to use information gained when finding one integer to find another integer. We first introduce a concept we will use in all our algorithms:

▶ Definition 4.1. The k-position of an integer y is the number of integers in S that have a value less than or equal to y

The general technique of the algorithms is to do a binary search for a chosen integer, but repeat each query of the binary search enough times to know the k-position of the queried integer. A straightforward application of binary search with repeated queries would take $\Omega(k^2 \log^2 n)$ queries to find the k-position of a number, even with a constant error probability. We essentially use the noisy binary search technique of Feige et. al. [2] to attain the optimal query complexity. We start with the following simple lemma:

▶ **Lemma 5.** *We can find the k-position of integer y by making $2k^2 \log \frac{2}{\delta}$ queries with the probability of being correct being at least $1 - \delta$.*

Proof. Let K_y be the k-position of y. We do m queries of y to find K_y. For each query Q_i, the probability of a response being '$\leq$' or '$>$' is given simply in terms of K_y:

$$Pr[Q_i =' \leq '] = \frac{K_y}{k}$$

$$Pr[Q_i =' > '] = \frac{1 - K_y}{k}$$

because K_y is the number of integers in S less than or equal to y and each such integer is chosen as the X_i for a query with equal probability. We use the analogy that the random variable Q_i is a coin with probability of heads (which represents '$\leq$') being $p = \frac{K_y}{k}$. Given m tosses of the coin, of which x are heads, we can approximate p as: $\hat{p} = \frac{x}{m}$. We need to find the relation between the number of tosses m and the probability of error in this approximation. Using standard concentration bounds [6], we see that $m \geq \frac{1}{2\varepsilon^2} \log \frac{2}{\delta}$ coin tosses are needed to guarantee that $|\hat{p} - p| \leq \varepsilon$ with error at most δ (where $\varepsilon > 0$).

We need to decide on a value for ε. Note that K_y is an integer in the range $[1, k]$ and therefore, p can only take on the values $0, \frac{1}{k}, \frac{2}{k}, \ldots, \frac{k}{k}$. Thus, we need $\varepsilon \leq \frac{1}{2k}$ so then we can always round $\hat{p}$ to the closest $\frac{i}{k}$, where $i \in \mathbb{Z}$ and $0 \leq i \leq k$. Using this in the results from [6], we see that $m = 2k^2 \log \frac{2}{\delta}$ coin tosses are enough to guarantee that we know the correct value of p with probability of error being at most δ. Given p, we have $K_y = kp$ so we have the k-position of y. ◀

We note that this immediately lets us solve the problem for $k \geq n$:

▶ **Corollary 6.** *When $k \geq n$, there is an $O(k^2 n \log n)$ algorithm to find all k integers in S with probability $1 - n^{-c}$ for all constant $c > 0$.*

Proof. We find the k-position of all n integers in the range $[1, n]$. Given the k-position of all n integers, we know how many of the k numbers have each integer value. If the k-position of $Y - 1$ is i and the k-position of Y is $i + j$, we know there are j of the chosen numbers with the value Y (for $1 < Y \leq n$. For $Y = 1$, we know the number of the chosen integers with this value is equal to the k-position of Y).

To find the k-position of an integer with probability of error at most δ, we need to perform $O(k^2 \log \frac{2}{\delta})$ queries. If we want the probability of error of the algorithm to be a constant, we need the probability of error of finding the k-position of each integer to be at most $\delta = n^{-(c+1)}$ so that applying a union bound gives a total probability of error $< n^{-c}$ (since we find the k-position of n integers). Thus, to find the k-position of each integer we need to perform $O\left(k^2 \log \frac{2}{\frac{1}{n^{c+1}}}\right) = O\left(k^2 \log n\right)$ queries. Since we do this for n integers, the total number of queries we make is: $O(k^2 n \log n)$. ◀

When $k \leq n$, we could have used an approach involving a binary search where our decision at each stage in the search is based on the k-position of the current number in the search. However, this approach is problematic because of the constant error each time we find the k-position of a number. This flaw is mentioned for a similar algorithm in [3]. The number of queries we make is $O(mk \log n) = O\left(k^3 \log n \log \frac{2}{\delta}\right)$. Each group of queries of the same y (m of them) give the wrong result with probability δ. Applying a union bound, our overall probability of error (Δ) is $\Delta = k \log(n)\delta$. If we want Δ to be a constant, we need $\delta = \frac{1}{k \log n}$ and thus, the number of queries we make is actually $O\left(k^3 \log(n) \log(2k \log n)\right)$.

To alleviate this problem, we model our algorithm as a random walk on a tree. In using this technique, we follow [2]. In [2], the random walk approach is taken to do a noisy binary search. We use this technique to find each of the chosen k integers, although each step of the random walk is modified to accommodate our lack of information about which of the k integers was chosen in a particular query. We use a binary tree where the leaves are (in order) the integers $1, 2, \ldots, n$. The internal nodes represent intervals that are the union of the leaves in their subtrees. For example, the root node has the interval $[1, n]$ and the left child of the root has the interval $[1, \lfloor \frac{n}{2} \rfloor]$. The tree height is $\log n$. Finally, we extend this tree by adding chains of length $m' = O(\log n)$ to each of the leaf nodes, where the nodes in these chains have the same value as the leaf they are attached to. An example tree with $n = 4$ is shown in Figure 1 below.

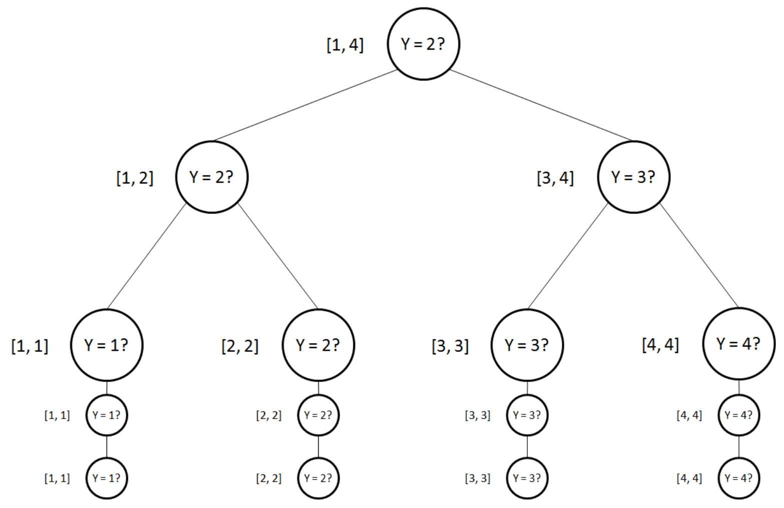


Figure 1 Tree for the random walk with $n = 4$

4.1 Algorithm

We discuss an algorithm for finding the t^{th} of the k chosen integers. This algorithm is repeated k times (once for each of the k numbers). Starting at the root, for each node v we take the following two steps:

1. We first check whether the t^{th} chosen integer is in the range of the node (call it $[a, b]$). To do this, we find the k-position of $a - 1$ and b by doing $8k^2$ queries of each of them. If we find that the k-position of $a - 1$ is at most $t - 1$ and the k-position of b is at least t, then the t^{th} number lies in the range $[a, b]$. Otherwise, we backtrack up the tree to the parent node of v .
2. If, according to the first step, the t^{th} number lies in the range $[a, b]$, we do $10k^2$ queries of the middle value of the range of the node (call this u where $u = \lfloor \frac{a+b}{2} \rfloor$). If v is not a leaf (or on a leaf chain) and the k-position of u is at most $t - 1$, we choose the right child of v. If the k-position of u is at least t, we choose the left child of v. If v is a leaf (or on a leaf chain), we go down the chain further regardless of the result of the queries.

Note that there is a constant probability of error each time we determine the k-position of an integer. This leads to a constant probability of choosing the wrong node to go to next. We will analyze this probability shortly.

The algorithm walks for $m = O(\log n)$ steps and then stops, where $m < m'$. If it stops on an internal node, the algorithm failed. If it stops on one of the leaf chains (or a leaf node), it outputs the value of the leaf (i.e. declares this value to be the value of the t^{th} of the k numbers).

The following theorem summarizes our results:

▶ **Theorem 7.** *Our algorithm finds all k integers in S in $O\left(k^3 \log\left(\frac{n}{\delta}\right)\right)$ time with probability of error at most δ for $k \le n$*

To reach this theorem, we use the following lemma:

▶ **Lemma 8.** *The algorithm finds the correct t^{th} integer in S with the probability of error being at most $e^{-\frac{m}{35}}$, where m is the number of steps in the random walk.*

Proof. We need to prove that the algorithm's position on the walk after m steps is the correct leaf chain with high probability. Orient all edges of the tree so they are directed towards the correct leaf chain (and within this leaf chain they are directed down). We can do this because the graph is a tree (there is only one path between every two vertices) and there is only one correct leaf. We can now consider the algorithm's position in the tree as a one dimensional random walk. We let the starting point of the walk be 0 (the root of the tree), the correct leaf be R steps to the right and any of the wrong leaves be R steps to the left. Note that $R = \log n$ (height of the tree).

We need to find the probabilities of moving left and right in the random walk. We will show that the probability of moving in the correct direction (to the right) is at least 0.7 at every node. Furthermore, note that the decision made at any node is independent of the previous steps in the random walk. Let q be the probability of going left at any move. This is equivalent to the probability of going along the wrong direction of an edge, which is equivalent to making a mistake somewhere in choosing the next vertex. The probability of incorrectly calculating whether the t^{th} number is in the range $[a, b]$ is at most the probability that we incorrectly calculate the k-position of either $a-1$ or b. Since we do $8k^2$ queries of each, by Lemma 5 we know that the probability of error in calculating the k-position of each is δ where $2\log\frac{2}{\delta} = 8 \Rightarrow \delta = \frac{1}{8}$. So the probability of incorrectly calculating the k-position of either $a-1$ or b is at most $1 - \left(\frac{7}{8}\right)^2 = \frac{15}{64}$. Similarly, we do $10k^2$ queries of u, so the probability of error is δ where $2\log\frac{2}{\delta} = 10 \Rightarrow \delta = \frac{1}{16}$. Thus, the total probability of error at each node is $\frac{15}{64} + \frac{1}{16} < 0.3$. Therefore, $q < 0.3$ and $p \geq 0.7$, where p is the probability of going to the right (i.e. the correct direction). Figure 2 illustrates the random walk space.

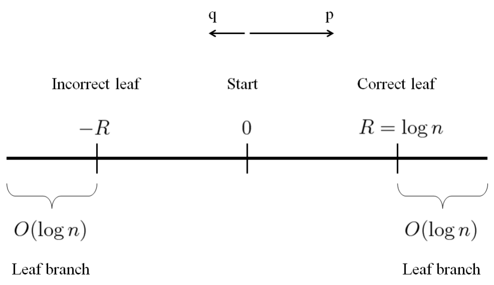


Figure 2 The random walk space

For the algorithm to be correct, it must be on or to the right of R after m steps (so it returns the correct integer), otherwise it is wrong. Let X be the random variable denoting the number of moves to the right made after m moves. Then $m - X$ is the number of moves to the left. Therefore, the algorithm is correct if $X - (m - X) = 2X - m \geq R$. This is equivalent to the condition that $X \geq \frac{R+m}{2}$. Then the probability that the algorithm is correct is $Pr[X \geq \frac{R+m}{2}] = 1 - Pr[X < \frac{R+m}{2}]$ and $Pr[X < \frac{R+m}{2}]$ is the probability of error we want to bound. To find $E[X]$, let X_i be an indicator random variable that is 1 if the algorithm moves to the right on the i^{th} move and 0 otherwise. Note that $Pr[X_i = 1] = p \Rightarrow E[X_i] = p$. Therefore, $E[X] = E[X_1] + E[X_2] + \ldots + E[X_m] = pm$ by linearity of expectation. We want to use a Chernoff bound to bound the probability of error, so we need to find a δ such that:

$$\frac{m+R}{2} = (1-\delta)pm \Rightarrow 1 - \delta = \frac{m+R}{2pm} \Rightarrow \delta = \frac{2pm - m - R}{2pm}$$

Note that $0 \leq \delta \leq 1$ because $0 \leq 2pm - m - R \leq 2pm$. Since each step of the random walk is independent of the other steps (i.e. X_i is independent of X_j for $i \neq j$), we can use the

Chernoff bound ([4]):

$$Pr[X < \frac{m+R}{2}] \leq e^{-\frac{\delta E[X]}{2}} = e^{-\left(\frac{2pm-m-R}{2pm}\right)^2 \frac{pm}{2}} = e^{-\frac{(2pm-m-R)^2}{8pm}}$$

Recall that $p \geq 0.7$ and set $m = x \log n$, where x is a constant. Then $\frac{(2pm-m-R)^2}{8pm} \geq \frac{(1.4x-x-1)^2}{5.6x} \log n$. We want to write this as $\frac{m}{d}$ where d is a constant. Then $d = \frac{x}{\frac{(0.4x-1)^2}{5.6x}}$. Note that as x increases, d decreases to some asymptotic value:

$$\lim_{x\to\infty} \frac{x}{\frac{(0.4x-1)^2}{5.6x}} = \lim_{x\to\infty} \frac{5.6x^2}{(0.4x-1)^2} = \lim_{x\to\infty} \frac{5.6}{\left(0.4-\frac{1}{x}\right)^2} = \frac{5.6}{0.4^2} = 35$$

Then we have that $\frac{(2pm-m-R)^2}{8pm} \geq \frac{m}{35}$. Therefore,

$$Pr[X < \frac{m+R}{2}] \leq e^{-\frac{m}{35}}.$$

Thus, we have bounded the probability of error as required. ◀

We apply Lemma 8 to prove the bound on the full algorithm. Even though our lower bounds works when the error probability is constant, the algorithm applies even when the error is very small ($n^{-O(1)}$). We are now ready to present the proof for Theorem 7.

Proof. We prove separately the cases when $\delta \geq \frac{1}{n}$ and when $\delta < \frac{1}{n}$. In the first case, we set $m = 70 \log n$. By Lemma 8, the probability of not finding the correct t^{th} number is at most $e^{-\frac{70 \log n}{35}} = e^{\ln n^{-2/\ln 2}} < \frac{1}{n^2}$. Applying a union bound of this over the k numbers we need to find, the probability of error is at most $\frac{k}{n^2} \leq \frac{\sqrt{n}}{n^2} = \frac{1}{n^{1.5}}$ because $k \leq \sqrt{n}$. Since $\frac{1}{n^{1.5}} < \frac{1}{n} \leq \delta$, the probability of error is bounded as required. So we need in total $70k \log n$ steps of the random walk algorithm. Recall that each such step takes $O(k^2)$ queries. Therefore, in total, we have a running time of $O(70k^3 \log n) = O(k^3 \log n) = O(k^3 \log \frac{n}{\delta})$ since $\delta < 1$.

We now consider the case when $\delta < \frac{1}{n}$. Set $m = 70 \log \frac{1}{\delta}$. The probability of not finding the correct t^{th} number is at most $e^{-\frac{70 \log \frac{1}{\delta}}{35}} = e^{-\frac{2}{\ln 2} \ln \frac{1}{\delta}} < \delta^2$ by Lemma 8 (and that $\delta < 1$). Applying a union bound over the k numbers we need to find, the overall probability of error is $k\delta^2 < n\delta^2 < \delta$ as required. Thus, we need $O\left(k \log \frac{1}{\delta}\right)$ steps in the random walk, where each consists of $O(k^2)$ queries. Therefore, the total running time is $O\left(k^3 \log \frac{1}{\delta}\right) = O\left(k^3 \log \frac{n}{\delta}\right)$. ◀

References

1 T. Cover, J. Thomas. *Elements of Information Theory*, New York: John Wiley & Sons, Inc., 1991

2 U. Feige, P. Raghavan, D. Peleg, and E. Upfal. Computing with noisy information, *SIAM Journal on Computing*, Vol 23, No. 5, pp. 1001–1018, 1994

3 R. Karp, R. Kleinberg. Noisy binary search and its applications, *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 881–890, 2007

4 E. Lehman, T. Leighton. *Mathematics for Computer Science*, Online PDF file, 2004

5 A. Pelc. Solution of Ulam's problem on searching with a lie, *Journal of Combinatorial Theory*, Series A, Vol 44, No. 1, pp. 129–140, 1987

6 M. Raginsky. *Introduction: What is Statistical Learning Theory?*, Online PDF file, 2011

7 B. Ravikumar, K.B. Lakshmanan. Coping with known patterns of lies in a search game, *Theoretical Computer Science*, Volume 33, Issue 1, pp. 85–94, 1984.

8 J. Spencer. Guess a Number-with Lying, *Mathematics Magazine*, Vol. 57, No. 2, pp. 105–108, 1984

Pebbling, Entropy and Branching Program Size Lower Bounds

Balagopal Komarath*[1] and Jayalal Sarma M N[2]

1 **Department of Computer Science & Engineering, IIT Madras Chennai, India**
`baluks@cse.iitm.ac.in`

2 **Department of Computer Science & Engineering, IIT Madras Chennai, India**
`jayalal@cse.iitm.ac.in`

Abstract

We contribute to the program of proving lower bounds on the size of branching programs solving the Tree Evaluation Problem introduced in [4]. Proving an exponential lower bound for the size of the non-deterministic thrifty branching programs would separate NL from P under the thrifty hypothesis. In this context, we consider a restriction of non-deterministic thrifty branching programs called *bitwise-independence*. We show that any bitwise-independent non-deterministic thrifty branching program solving $\mathsf{BT}_2(\mathsf{h},\mathsf{k})$ must have at least $\frac{1}{2}k^{h/2}$ states. Prior to this work, lower bounds were known for general branching programs only for fixed heights $h = 2, 3, 4$ [4]. Our lower bounds are also tight (up to a factor of k), since the known[4] non-deterministic thrifty branching programs for this problem of size $O(k^{h/2+1})$ are bitwise-independent. We prove our results by associating a fractional pebbling strategy with any bitwise-independent non-deterministic thrifty branching program solving the Tree Evaluation Problem. Such a connection was not known previously even for fixed heights.

Our main technique is the entropy method introduced by Jukna and Zak[6] originally in the context of proving lower bounds for read-once branching programs. We also show that the previous lower bounds known[4] for deterministic branching programs for Tree Evaluation Problem can be obtained using this approach. Using this method, we also show tight lower bounds for any k-way deterministic branching program solving Tree Evaluation Problem when the instances are restricted to have the same group operation in all internal nodes.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Pebbling, Entropy Method, Branching Programs, Size Lower Bounds.

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.622

1 Introduction

The question whether L is a proper subset of P is one of the central problems in complexity theory. One of the approaches to the problem was proposed as a program by Cook [3] by introducing a suitable computational problem, namely the solvable path systems. The program suggests to prove lower bounds for increasingly stronger models of computation solving the solvable path systems problem. Indeed, for the specific problem, the attempt is to discover the structure of that restricted variant of the underlying computation process that captures the most natural, and if possible the most general, algorithmic strategies for

* Supported by the TCS Research Fellowship.

solving the problem. Cook [3] also proved super-logarithmic space lower bounds for marking machines solving the solvable path systems problem. Marking machines capture pebbling algorithms (which is a class of "natural" algorithms) solving this problem.

Barrington and Mckenzie [2] took a similar approach by considering the problem GEN and attempted to prove (upper and lower bounds) for increasingly stronger models of computation for solving GEN. Specifically, Barrington [2] considered "oracle" branching programs where each state of the branching program is allowed to ask a question about the input. For example, a general BP can ask queries of the form "What is the i^{th} bit of the input?" (This is called a branching program with BIT oracle.). Barrington [2] proved exponential size lower bounds for branching programs equipped only with certain "weak" oracles. Gal *et al* [5] considered incremental branching programs for solving GEN which can be thought of as branching programs trying to solve the GEN problem by incrementally finding the elements of the closure.

Cook *et al* [4] proposed the tree-evaluation problem for separating L and P and introduced thrifty branching programs as a model that captures "natural" algorithms solving the tree-evaluation problem. It is shown in [4] that deterministic thrifty branching programs exactly correspond to algorithms that implement black-pebbling. They also introduced the concept of fractional black-white pebbling and showed that non-deterministic thrifty branching programs can implement fractional black-white pebbling. It is also known that exponential size lower-bounds for deterministic semantic incremental branching programs solving the GEN problem follows from exponential size lower-bounds for deterministic thrifty branching programs solving a generalization of tree-evaluation problem called the DAG-evaluation problem [10].

Tree Evaluation Problem: We now briefly describe the tree-evaluation problem (see section 2 for a formal definition). An instance of the tree evaluation problem, $\mathsf{FT_d(h,k)}$, is a complete d-ary tree where each leaf is associated with an element from $[k]$ (which we think of as the value of the leaf node) and the i^{th} internal node is associated with a function $f_i : [k]^d \mapsto [k]$. The value of an internal node is obtained by applying this function to the values of its children. The output is the value of the root node. The corresponding Boolean version, $\mathsf{BT_d(h,k)}$, differs from $\mathsf{FT_d(h,k)}$ in that the function at the root node maps a value in $[k]^d$ to a value in $\{0,1\}$. An instance of $\mathsf{BT_d(h,k)}$ is called a "yes" instance if and only if the value of the root node is 1. Another variant of the tree-evaluation problem is the single function variant $\widehat{\mathsf{FT}}_\mathsf{d}(\mathsf{h,k})$ where the functions at all internal nodes are the same. A natural computational model for tree-evaluation problem is k-way branching program where each node queries the value of a single k-ary variable (i.e., the query is either i, where i is a leaf node, or $f_j(r,s)$, where j is an internal node and $r,s \in [k]$.). As observed in [4], any size lower bound of the form $\Omega(k^{r(h)})$ for k-way branching programs, where $r(h)$ is an unbounded function, would prove that $\mathsf{L} \neq \mathsf{P}$. We only consider k-way branching programs in this paper. Here we think of the parameter d as fixed and are interested in how the size of the branching programs solving $\mathsf{FT_d(h,k)}$ increases with h and k.

A natural algorithm to solve $\mathsf{FT_d(h,k)}$ is to evaluate the tree in a bottom-up fashion. This can be captured by the concept of black pebbling $\mathsf{T^h_d}$ (The complete d-ary tree of height h.). A black pebble on a node indicates that the value of the node is known. A black pebble can be placed on an internal node only if both its children are black pebbled. As a special case, a black pebble can be freely placed on any leaf node. It can be shown that h pebbles are necessary and sufficient for black pebbling $\mathsf{T^h_2}$. Since a value in $[k]$ can be represented using $\log k$ bits. This corresponds to a size bound of $\Theta(k^h)$ for branching programs. Similarly, fractional black-white pebbling captures natural non-deterministic algorithms solving $\mathsf{FT_d(h,k)}$. A white pebble can be freely placed on any node and corresponds to guessing the value of that node. However, to remove a white pebble from a node (this corresponds to verifying the guessed value) both its children have to be pebbled. Moreover, a branching program may compute or guess a fraction of bits of the values of nodes and this results in fractional black and white pebbles respectively.

A deterministic thrifty branching program is one in which the branching program is only allowed to query $f_j(r, s)$ when r and s are the values of the children of node j. Cook *et al.* [4] showed that deterministic thrifty branching programs solving $\mathsf{BT}_2(\mathsf{h}, \mathsf{k})$ require $\Omega(k^h)$ states by showing that such branching programs implement exactly a black pebbling strategy. Cook *et al.* [4] also proved tight lower bounds for non-deterministic thrifty branching programs for $h = 2, 3, 4$. They also show an upper-bound of $O(k^{h/2+1})$ for non-deterministic thrifty branching programs solving $\mathsf{FT}_2(\mathsf{h}, \mathsf{k})$. This shows that the non-deterministic variant is more powerful.

Our Results: In this paper, we show that computation of non-deterministic thrifty branching programs with an additional restriction that we call *bitwise-independence* can be associated with a fractional black-white pebbling sequence and therefore requires exponential size. The additional restriction of bitwise-independence is not too severe since all known upper-bounds using non-deterministic thrifty branching programs can be achieved using those with bitwise-independence property. In particular, the branching program described in [4] that achieve $O(k^{h/2+1})$ upper-bound satisfy bitwise-independence. Our main result is the first exponential lower bound for the above restriction of non-deterministic thrifty branching programs.

▶ **Theorem 1** (Main Theorem). *If B is a bitwise-independent non-deterministic thrifty branching program solving* $\mathsf{BT}_2(\mathsf{h}, \mathsf{k})$*, then B has at least $\frac{1}{2}k^{h/2}$ states.*

We associate these branching programs with fractional pebbling. Cook *et al.* [4] showed that if the tree $\mathsf{T}_\mathsf{d}^\mathsf{h}$ can be fractionally pebbled using p pebbles, then the corresponding (binary) Tree evaluation problem can be solved by a non-deterministic thrifty branching program of size $O(k^p)$. However, the converse direction is far from clear. We make progress in this direction and prove our lower bound by connecting bitwise-independent non-deterministic thrifty branching programs to fractional black-white pebbling sequences. We use the known result[8] (see also [4]) that $h/2 + 1$ pebbles are necessary and sufficient to pebble T_2^h using fractional black-white pebbling, to derive our lower bounds. We note that the lower bounds for $h = 2, 3, 4$ in [4] were not shown by associating it with fractional black-white pebbling.

Our main technique is a method proposed by Jukna and Zak [6] for proving size lower bounds for branching programs which they call the *entropy method.* Briefly, the method is to distribute a large set of inputs among the states of the branching program such that only a small number of inputs get mapped to any particular state. To achieve this, Jukna and Zak[6] proposed to consider the set F of inputs reaching that state and show that we can uniquely determine an input in F by a decision tree of low average depth (equivalently, the set F has low entropy.). It follows that there are a large number of states.

As our next contribution, we show that the lower bound proofs in [4] for k-way branching programs solving $\mathsf{FT}_2^3(\mathsf{k})$, $\mathsf{Children}_2^4(\mathsf{k})$ and thrifty branching programs solving $\mathsf{BT}_2(\mathsf{h}, \mathsf{k})$ can be obtained using this framework. Thus we get new simplified and unified views of the proofs for the following theorems.

▶ **Theorem 2.**
- *Any deterministic k-way branching program solving* $\mathsf{FT}_2^3(\mathsf{k})$ *must have at least k^3 states.*
- *Any deterministic k-way branching program solving* $\mathsf{Children}_2^4(\mathsf{k})$ *must have at least k^4 states.*

We then apply our method in a restricted setting where the functions at all internal nodes are given to be the same.

▶ **Theorem 3.** *Any deterministic k way branching program solving $\widehat{\mathsf{FT}}_2(\mathsf{h},\mathsf{k})$ with the functions at internal nodes restricted to a group operation must have at least $2^{h-2}k$ states.*

We observe that the above lower bound is tight. Indeed, when the internal operation is that of a group, the associativity property can be used to design branching programs of size $O(2^h k)$, when the function at the internal nodes is fixed. When the function at the internal nodes is also a part of the input, the upper bound is off by a factor of k, namely $O(2^h k^2)$.

The rest of the paper is organized as follows: In Section 2 we introduce the preliminaries needed for the paper. We prove the main result in section 3. Further applications of the entropy method are described in Section 4.

2 Preliminaries

For definitions of basic notions in complexity theory, we refer the reader to a standard textbook [1, 9]. We give the formal definition of the Tree Evaluation Problem first. In the following discussion, we label the nodes of the tree using usual heap numbering. The root node is labelled 1 and for each internal node i, its children are labelled $d(i-1)+2,\dots,di+1$. We use v_i to denote the value of the ith node in the input tree. When we are talking about a specific input I, we use v_i^I to denote the value of node i of the input I.

We now define the function and Boolean versions of the tree-evaluation problem.

▶ **Definition 4.** (Tree Evaluation Problems [4]) Input: The tree $\mathsf{T}_\mathsf{d}^\mathsf{h}$ where each leaf node is associated with a value from $[k]$ and each internal node i is associated with a function $f_i : [k]^d \mapsto [k]$, where $d, h, k \geq 2$

Output for $\mathsf{FT}_\mathsf{d}(\mathsf{h},\mathsf{k})$: The value $v_1 \in [k]$ of the root node, where in general $v_i = a$ if i is a leaf and a is the value associated with ith node in the input and $v_i = f_i(v_{j_1},\dots,v_{j_d})$ if i is a non-leaf node with nodes $j_1,\dots,j_d$ as children.

Output for $\mathsf{BT}_\mathsf{d}(\mathsf{h},\mathsf{k})$: The value $v_1 \in \{0,1\}$ of the root node. The evaluation rules are the same as for $\mathsf{FT}_\mathsf{d}(\mathsf{h},\mathsf{k})$.

It is known that tree-evaluation problems are in LOGDCFL [4] (For definition of LOGDCFL see [7].). Since we think of the parameter d as a constant, the input size is $O(d^h k^2 \log k)$. Since all the values in the definition of tree-evaluation problems are k-ary, a general model to solve tree-evaluation problem is a branching program that queries k-ary variables. Such branching programs are called k-way branching programs, since each query has k possible outcomes (depending on the value of the queried variable.). We define these models formally now. Throughout the technical sections of this paper, we use BP as short form for branching program.

▶ **Definition 5** (k-way BPs[4])**.** A non-deterministic k-way BP B for $\mathsf{FT}_\mathsf{d}(\mathsf{h},\mathsf{k})$ is a directed rooted multigraph. It consists of k final states labelled $1,\dots,k$. All other states of the BP are query states. A query state is labelled either i where i is a leaf node or labelled $f_i(x_1,\dots,x_d)$ where i is an internal node, $x_1,\dots,x_d \in [k]$, and each outgoing edge is labelled by an element from $[k]$. A computation path on input x is a directed path from the root (the start state) and each edge in the path is consistent with x. At least one such computation must end in a final state. The BP B is deterministic if and only if each query state has exactly k outgoing edges labelled $1, \ldots, k$.

A non-deterministic k-way BP B for $\mathsf{BT_d(h,k)}$ is defined similarly except that each query state labelled $f_1(x_1, \ldots, x_d)$ where $x_1, \ldots, x_d \in [k]$ has all of its outgoing edges labelled by either 0 or 1. There are two final states labelled 0 and 1. The BP B is deterministic if and only if each query state labelled $f_1(x_1, \ldots, x_d)$ has exactly two outgoing edges labelled 0 and 1 and every other query state has exactly k outgoing edges labelled $1, \ldots, k$. The BP B solves $\mathsf{BT_d(h,k)}$ if and only if for every "yes" instance has at least one accepting computation path and every "no" instance has no accepting computation path.

By a sub-BP B' of B obtained by restricting input set E to E', we refer to the BP obtained from B by removing edges not used by inputs in E' and by shortcutting states for which only one outgoing edge can be active when we consider computation on instances in E'.

The size of binary binary for solving tree-evaluation problems differ from the size of k-way BPs by a factor of at most k. Therefore, a size lower bound of $\Omega(k^{r(h)})$ for k-way branching programs, where $r(h)$ is an unbounded function, would separate L from $\mathsf{LOGDCFL}$.

▶ **Definition 6** (Thrifty BPs [4]). A non-deterministic BP solving $\mathsf{BT_d(h,k)}$ is called *thrifty* if and only if for any accepting computation path on instance I any query state labelled $f_i(x_{d(i-1)+2}, \ldots, x_{di+1})$ satisfies $x_j = v_j^I$ for $d(i-1)+2 \leq j \leq di+1$ (i.e., the internal nodes are queried only at the correct values of its children.).

2.1 Pebbling

Pebbling sequences capture "natural" algorithms that solve the tree-evaluation problems that evaluate the values at nodes of the tree in a bottom-up fashion. A black pebble value at a node indicates the fraction of the value at the node that is known to the BP. Similarly, a white pebble indicates the fraction of the value at the node guessed by the BP (respectively, the fraction of value at the node that needs to be verified by the BP). For example, a black pebble value of 1 indicates that value is completely known and a white pebble value of 1 indicates that the value was guessed from $[k]$. In order to compute or guess (a fraction of) the value at any node, the BP must completely figure out (by computing or guessing) the values of its children.

▶ **Definition 7** (Fractional Black-White Pebbling [4]). A fractional pebbling configuration on a rooted d-ary tree T is an assignment of a pair of real numbers $(b(i), w(i))$ to each node i of the tree. The values $b(i)$ and $w(i)$ are called the black and white pebble values, respectively, of node i. We have for every i

$$\begin{aligned} b(i) + w(i) &\leq 1 \\ 0 \leq b(i), w(i) &\leq 1 \end{aligned} \tag{1}$$

The legal pebble moves are as follows.

1. For any node i, decrease $b(i)$ arbitrarily.
2. For any node i, increase $w(i)$ arbitrarily.
3. For any node i, if each child of i has pebble value 1, then decrease $w(i)$ to 0.
4. For any node i, if each child of i has pebble value 1, then increase $b(i)$ arbitrarily and simultaneously decrease the black pebble value of children of i.

The number of pebbles in a configuration is the sum over all nodes i of $b(i) + w(i)$. A fractional pebbling of T using p pebbles is a sequence of (legal) fractional pebbling moves on nodes of T which starts and ends with each node having pebble value 0 and at some point the root node has black pebble value 1, and no configuration has more than p pebbles.

A black pebbling is a fractional black-white pebbling such that for all i the black pebble value $b(i)$ always takes values from $\{0, 1\}$ and $w(i) = 0$.

It is known that $h/2+1$ pebbles are necessary and sufficient to pebble $\mathsf{T}_2^{\mathsf{h}}$ using fractional black-white pebbling [8].

2.2 Entropy Method

We now formally describe the entropy method introduced in [6]. We specialize the definition slightly to suit our application of the method. Let B be a BP computing the characteristic function of language L. Let A be a particular set of inputs. Define a "distribution" function $g : A \mapsto States(B)$. Now consider an arbitrary state s in the range of g and let $F = g^{-1}(s)$. Define a decision tree D such that each $a \in F$ reaches a unique leaf in D. Such a decision tree is called a 'splitting tree' for F in [6]. The next step is to prove that D has low depth which will imply that the entropy of F, $h(F) = \log |F|$, is small. Then we have $Size(B) \geq 2^{|A|-h(F)}$. In defining A and g, we may use properties of L and any restrictions imposed on the structure of B. The goal is to minimize the maximum value of $h(F)$ over all choices of F by using an A that is as large as possible.

For the rest of our discussion, we fix $d = 2$. However all our arguments can be easily generalized to arbitrary constant d.

2.3 Bitwise-independent Non-deterministic Thrifty BPs

We use N to denote the total number of non-root nodes in $\mathsf{T}_2^{\mathsf{h}}$. Let B be a non-deterministic thrifty BP for $\mathsf{BT}_2(\mathsf{h}, \mathsf{k})$. Let s be a state of B. We define

$$F_s = \{(v_2^I, \ldots, v_{N+1}^I) \colon \exists I \text{ and a computation path } C(I) \text{ such that } s \in C(I)\}$$
$$A_s = \{(v_2^I, \ldots, v_{N+1}^I) \colon \exists I \text{ and an accepting computation path } C(I) \text{ such that } s \in C(I)\}$$

We use $\pi(S, i)$ to denote the set of all i^{th} component of the tuples in S (typically, S is either F_s or A_s for some s.). That is, the set formed by projecting the i^{th} component of all tuples in S. For any encoding function $\phi : [k] \mapsto \{0, 1\}^{\lceil \log k \rceil}$, we use $(r)_i$ to denote the i^{th} bit of $r \in [k]$ when r is encoded using ϕ.

▶ **Definition 8** (Bitwise-independent Non-deterministic Thrifty BPs). Let $k = 2^\ell$ and let B be a non-deterministic thrifty BP solving $\mathsf{BT}_2(\mathsf{h}, \mathsf{k})$. Then B is bitwise-independent if and only if there exists an encoding function $\phi : [k] \mapsto \{0, 1\}^\ell$ such that for every state s in B the following two conditions are satisfied.

$$F_s = \bigtimes_{i=2}^{N+1} \phi^{-1} \left(\bigtimes_{j=1}^{\ell} (\pi(F_s, i))_j \right)$$
$$A_s = \bigtimes_{i=2}^{N+1} \phi^{-1} \left(\bigtimes_{j=1}^{\ell} (\pi(A_s, i))_j \right)$$

Where we think of the first $\times$ as the normal Cartesian product and the second one (over all the bits) as concatenating all the bits after forming the Cartesian product. When k is not a power of two, we consider the largest power of two smaller than k. Let this be 2^ℓ. Then B is bitwise-independent if and only if the above two conditions are satisfied with equality replaced by superset.

The intuition is that at any state the bits of values of non-root nodes can be partitioned into "fixed" bits and "unfixed" bits and the sets F_s and A_s are such that all possible combinations of unfixed bits are in the set. i.e., the BP cannot store implicit information about bits (such as, the second bit is the complement of the first bit).

If we only consider minimal bitwise-independent non-deterministic thrifty BPs, then we have $|F_s|, |A_s| \geq 1$ for any query state s. This is because any query state s that does not have any accepting computation path passing through it can be merged with the reject state. Also note that by the definition of bitwise independence, for any i and s, we have $\pi(F_s, i)$ and $\pi(A_s, i)$ are always powers of two when k is a power of two.

We now define the pebbling values assigned to a non-root node i at state s of the BP. These pebbling values are referred to as "actual" pebble values.

$$\begin{aligned} b(i,s) &= 1 - \log_k |\pi(F_s, i)| \\ w(i,s) &= \log_k \frac{|\pi(F_s, i)|}{|\pi(A_s, i)|} \end{aligned} \tag{2}$$

3 Lower Bounds for Bitwise-independent Non-deterministic Thrifty BPs

In this section, we prove exponential size lower bounds for bitwise-independent non-deterministic thrifty BPs. The proof uses the entropy method. We consider the input set E for the problem $\mathsf{BT}_2(\mathsf{h},\mathsf{k})$, where each leaf node is allowed to take values from $[k]$ and for each instance I and each internal node i, we allow $f_i(v^I_{2i}, v^I_{2i+1})$ to take any value from $[k]$ and restrict it to 1 elsewhere. We also fix $f_1(v^I_2, v^I_3) = 1$ and 0 elsewhere so that $|E| = k^N$. Note that all instance $I \in E$ are "yes" instances. The idea is to map an accepting computation path on $I \in E$ into a valid fractional black-white pebbling sequence. To this end, we define a set of critical states for each node on an accepting computation path of $I \in E$. The distribution function then maps each input to the state where the pebble value is maximum (which will imply that the number of inputs is small).

We now show that our definition of pebble values satisfy the restrictions imposed on pebble values by (1).

▶ **Claim 9.** *For any non-root node i and state s, $0 \leq b(i,s), w(i,s) \leq 1$.*

▶ **Claim 10.** *For any non-root node i and state s, $b(i,s) + w(i,s) \leq 1$.*

The following claim establishes the fact that if the total pebble value of the tree (in non-root nodes) is high at a state, then there are only a few inputs on an accepting computation path reaching that state. In other words, if the pebble value at a point of the computation is high, then the entropy at that point is low.

▶ **Claim 11.** *If the total pebble value of the non-root nodes of the tree at a state s is p, then the number of "yes" instances reaching s on an accepting computation path is k^{N-p}.*

Proof. Consider a particular non-root node i. Assume that the total pebble value at i is p_i. From this we have $1 - \log_k |\pi(F_s, i)| + \log_k \frac{|\pi(F_s,i)|}{|\pi(A_s,i)|} = p_i$. Therefore $|\pi(A_s, i)| = k^{1-p_i}$. Now by simple counting the total number of inputs on an accepting computation path is $k^{\sum_{i=2}^{N+1}(1-p_i)} = k^{N-p}$. ◀

Consider an input $I \in E$ and an accepting computation path $C(I)$ for I. Our aim is to define a sequence of critical states $(\gamma_0 =)s_0, s_1, \ldots, s_{t+1}(= acc)$ (s_0 and s_{t+1} are always start and final states of the BP) and associate a fractional pebbling configuration with each critical state. The sequence thus obtained will be a valid fractional pebbling sequence and it will be defined such that the pebbling values at each node will underestimate the actual pebbling values defined by (2).

Critical States

We now define the sequence of critical states on the accepting computation path $C(I)$. The critical state for the root node is the last state querying the root node. Every non-root node j may have multiple critical states. Let s denote *a* critical state of parent of j. If $b(j, s) > 0$, then the last node querying node j before s is a critical state for j. If $w(j, s) > 0$, then the first node querying node j after s is a critical state for j.

Pebble Configurations

We now define the sequence of pebble configurations associated with critical states on an accepting computation path of input I. The black pebble value of the root node becomes 1 immediately after its critical state and remains so until the end of the computation where it becomes 0. Now we define the pebble values of an arbitrary non-root node j. Let s' be a critical state for j', the parent of j. If $b = b(j, s') > 0$, then this black pebble value must have increased from 0 to b at some point of computation. Now consider the critical state s for j before s' as defined before. The black pebble value of node j is increased from 0 to b at the critical state immediately following s. Similarly, if $w = w(j, s') > 0$, then this white pebble value must decrease from w to 0 at some point of computation. Now consider the critical state s for j after s' as defined before. The white pebble value is reduced from w to 0 from the critical state immediately following s.

The following claims about the validity of the starting and ending pebbling configurations are easily proved.

▶ **Claim 12.** *The start state has an empty pebbling configuration.*

▶ **Claim 13.** *The accepting state has an empty pebbling configuration.*

The following lemmas establish the fact that the pebbling sequence defined above is a valid pebbling sequence.

▶ **Lemma 14.** *Let s be a critical state for node j, then both of j's children are fully pebbled at s.*

Proof. Let s query $f_j(u, v)$. We have $\pi(A_s, 2j) = \{u\}$ (and $\pi(A_s, 2j + 1) = \{v\}$) by the thrifty property. Then $b(2j, s) + w(2j, s) = 1 - \log_k |\pi(F_s, 2j)| + \log_k \frac{|\pi(F_s,2j)|}{|\pi(A_s,2j)|} = 1$ (and similarly for $2j + 1$). ◀

▶ **Lemma 15.** *If the black pebble value of node j is increased or the white pebble value of node j is decreased at state s, then both its children are fully pebbled at the critical state immediately before s.*

Proof. For a node j, the black pebble value is increased or the white pebble value is decreased only at the critical state immediately following a critical state for j. By Lemma 14 both children of node j are fully pebbled at this critical state. ◀

The following is our key technical lemma and establishes the fact that the pebbling values defined for the critical states never overestimate the actual pebbling values of nodes.

▶ **Lemma 16.** *Let b and w be the pebble values defined at state s for an arbitrary non-root node $2j$, then $b \leq b(2j, s)$ and $w \leq w(2j, s)$.*

Proof. The proof is divided into two parts. First, we show that the black pebble values are never overestimated. Then we show that white pebble values are never overestimated.

We consider an arbitrary state s at which the black pebble value of node $2j$ is defined as $b > 0$. Note that the black pebble value of a non-root node $2j$ is non-zero if and only if there exists a critical state for the parent of $2j$ at which the actual pebble value of $2j$ is b. Therefore, there exists a state s_{2j} that is a critical state for $2j$ before s and s_j that is a critical state for j, the parent of $2j$, after s (with $s = s_j$ possibly.). Now suppose that the actual black pebble value for node $2j$ at state s is $b(2j, s)$ and that $b(2j, s) < b$.

$$\begin{aligned} 1 - \log_k |\pi(F_s, 2j)| &< b \\ \implies |\pi(F_s, 2j)| &> k^{1-b} \end{aligned}$$

Now by the independence assumption we may conclude that there are more than k^{1-b} inputs that differ only at the value of node $2j$. By the definition of critical states, there does not exist any node querying $2j$ in $C(I)$ from s to s_j. All these inputs can follow the same path to the critical state s_j. Therefore, the black pebble value is $b(2j, s_j) < b$, a contradiction.

It remains to prove that white pebble values are never overestimated. We will prove that the white pebble value of a node $2j$ is at least the estimated value w between all states from s_j to s_{2j} (both inclusive). Here s_j is a critical state for j at which node $2j$ acquired a white pebble value of w and s_{2j} is the critical state for $2j$ after which this pebble value is removed. In order to prove this, it is sufficient to prove that the ratio $\frac{f'}{a'} = \frac{|\pi(F_{s'}, 2j)|}{|\pi(A_{s'}, 2j)|}$ for any state s' is greater than the corresponding ratio $\frac{f}{a}$ at state s_j, where s' is a state on $C(I)$ in the segment from s_j to s_{2j}. By the independence argument, we have $f' \geq f$ by taking projections of all f inputs that differ from I only at node $2j$. We will show that if $a' > a$, then $f' > f$ by an appropriate amount so that the ratio is not reduced.

Since the white pebble value is acquired at state s_j, we have $w(2j, s_j) = w$. Now consider a state s' (Possibly equal to s_{2j}) on the segment of the computation path $C(I)$ between s_j and s_{2j}. Our aim is to prove that $w \leq w(2j, s')$. Let $f = |\pi(F_{s_j}, 2j)|$, $f' = |\pi(F_{s'}, 2j)|$, $a = |\pi(A_{s_j}, 2j)|$ and $a' = |\pi(A_{s'}, 2j)|$. First of all note that $f' \geq f$ since there are no nodes querying $2j$ from s_j to s_{2j} and the independence property guarantees f inputs that differ from I only at node $2j$ will reach s'. Now we will show that whenever $a' > a$, f' is greater than f by the same multiplicative factor. Note that both f and a are powers of two. By the assumption of bit-wise independence, we can partition bits of node $2j$ into "fixed" bits and "unfixed" bits for any F_s (and A_s). The only way to add elements to these sets are by unfixing bits. Let us assume that exactly one more bit became unfixed in $\pi(A_{s'}, 2j)$. So $a' = 2a$.

Let r' be a value in $\pi(A_{s'}, 2j) \setminus \pi(A_{s_j}, 2j)$. We claim that $r' \notin \pi(F_{s_j}, 2j)$. We will prove this by contradiction. Suppose $r' \in \pi(F_{s_j}, 2j)$, then by the independence property there is an input J which is the same as I except that $v^J_{2j} = r'$ reaches s' through s_j. Since $r' \in \pi(A_{s'}, 2j)$, there is an accepting path for J through s_j. This accepting path is obtained by using the independence property of $A_{s'}$ and the fact that an accepting computation for I passes through s'. But this path makes a non-thrifty query at s_j. Therefore $r' \notin \pi(F_{s_j}, 2j)$

as claimed. Since $r' \in \pi(F_{s'}, 2j)$, at least one bit must have become unfixed. But this implies $f' \geq 2f$. This proof can be easily extended to the case where $a' = 2^m a$ for any m. ◀

We now prove our main result by associating an accepting computation on input $I \in E$ to a valid fractional black-white pebbling sequence.

▶ **Theorem 17.** *If B is a bitwise-independent non-deterministic thrifty BP solving* $\mathsf{BT}_2(\mathsf{h}, \mathsf{k})$*, then B has at least $\frac{1}{2}k^{h/2}$ states.*

Proof. Assume that k is a power of two. We now apply the entropy method described in subsection 2.2. Our input set is the set E described previously. We now describe our distribution function h. Recall that each instance I in E is a "yes" instance and therefore guaranteed to have an accepting computation path $C(I)$ in B. As we have already seen, we can identify a sequence of critical states in $C(I)$ and associate a fractional black-white pebbling configuration with each critical state such that the sequence of fractional black-white pebbling configurations form a valid fractional pebbling of $\mathsf{T}_2^{\mathsf{h}}$ (See Claims 9, 10, 12, 13, 14, and 15). But we know that any valid fractional black-white pebbling of $\mathsf{T}_2^{\mathsf{h}}$ must have a configuration with at least $h/2$ pebbles on non-root nodes [8]. Let s be the critical state in $C(I)$ that corresponds to this configuration. Our distribution function h maps I to s. Now consider an arbitrary state s in $range(h)$ and consider the set $G_s = h^{-1}(s)$. By Claim 11, we have $|G_s| \leq k^{N-h/2}$. It follows that B has at least $k^{h/2}$ states. (Intuitively, if we consider the splitting tree for G_s, we can determine the input by querying only those bits that are "unfixed" at s.)

When k is not a power of two, we consider the highest power of two (2^ℓ) smaller than k. Consider the sub-BP of B that solves $\mathsf{BT}_2(\mathsf{h}, \mathsf{k})$ when the values are from the set $[2^\ell]$. By definition of bitwise-independence and the lower bound when k is a power of two, we have that this sub-BP of B has at least $2^{\ell^{h/2}} > \frac{1}{2}k^{h/2}$ states. ◀

▶ Remark. We note that the lower-bound proof in [4] for deterministic thrifty BPs can be obtained by specializing our argument to deterministic thrifty BPs. Specifically, we define the black pebble value of a node as 1 if and only if its value is known. The critical state for the root node is the last state querying root and critical state for other nodes j are those states which query j and immediately precedes the critical state for j's parent. The fact that the computation follows a valid black pebbling can be argued using thriftiness (bitwise-independence is not required.). We then map each input to the state that has h or more pebbles. The lower bound follows.

4 Lower Bounds for Deterministic BPs Using Entropy Method

In this section, we show that many lower-bound proofs in [4] can be derived using the entropy method and derive some new applications of the method. We believe that reformulating it in terms of the entropy method makes the connection more explicit.

▶ **Theorem 18.** *([4]) Any deterministic k-way BP solving* $\mathsf{FT}_2^3(\mathsf{k})$ *has at least k^3 states.*

Proof. First, we will consider a k-way BP that takes two inputs $u, v \in [k]$ and computes $u +_k v$ where $+_k$ is addition modulo k (appropriately defined on $[k]$.). We will prove a size lower-bound of k states for this problem. Then we will use this result to prove the theorem.

Let B be a k-way BP solving the above problem. We apply the entropy method to prove the required size lower-bound. Our input set A consists of k^2 inputs (all inputs). Our distribution function maps each input in A to the last edge in the k-way BP B solving this problem. Now consider an arbitrary edge e labelled r and connecting a state labelled (w.l.g.) u to the output state s. Now consider the set of inputs F_e reaching this edge. The only possible inputs are those with $u = r$ and $u +_k v = s$. But this implies that $v = s -_k r$. Therefore $F_e = \{(r, s -_k r)\}$ has cardinality one. Since the choice of e was arbitrary, we have $Edges(B) \geq k^2/1 = k^2$. Since we are considering k-way BPs where each state has exactly k outgoing edges $States(B) \geq k$.

Consider the sub-problem of $\mathsf{FT}_2^3(\mathsf{k})$ where $f_1 = +_k$, leaves are allowed to take arbitrary values, and for any input I, we allow $f_j^I(v_{2j}^I, v_{2j+1}^I)$ for $j = 2, 3$ to take arbitrary values and restrict it to 1 elsewhere. Consider a k-way BP B solving this problem. Now consider the sub-BP b' obtained from B by fixing $(v_4, v_5) = (v_6, v_7) = (r, s)$ for some $r, s \in [k]$. Note that the sub-BP B' computes $u +_k v$ for $u = f_2(r, s)$ and $v = f_3(r, s)$ and therefore must have at least k states. Now the set of all states querying f_2 or f_3 in B is the disjoint union of all states querying $f_2(r, s)$ and $f_3(r, s)$ for k^2 (r, s) pairs. Therefore $States(B) \geq k^3$ as claimed. ◀

The $\mathsf{Children}_2^4(\mathsf{k})$ problem is the same as $\mathsf{FT}_2^4(\mathsf{k})$ problem except that the tree has no root node and the values at nodes 2 and 3 together is defined as the output.

▶ **Theorem 19.** *([4]) Any deterministic k-way BP solving* $\mathsf{Children}_2^4(\mathsf{k})$ *has at least k^4 states.*

Proof. Consider a k-way BP that takes four inputs u, v, w, x and computes the tuple $(u +_k v, w +_k x)$. We will prove a size lower-bound of k^2 states for this problem and argue that the theorem follows from this result.

Let B be a deterministic k-way BP solving this problem. We now apply the entropy method. Our input set A is the set of all inputs and therefore $|A| = k^4$. Our distribution function will map each input in A to the last edge in its computation path on B. Consider an arbitrary edge e labelled r that connects a query state labelled u to the output state (s, t). Now consider the set of inputs F_e that get mapped to e. We have $u = r$, $v = s -_k r$, and $w +_k x = t$. Since there are exactly k inputs that satisfy these conditions $|F_e| \leq k$. Therefore $Edges(B) \geq k^4/k = k^3$ and it follows that $States(B) \geq k^2$.

Consider the sub-problem of $\mathsf{Children}_2^4(\mathsf{k})$ where $f_2 = f_3 = +_k$, leaves are allowed to take arbitrary values, and for any input I, we allow $f_j^I(v_{2j}^I, v_{2j+1}^I)$ for $j = 4, 5, 6, 7$ to take arbitrary values and restrict it to 1 elsewhere. Consider a k-way BP B solving this problem. Now consider the sub-BP B' obtained from B by fixing values of sibling leaves to (r, s). Note that the sub-BP B' solves the problem discussed in the previous paragraph and hence requires k^2 states. As before, since the level 2 query states of B are the disjoint union of query states for k^2 distinct (r, s) pairs, we have $States(B) \geq k^4$. ◀

We now present a new lower-bound of $\Omega(2^h k)$ for $\widehat{\mathsf{FT}}_2(\mathsf{h}, \mathsf{k})$ problem when the function at internal nodes are restricted to a group operation. This forms a special case of the general problem.

▶ **Theorem 20.** *Any deterministic k way BP solving* $\widehat{\mathsf{FT}}_2(\mathsf{h}, \mathsf{k})$ *with the functions at internal nodes restricted to a group operation has at least $2^{h-2}k$ states.*

Proof. Assume without loss of generality that the functions at internal nodes are $+_k$. The leaf nodes are labelled $x_1 = 2^{h-1}, \ldots, x_{2^{h-1}} = 2^h - 1$. Let B be a deterministic k-way BP solving this problem. Now consider the sub-BP B' obtained from B by fixing $x_3, \ldots, x_{2^{h-1}}$ to 1. The sub-BP B' computes $x_1 +_k x_2$ and therefore has at least k states. A similar argument can be applied to each pair of leaves. Since there are 2^{h-2} disjoint pairs of leaves, the BP B must have at least $2^{h-2}k$ states. ◀

Upper Bounds: We observe upper bounds for the size of branching programs computing $\widehat{\mathsf{FT}}_2(\mathsf{h}, \mathsf{k})$ problem when the function at internal nodes are restricted to a group operation. The associativity of the group operation implies upper bounds. We now briefly describe a BP for this problem. The BP is a layered BP of width k. The BP evaluates the group product in a left-associative fashion. In order to do this, the BP only has to remember the value of the product $v_1 \ldots v_{i-1}$ in the i^{th} layer. This value is in $[k]$ and can be remembered using width k. Then, in the i^{th} layer, the BP reads v_i and moves to $i + 1^{\text{st}}$ layer updating the remembered value as required. There are two variations possible in this setting. In the first one, the function at the internal nodes is fixed. In this case the branching program described will be of size $2^h k$ and hence Theorem 20 is tight. In the second version, when the function at the internal node is also a part of the input, the described method will give an upper bound of $2^h k^2$ (since we also have to query the function values).

References

1 S. Arora and B. Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, 2009.

2 David A.Mix Barrington and Pierre McKenzie. Oracle Branching Programs and Logspace versus P. *Information and Computation*, 95(1):96 – 115, 1991.

3 Stephen A. Cook. An observation on time-storage trade off. *Journal of Computer and System Sciences*, 9(3):308–316, 1974.

4 Stephen A. Cook, Pierre McKenzie, Dustin Wehr, Mark Braverman, and Rahul Santhanam. Pebbles and Branching Programs for Tree Evaluation. *ACM Transactions on Computation Theory (TOCT)*, 3(2):4:1–4:43, 2012.

5 Anna Gal, Michal Koucky, and Pierre McKenzie. Incremental Branching Programs. *Theory of Computing Systems*, 43(2):159–184, April 2008.

6 S. Jukna and S. Žák. On Uncertainty versus Size in Branching Programs. *Theoretical Computer Science*, 290(3):1851–1867, January 2003.

7 I. H. Sudborough. On the Tape Complexity of Deterministic Context-free Languages. *Journal of ACM*, 25(3):405–414, July 1978.

8 Frank Vanderzwet. Fractional Pebbling Game Lower Bounds, December 2011. http://www.cs.toronto.edu/ fvan/mt11.pdf.

9 H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach.* Springer New York Inc., 1999.

10 Dustin Wehr. Lower bound for Deterministic Semantic-incremental Branching Programs Solving GEN. *CoRR*, abs/1101.2705, 2011.

Advice Lower Bounds for the Dense Model Theorem*

Thomas Watson[1]

1 **University of California, Berkeley**
`tom@cs.berkeley.edu`

Abstract

We prove a lower bound on the amount of nonuniform advice needed by black-box reductions for the Dense Model Theorem of Green, Tao, and Ziegler, and of Reingold, Trevisan, Tulsiani, and Vadhan. The latter theorem roughly says that for every distribution D that is δ-dense in a distribution that is ϵ'-indistinguishable from uniform, there exists a “dense model” for D, that is, a distribution that is δ-dense in the uniform distribution and is ϵ-indistinguishable from D. This ϵ-indistinguishability is with respect to an arbitrary small class of functions F. For the natural case where $\epsilon' \geq \Omega(\epsilon\delta)$ and $\epsilon \geq \delta^{O(1)}$, our lower bound implies that $\Omega\big(\sqrt{(1/\epsilon)\log(1/\delta)} \cdot \log|F|\big)$ advice bits are necessary. There is only a polynomial gap between our lower bound and the best upper bound for this case (due to Zhang), which is $O\big((1/\epsilon^2)\log(1/\delta)\cdot\log|F|\big)$. Our lower bound can be viewed as an analog of list size lower bounds for list-decoding of error-correcting codes, but for “dense model decoding” instead. Our proof introduces some new techniques which may be of independent interest, including an analysis of a majority of majorities of p-biased bits. The latter analysis uses an extremely tight lower bound on the tail of the binomial distribution, which we could not find in the literature.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Pseudorandomness, advice lower bounds, dense model theorem

Digital Object Identifier 10.4230/LIPIcs.STACS.2013.634

1 Introduction

The question of whether the prime numbers contain arbitrarily long arithmetic progressions was a long-standing and famous open problem until Green and Tao [9] answered the question in the affirmative in a breakthrough paper in 2004. A key ingredient in their proof is a certain transference principle which, very roughly, states the following. Let U denote the set of positive integers. Then for every $D \subseteq U$, if there exists an $R \subseteq U$ such that D is dense in R and R is “indistinguishable” from U, then there exists an $M \subseteq U$ such that M is dense in U and D is “indistinguishable” from M. Tao and Ziegler [13] proved a much more general version of the transference principle, which has come to be known as the Dense Model Theorem (since M is a dense “model” for D).

Reingold, Trevisan, Tulsiani, and Vadhan [12] demonstrated the relevance of the Dense Model Theorem to computer science, and they gave a new proof which is much simpler and achieves better parameters than the proof of Green, Tao, and Ziegler. Gowers [6] independently came up with a similar proof. In addition to the original application of

* This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-0946797 and by the National Science Foundation under Grant No. CCF-1017403.

$$\begin{tikzcd}
D \arrow[r, "\delta\text{-dense}"] \arrow[d, leftrightarrow, "(\epsilon, F)\text{-indistinguishable}"'] & R \arrow[d, leftrightarrow, "(\epsilon', F')\text{-indistinguishable}"] \\
M \arrow[r, "\delta\text{-dense}"'] & U
\end{tikzcd}$$

Figure 1 Relations among distributions in the Dense Model Theorem

showing that the primes contain arbitrarily long arithmetic progressions, the Dense Model Theorem has found applications in differential privacy [11], pseudoentropy and leakage-resilient cryptography [2, 12, 3], and graph decompositions [12], as well as further applications in additive combinatorics [7, 8]. Subsequent variants of the Dense Model Theorem have found applications in cryptography [5] and pseudorandomness [14].

To formally state the Dense Model Theorem, we first need some definitions. We identify $\{0,1\}^{2^n}$ with the set of functions from $\{0,1\}^n$ to $\{0,1\}$. We use $\mathcal{D}_n$ to denote the set of all distributions on $\{0,1\}^n$. The domain $\{0,1\}^n$ could be replaced by any finite set of size 2^n; we use the domain $\{0,1\}^n$ for concreteness.

▶ **Definition 1.** We say $D_1 \in \mathcal{D}_n$ is δ-*dense* in $D_2 \in \mathcal{D}_n$ if for all $x \in \{0,1\}^n$, $\Pr_{D_1}[x] \leq \frac{1}{\delta}\Pr_{D_2}[x]$.

▶ **Definition 2.** We say $f \in \{0,1\}^{2^n}$ ϵ-*distinguishes* $D_1, D_2 \in \mathcal{D}_n$ if $\left|\mathrm{E}_{D_1}[f] - \mathrm{E}_{D_2}[f]\right| > \epsilon$.

▶ **Definition 3.** For $F \subseteq \{0,1\}^{2^n}$, we say $D_1, D_2 \in \mathcal{D}_n$ are (ϵ, F)-*indistinguishable* if there is no $f \in F$ that ϵ-distinguishes D_1 and D_2.

The following is quantitatively the best known version of the theorem, due to Zhang [15] (building on [12, 1]).

▶ **Theorem 4** (Dense Model Theorem). *For every $F \subseteq \{0,1\}^{2^n}$ and every $D \in \mathcal{D}_n$, if there exists an $R \in \mathcal{D}_n$ such that D is δ-dense in R and (R, U) are (ϵ', F')-indistinguishable where $U \in \mathcal{D}_n$ is the uniform distribution, then there exists an $M \in \mathcal{D}_n$ such that M is δ-dense in U and (D, M) are (ϵ, F)-indistinguishable, where $\epsilon' \geq \Omega(\epsilon\delta)$ and F' consists of all linear threshold functions with ± 1 coefficients applied to $O\big((1/\epsilon^2)\log(1/\delta)\big)$ functions from F.*

The relations among the four distributions in Theorem 4 are illustrated in Figure 1. We remark that the theorem also holds when we allow $[0,1]$-valued functions f rather than just $\{0,1\}$-valued functions f. The proof of [12] gives the same result but where $O\big((1/\epsilon^2)\log(1/\epsilon\delta)\big)$ functions from F are combined to get a function from F'. The original proof of [13] achieves an F' which is qualitatively simpler, namely all products of $\mathrm{poly}(1/\epsilon, 1/\delta)$ functions from F, but it only achieves $\epsilon' \geq \exp(-\mathrm{poly}(1/\epsilon, 1/\delta))$.[1] We note that the dependence $\epsilon' \geq \Omega(\epsilon\delta)$ is tight in two senses.

- The Dense Model Theorem is actually false when $\epsilon' > \epsilon\delta$, even if $F' = \{0,1\}^{2^n}$. See [15] for the simple argument.
- The following converse to the Dense Model Theorem holds: If there exists an $M \in \mathcal{D}_n$ such that M is δ-dense in U and (D, M) are (ϵ, F)-indistinguishable, then there exists

[1] Another proof that also achieves this is given in [12].

an $R \in \mathcal{D}_n$ such that D is δ-dense in R and (R, U) are (ϵ', F')-indistinguishable, where $\epsilon' = \epsilon\delta$ and $F' = F$. To see this, note that $U = \delta M + (1-\delta)\widehat{M}$ for some $\widehat{M} \in \mathcal{D}_n$, so we can let $R = \delta D + (1-\delta)\widehat{M}$; then D is δ-dense in R, and for every $f \in \{0,1\}^{2^n}$ we have $\mathrm{E}_R[f] - \mathrm{E}_U[f] = \delta\big(\mathrm{E}_D[f] - \mathrm{E}_M[f]\big)$ and thus if $\big|\mathrm{E}_R[f] - \mathrm{E}_U[f]\big| > \epsilon'$ then $\big|\mathrm{E}_D[f] - \mathrm{E}_M[f]\big| > \epsilon$.

The Dense Model Theorem has an undesirable feature: The class F' is more complex than the class F. Thus, if we wish to conclude that D and M are indistinguishable for a class F, we need to assume that R and U are indistinguishable for a more complex class F'. The less complex F' is, the stronger the theorem is. The reason for this loss in complexity is because the theorem is proved using a *black-box reduction.* In other words, the contrapositive is proved: We assume that for every M δ-dense in U there exists a function from F that ϵ-distinguishes D and M, and we show that some of these functions can be plugged into the reduction to get a function that ϵ'-distinguishes R and U. Thus the resulting function is necessarily more complex than the functions that get plugged into the reduction. There are three notions of complexity that are interesting to address here.

1. *Computational complexity.* If F consists of functions computed by small constant-depth circuits (AC^0), then can we let F' consist of functions computed by (slightly larger) constant-depth circuits? This is not known to be true when $\epsilon' \geq \Omega(\epsilon\delta)$, because the reductions of [12, 15] involve a linear threshold function, which cannot be computed by small constant-depth circuits. Is it necessary that the reduction computes a linear threshold function? The original result of [13] shows that this is *not* necessary if ϵ' is extremely small.
2. *Query complexity.* If F consists of functions computed by circuits of size s, then F' will need to consist of functions computed by circuits of a larger size s' — but how much larger? If the reduction makes q queries to functions from F, then plugging in size-s circuits for these functions yields a circuit of size $\geq q \cdot s$, and thus we must have $s' \geq q \cdot s$. Hence it is desirable to minimize q. Can we do better than $q \leq O\big((1/\epsilon^2)\log(1/\delta)\big)$ as in Theorem 4?
3. *Advice complexity.* Suppose F consists of functions computed by uniform algorithms running in time t (that is, a single algorithm computes a sequence of functions, one for each $n = 1, 2, 3, \ldots,$). Then can we let F' consist of functions computed by uniform algorithms running in some (slightly larger) time t'? (Here, the distributions D, M, R, U would need to be sequences of distributions, and a distinguisher would only be required to succeed for infinitely many n.) The proofs of [12, 15] do not yield this, because the reductions need a nonuniform advice string to provide some extra information about the nth distribution D. How many bits of advice are needed?

Before proceeding we draw attention to the fact that, as we just alluded to, the advice strings used by the reductions of [12, 15] depend on D *but do not depend on* R. Hence something a little stronger than Theorem 4 actually holds: Although the statement of Theorem 4 says we need to assume that for some R in which D is δ-dense, there is no function in F' that ϵ'-distinguishes R and U, we actually only need to assume that there is no function in F' that simultaneously ϵ'-distinguishes U from every R in which D is δ-dense (the quantifiers are swapped). We are interested in proving lower bounds on the complexity of this type of black-box reduction for the Dense Model Theorem, where the advice does not depend on R.

The *query complexity* was considered by Zhang [15], who showed that for a certain type of nonadaptive black-box reduction, $\Omega\big((1/\epsilon^2)\log(1/\delta)\big)$ queries are necessary when

$\epsilon' \geq \Omega(\epsilon\delta)$ and $\epsilon \geq \delta^{O(1)}$, matching the upper bound of $O\big((1/\epsilon^2)\log(1/\delta)\big)$ for this case. In this paper we consider the *advice complexity*. We show that for arbitrary black-box reductions, $\Omega\big(\sqrt{(1/\epsilon)\log(1/\delta)}\cdot\log|F|\big)$ advice bits are necessary when $\epsilon' \geq \Omega(\epsilon\delta)$ and $\epsilon \geq \delta^{O(1)}$, which comes close to matching the upper bound of $O\big((1/\epsilon^2)\log(1/\delta)\cdot\log|F|\big)$ for this case [15]. Our result also holds for much more general settings of the parameters (with some degradation in the lower bound). Proving lower bounds on the *computational complexity* remains open.

Let us formally state what we mean by a black-box reduction. Recall the standard notation $[k] = \{1, \ldots, k\}$.

▶ **Definition 5.** An $(n, \epsilon, \delta, \epsilon', k, \alpha)$-*reduction* (for the Dense Model Theorem) is a function

$$\mathrm{Dec} : \big(\{0,1\}^{2^n}\big)^k \times \{0,1\}^\alpha \to \{0,1\}^{2^n}$$

such that for all $f_1, \ldots, f_k \in \{0,1\}^{2^n}$ and all $D \in \mathcal{D}_n$, if for every $M \in \mathcal{D}_n$ that is δ-dense in the uniform distribution $U \in \mathcal{D}_n$ there exists an $i \in [k]$ such that f_i ϵ-distinguishes D and M, then there exists an advice string $a \in \{0,1\}^\alpha$ such that for every $R \in \mathcal{D}_n$ in which D is δ-dense, $\mathrm{Dec}(f_1, \ldots, f_k, a)$ ϵ'-distinguishes R and U.

The proofs of [12, 15] work by exhibiting such reductions. The functions $\{f_1, \ldots, f_k\}$ correspond to the class F (which, if we were considering uniform algorithms, would be the restrictions of all the algorithms in the class to a particular input length n). We now state our theorem.

▶ **Theorem 6** (Main). *If there exists an $(n, \epsilon, \delta, \epsilon', k, \alpha)$-reduction for the Dense Model Theorem, and if $w > 1$ is an integer such that $2^{w+2} \cdot \delta^{w/160} \leq \epsilon'$, then*

$$\alpha \;\geq\; \Big\lfloor \tfrac{1}{160w}\sqrt{(1/\epsilon)\log_2(1/\delta)}\Big\rfloor \cdot \log_2 k - \log_2 w - 1$$

provided $2^n \geq \frac{w\log_2 k}{\epsilon\delta^2(\epsilon')^2}$, $\epsilon \leq 1/64\log_2(1/\delta)$, and $k \geq 1/16\epsilon^4$.

For the case where $\epsilon' \geq \Omega(\epsilon\delta)$ and $\epsilon \geq \delta^{O(1)}$ (which is reasonable), the condition $2^{w+2} \cdot \delta^{w/160} \leq \epsilon'$ is met provided w is a sufficiently large constant and δ is less than a sufficiently small constant,[2] and thus we get a lower bound $\alpha \geq \Omega\big(\sqrt{(1/\epsilon)\log(1/\delta)}\cdot\log k\big)$. Note that the three conditions at the end of the statement of Theorem 6 are very generous.

Our proof of Theorem 6 is somewhat reminiscent of the proof of a lower bound due to Lu, Tsai, and Wu [10] on the advice complexity of black-box reductions for the Hardcore Lemma, but our proof diverges significantly. We now give a quick preview of some of our ingredients. We use the probabilistic method to find a class of functions $f_1, \ldots, f_k$ for which many advice strings are needed to “cover” all the distributions D that do not have dense models. The key technical ingredients in the analysis (which differ from the ingredients in [10] and which may be of independent interest) include (1) a combinatorial argument identifying when several distributions D cannot share the same advice string, and (2) an analysis of a majority of majorities applied to overlapping sets of p-biased bits, where the sets form an almost-disjoint family (see Figure 2). The latter analysis makes use of extremely tight lower bounds on the tail probabilities of the binomial distribution, which we also prove (but could not find in the literature).

In the full version we point out an analogy between our lower bound and list size lower bounds for list-decoding of error-correcting codes, and we summarize analogous previous work on lower bounds for hardness amplification and list-decoding. The rest of this paper is devoted to proving Theorem 6. In Section 2 we give some intuition for the proof, and then in Section 3 we give the formal proof.

[2] The statement of Theorem 6 requires $\delta < 2^{-160}$, but this constant can be drastically improved.

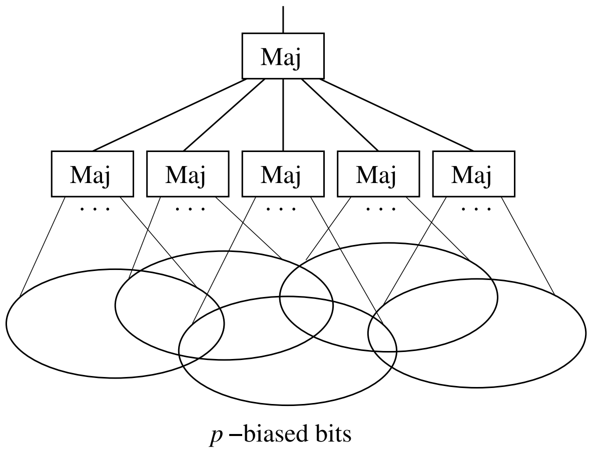


Figure 2 The majority of majorities

2 Intuition

According to Definition 5, for Dec to succeed as a reduction, it must be the case that for all $f_1, \dots, f_k \in \{0,1\}^{2^n}$ and all $D \in \mathcal{D}_n$, if D has no "dense model" then there is some advice string a such that $\mathrm{Dec}(f_1, \dots, f_k, a)$ "covers" D in a certain sense. To show that Dec needs many advice strings in order to succeed, we find functions $f_1, \dots, f_k \in \{0,1\}^{2^n}$ and a large family of distributions in $\mathcal{D}_n$ such that

(i) each distribution in the family has no dense model (with respect to $f_1, \dots, f_k$), and

(ii) each function $f \in \{0,1\}^{2^n}$ covers few of the distributions in the family.

So (i) implies that each distribution in the family needs to get covered, while (ii) implies that for each advice string a, $\mathrm{Dec}(f_1, \dots, f_k, a)$ does not cover very many of them. Since the family is large, many advice strings are needed.

First we describe a technique for achieving (i), then we describe a technique for achieving (ii), and then we show how to consolidate the techniques to achieve both properties simultaneously. When we say D has no "dense model" we mean that for every $M \in \mathcal{D}_n$ that is δ-dense in U there exists an $i \in [k]$ such that f_i ϵ-distinguishes D and M. When we say a function "covers" D we mean that it ϵ'-distinguishes R and U for every $R \in \mathcal{D}_n$ in which D is δ-dense. The only distributions D we need to consider are uniform distributions over subsets of $\{0,1\}^n$.

Given $f_1, \dots, f_k \in \{0,1\}^{2^n}$, what is an example of a distribution with no dense model? Suppose we pick any $I \subseteq [k]$ of size $1/4\epsilon$ and we let X_I be the set of all $x \in \{0,1\}^n$ such that $f_i(x) = 1$ for the majority of $i \in I$. Suppose we take D_I to be the uniform distribution over X_I. Then we have $\Pr_{x \sim D_I,\, i \sim I}[f_i(x) = 1] \geq 1/2 + 2\epsilon$ where $i \sim I$ means picking $i \in I$ uniformly at random. If X_I is roughly a $\delta/2$ fraction of $\{0,1\}^n$, then every distribution M that is δ-dense in U has at least half its mass outside of X_I, on strings x where $\Pr_{i \sim I}[f_i(x) = 1] \leq 1/2 - 2\epsilon$. It is possible to show that $\Pr_{x \sim M,\, i \sim I}[f_i(x) = 1] < \Pr_{x \sim D_I,\, i \sim I}[f_i(x) = 1] - \epsilon$ and thus there exists an $i \in I$ (depending on M) such that f_i ϵ-distinguishes D_I and M. So if $|X_I| \approx (\delta/2)2^n$ then D_I has no dense model. This is the technique we use for finding distributions without dense models.

Now, what is an example of a pair of distributions such that no function can cover both simultaneously? If we can show that every pair of distributions in the family is like this, then we will have achieved (ii). Because of an issue described below, we actually need to consider small collections of distributions rather than just pairs, but for now we consider pairs. Suppose D is uniform over some $X \subseteq \{0,1\}^n$ of size roughly $(\delta/2)2^n$, and similarly D' is uniform over some $X' \subseteq \{0,1\}^n$ of size roughly $(\delta/2)2^n$. If $X \cap X' = \emptyset$, then it can be shown that no function covers both D and D'.[3] Furthermore, if $|X \cap X'|$ is at most roughly $\epsilon' 2^n$ then this property still holds.

To consolidate the two techniques, we find a large family of sets $I \subseteq [k]$ each of size $1/4\epsilon$, where

(A) $|X_I| \approx (\delta/2)2^n$ for each I in the family, and
(B) the pairwise intersections of the X_I's (for I in the family) all have size at most roughly $\epsilon' 2^n$.

This would imply that the corresponding distributions D_I (for I in the family) have no dense models, and no function would cover more than one of them, so (i) and (ii) would be achieved.

We choose the functions $f_1, \ldots, f_k \in \{0,1\}^{2^n}$ randomly in some way, and we argue that for an appropriate family of sets I, properties (A) and (B) both hold with high probability. Property (A) suggests that we should choose p so that the probability a majority of $1/4\epsilon$ independent coins each with expectation p come up 1 is exactly $\delta/2$. Then we can set $f_i(x) = 1$ with probability p independently for each $i \in [k]$ and each $x \in \{0,1\}^n$, so for each I of size $1/4\epsilon$, $\Pr[x \in X_I] = \delta/2$. Then by concentration, $|X_I| \approx (\delta/2)2^n$ with high probability over $f_1, \ldots, f_k$.

If we choose $f_1, \ldots, f_k$ randomly in this way, how big will $|X_I \cap X_{I'}|$ be, for I and I' in the family? By concentration, we would have that with high probability over $f_1, \ldots, f_k$, $|X_I \cap X_{I'}|$ is roughly 2^n times $\Pr[x \in X_I \cap X_{I'}]$ (which is the same for all $x \in \{0,1\}^n$), so we would like the latter probability to be $\leq \epsilon'$. So what is the probability that the conjunction of two majorities of p-biased bits is 1? The best case is if $I \cap I' = \emptyset$, in which case the probability is exactly $(\delta/2)^2$. There are two problems with this.

(1) We cannot get a very large family of sets I if we require them to be pairwise disjoint.
(2) This requires $\epsilon' \geq (\delta/2)^2$. In a typical setting where $\epsilon' \geq \Omega(\epsilon\delta)$, this would require $\epsilon > \delta$, which is an odd and somewhat severe restriction.

To solve problem (1), we use the natural idea to allow the sets I to be pairwise almost-disjoint, rather than disjoint (which allows us to get a much larger family). So if $|I \cap I'|$ is at most some value b, how small does b have to be to ensure that the probability both majorities are 1 is not much more than $(\delta/2)^2$? We analyze this using the following trick: If both majorities are 1, then the fraction of coins that are 1 among $I \cup I'$ is at least q, where $q = 1/2 - 2\epsilon b = \frac{1/4\epsilon - b}{1/2\epsilon} \leq \frac{|I|/2 + |I'|/2 - b}{|I \cup I'|}$. Using an extremely tight characterization of the tail probabilities of the binomial distribution (which we prove using known techniques but which we could not find in the literature), we can show that $p \approx 1/2 - \sqrt{\epsilon \log(1/\delta)}$ and the probability of getting $\geq q$ fraction of 1's among the $|I \cup I'|$ coins is not much more than $(\delta/2)^2$ provided q is at least a constant factor closer to $1/2$ than p is, say $q \approx 1/2 - \sqrt{\epsilon \log(1/\delta)}/4$. Thus it suffices to have $b \approx \sqrt{\epsilon \log(1/\delta)}/8\epsilon \geq \Omega\big(\sqrt{(1/\epsilon)\log(1/\delta)}\big)$. Since the family of sets I needs to be in the universe $[k]$, there exists such a family of roughly k^b many sets with pairwise intersections bounded in size by b. Since each function can cover D_I for only one I

[3] Actually, there is an issue having to do with the absolute value signs in the definition of distinguishing; this is dealt with in the formal proof.

in the family, roughly k^b advice strings are needed, which gives an advice lower bound of roughly $\log(k^b) \geq \Omega\big(\sqrt{(1/\epsilon)\log(1/\delta)} \cdot \log k\big)$.

Problem (2) is solved in the formal proof by considering small collections of sets from the family, rather than pairs. The parameter w in Theorem 6 is used to determine how big these collections should be. Then instead of considering the conjunction of two majorities, we need to consider the majority of several majorities, which explains where Figure 2 comes from.

3 Formal Proof

In Section 3.1, Section 3.2, and Section 3.3 we give preliminary lemmas, definitions, and notation. Then in Section 3.4 we give the proof of Theorem 6.

3.1 Binomial Distribution Tail

We let $\mathrm{Tail}(m, p, q)$ denote the probability that when m independent coins are flipped each with probability p of heads, at least a q fraction of the coins are heads (in other words, the probability the (m, p) binomial distribution is at least qm). For our proof of Theorem 6 we need extremely tight upper and lower bounds on the value of $\mathrm{Tail}(m, p, q)$. Such bounds can be given in terms of the fundamental quantity $\mathrm{RE}(q\|p) = q\log_2(\frac{q}{p}) + (1-q)\log_2(\frac{1-q}{1-p})$ which is known by a variety of names such as relative entropy, information divergence, and Kullback-Leibler distance.

We need the following fact, which can be seen using derivatives.

▶ Fact 7. For all $1/4 \leq p \leq q \leq 3/4$, we have $2(q-p)^2 \leq \mathrm{RE}(q\|p) \leq 4(q-p)^2$.

We also need the following standard and well-known form of the Chernoff-Hoeffding bound.

▶ **Lemma 8.** *For all $m \geq 1$ and all $0 \leq p \leq q \leq 1$, we have* $\mathrm{Tail}(m, p, q) \leq 2^{-\mathrm{RE}(q\|p)m}$.

The following lemma (see the full version for the proof) shows that Lemma 8 is very tight.

▶ **Lemma 9.** *For all $m \geq 1$ and all $1/4 \leq p \leq q \leq 1$, we have* $\mathrm{Tail}(m, p, q) \geq \frac{1}{48\sqrt{m}} \cdot 2^{-\mathrm{RE}(q\|p)m}$.

Although Lemma 9 is very simple and general, for our purpose we can only use it for a limited range of parameters, namely when $\epsilon \gg \delta$. This is because $\mathrm{RE}(q\|p)$ could be so close to 0 that $\frac{1}{48\sqrt{m}}$ completely swamps $2^{-\mathrm{RE}(q\|p)m}$, in which case Lemma 9 is not very tight. To handle the full range of ϵ and δ, we use the following stronger lower bound for the case $q = 1/2$. We prove this lemma in the full version.

▶ **Lemma 10.** *For all $m \geq 9$ and all $1/4 \leq p < 1/2$, we have*

$$\mathrm{Tail}(m, p, 1/2) \;\geq\; \min\Big(\tfrac{1}{256}, \tfrac{1}{128\sqrt{m}(1/2-p)}\Big) \cdot 2^{-\mathrm{RE}(1/2\|p)m}.$$

3.2 Combinatorial Designs

For our proof of Theorem 6 we need the existence of large families of almost-disjoint subsets of a finite set. Such combinatorial designs have numerous applications in theoretical computer science.

▶ **Definition 11.** An (ℓ, k, s, b)-*design* is a family of sets $I_1, \ldots, I_\ell \subseteq [k]$ all of size s such that $|I_j \cap I_{j'}| \leq b$ for every $j \neq j'$.

▶ **Lemma 12.** *For every k, s, b there exists an (ℓ, k, s, b)-design with $\ell \geq k^{b/8}$, provided $k \geq 16s^4$.*

There is nothing very novel about this lemma, and this precise version follows from a result in [4], but we provide a simple, self-contained proof in the full version. The proof uses the probabilistic method with a simple concentration bound for the hypergeometric distribution.

3.3 Notational Preliminaries

The parameters $n, \epsilon, \delta, \epsilon', k$, and w are fixed as in the statement of Theorem 6, and we always use D, M, R, U (possibly subscripted) to denote distributions in $\mathcal{D}_n$, in their roles as in Definition 5.

We let Maj denote the majority function on bit strings, and for even length strings we break ties by returning 1. We let And denote the and function on bit strings. We let Maj^t denote the function that takes t bit strings and returns their majorities as a length-t bit string. We use $\circ$ for function composition.

We also adhere to the following notational conventions. We use x for elements of $\{0,1\}^n$ and X for subsets of $\{0,1\}^n$. We use f for elements of $\{0,1\}^{2^n}$ (identified with functions from $\{0,1\}^n$ to $\{0,1\}$) and F for subsets of $\{0,1\}^{2^n}$. We use $[k]$ to index functions f, and we use i for elements of $[k]$ and I for subsets of $[k]$. We use $[\ell]$ to index subsets I (as in Definition 11), and we use j for elements of $[\ell]$ and J for subsets of $[\ell]$. We generally use s for the size of I, and t for the size of J.

The following notation is with respect to fixed $f_1, \ldots, f_k \in \{0,1\}^{2^n}$. Given $I \subseteq [k]$ we define

- f_I is the function that takes $x \in \{0,1\}^n$ and returns the length-$|I|$ bit string $(f_i(x))_{i \in I}$;
- X_I is the set of $x \in \{0,1\}^n$ on which $\mathrm{Maj} \circ f_I$ returns 1;
- D_I is the uniform distribution over X_I (and if $X_I = \emptyset$ then D_I is undefined).

The following notation is with respect to fixed $f_1, \ldots, f_k \in \{0,1\}^{2^n}$ and fixed $I_1, \ldots, I_\ell \subseteq [k]$. Given $J \subseteq [\ell]$ we define

- f_{I_J} is the function that takes $x \in \{0,1\}^n$ and returns the $|J|$-tuple $(f_{I_j}(x))_{j \in J}$;
- X_{I_J} is the set of $x \in \{0,1\}^n$ on which $\mathrm{Maj} \circ \mathrm{Maj}^{|J|} \circ f_{I_J}$ returns 1.

We use $\sim$ to denote sampling from a distribution (for example $x \sim D$), and we use the convention that sampling from a set (for example $i \sim I$) means sampling from the uniform distribution over that set.

3.4 Proof of Theorem 6

Consider an arbitrary function $\mathrm{Dec} : \left(\{0,1\}^{2^n}\right)^k \times \{0,1\}^\alpha \to \{0,1\}^{2^n}$. Supposing that $\alpha < \left\lfloor \frac{1}{160w} \sqrt{(1/\epsilon) \log_2(1/\delta)} \right\rfloor \cdot \log_2 k - \log_2 w - 1$, we show that Dec is not an $(n, \epsilon, \delta, \epsilon', k, \alpha)$-reduction. We first introduce some terminology to make things concise. Given $f_1, \ldots, f_k \in \{0,1\}^{2^n}$, a *dense model* for $D \in \mathcal{D}_n$ is an $M \in \mathcal{D}_n$ that is δ-dense in the uniform distribution $U \in \mathcal{D}_n$ and is such that for all $i \in [k]$, f_i does not ϵ-distinguish D and M. We say a function $f \in \{0,1\}^{2^n}$ *covers* $D \in \mathcal{D}_n$ if for every $R \in \mathcal{D}_n$ in which D is δ-dense, f ϵ'-distinguishes R and U.

Thus to show that Dec is not an $(n, \epsilon, \delta, \epsilon', k, \alpha)$-reduction, we need to find $f_1, \ldots, f_k \in \{0,1\}^{2^n}$ such that some D has no dense model but is not covered by $\mathrm{Dec}(f_1, \ldots, f_k, a)$ for any advice string $a \in \{0,1\}^\alpha$.

3.4.1 Distributions Without Dense Models

The following claim is our tool for finding distributions that have no dense models. We prove this claim in the full version.

▶ Claim 13. For every $f_1, \ldots, f_k \in \{0,1\}^{2^n}$ and every $I \subseteq [k]$ of size $0 < s \leq 1/4\epsilon$ (for some s), if $0 < |X_I| \leq (2\delta/3)2^n$ then D_I has no dense model.

3.4.2 Distributions That Cannot Be Covered

We say a function $f \in \{0,1\}^{2^n}$ *positively covers* $D \in \mathcal{D}_n$ if for every $R \in \mathcal{D}_n$ in which D is δ-dense, $\mathrm{E}_R[f] - \mathrm{E}_U[f] > \epsilon'$ (note the absence of absolute value signs). Observe that if $f \in \{0,1\}^{2^n}$ covers D then either f or its complement positively covers D. This is because if there existed $R_1, R_2 \in \mathcal{D}_n$ in which D is δ-dense and such that $\mathrm{E}_{R_1}[f] < \mathrm{E}_U[f] < \mathrm{E}_{R_2}[f]$, then some convex combination R_3 of R_1 and R_2 would have $\mathrm{E}_{R_3}[f] = \mathrm{E}_U[f]$. However, D would be δ-dense in R_3 since the set of R in which D is δ-dense is convex, so f would not cover D.

▶ Claim 14. For every $f_1, \ldots, f_k \in \{0,1\}^{2^n}$, every $I_1, \ldots, I_\ell \subseteq [k]$ (for some ℓ), and every $J \subseteq [\ell]$ of size $t > 1$ (for some t), if $|X_{I_J}| \leq (\epsilon'/2)2^n$ and $|X_{I_j}| \geq (\delta/2 - \epsilon'/4)2^n$ for all $j \in J$ then there is no function that simultaneously positively covers D_{I_j} for all $j \in J$.

Proof. Assume that $|X_{I_J}| \leq (\epsilon'/2)2^n$ and $|X_{I_j}| \geq (\delta/2 - \epsilon'/4)2^n$ for all $j \in J$. Consider an arbitrary $f \in \{0,1\}^{2^n}$ and let X be the set of $x \in \{0,1\}^n$ such that $f(x) = 1$. For $\tau \in \{0, 1, \ldots, t\}$ let $X^{(\tau)}$ be the set of $x \in \{0,1\}^n$ such that there are exactly τ values of $j \in J$ for which $x \in X_{I_j}$ (in other words, $(\mathrm{Maj}^t \circ f_{I_J})(x)$ has Hamming weight τ). Note that $X_{I_J} = \bigcup_{\tau=t'}^{t} X^{(\tau)}$ where $t' = \lceil t/2 \rceil$. Let $\pi = \min_{j \in J} \left[\mathrm{E}_{D_{I_j}}[f] \right]$. Then for every $j \in J$ we have $|X \cap X_{I_j}| \geq \pi \cdot |X_{I_j}| \geq \pi \cdot (\delta/2 - \epsilon'/4)2^n$. We have

$$\begin{aligned}
(t/2) \cdot \big(|X| + |X_{I_J}|\big) &\geq (t/2) \cdot |X \cap \overline{X_{I_J}}| + t \cdot |X \cap X_{I_J}| \\
&\geq \textstyle\sum_{\tau=0}^{t} \tau \cdot |X \cap X^{(\tau)}| \\
&= \textstyle\sum_{j \in J} |X \cap X_{I_j}| \\
&\geq t \cdot \pi \cdot (\delta/2 - \epsilon'/4)2^n
\end{aligned}$$

which implies that

$$|X| \;\geq\; \pi \cdot (\delta - \epsilon'/2)2^n - |X_{I_J}| \;\geq\; \pi\delta 2^n - \epsilon' 2^n \;=\; (\pi - \epsilon'/\delta) \cdot \delta 2^n$$

since $\pi \leq 1$ and $|X_{I_J}| \leq (\epsilon'/2)2^n$. We might have $\pi - \epsilon'/\delta < 0$, but this is not problematic. Let M be a distribution δ-dense in U that maximizes $\mathrm{E}_M[f]$, and observe that

$$\mathrm{E}_M[f] \;=\; \min\big(|X|/\delta 2^n, 1\big) \;\geq\; \pi - \epsilon'/\delta.$$

We have $U = \delta M + (1-\delta)\widehat{M}$ for some $\widehat{M} \in \mathcal{D}_n$. Let $j \in J$ be such that $\mathrm{E}_{D_{I_j}}[f] = \pi$, and define the distribution $R = \delta D_{I_j} + (1-\delta)\widehat{M}$ so that D_{I_j} is δ-dense in R. Then we have

$$\mathrm{E}_R[f] \;=\; \delta\pi + (1-\delta)\,\mathrm{E}_{\widehat{M}}[f]$$

and

$$\mathrm{E}_U[f] \;=\; \delta\,\mathrm{E}_M[f] + (1-\delta)\,\mathrm{E}_{\widehat{M}}[f] \;\geq\; \delta\pi - \epsilon' + (1-\delta)\,\mathrm{E}_{\widehat{M}}[f] \;=\; \mathrm{E}_R[f] - \epsilon'$$

so f does not positively cover D_{I_j}. This finishes the proof of Claim 14. ◀

3.4.3 Setting the Parameters

Define $s = \lfloor 1/4\epsilon \rfloor$ and $t = w$ and $b = \left\lfloor \frac{1}{20t}\sqrt{(1/\epsilon)\log_2(1/\delta)} \right\rfloor$. By Lemma 12 there exists an (ℓ, k, s, b)-design $I_1, \ldots, I_\ell$ with $\ell = \lceil k^{b/8} \rceil$ (note that we do have $k \geq 16s^4$). Define p to be such that $\mathrm{Tail}(s, p, 1/2) = \delta/2$. We prove the following claim in the full version, using Lemma 10.

▶ Claim 15. $\frac{1}{2}\sqrt{\epsilon \log_2(1/\delta)} \;\leq\; 1/2 - p \;\leq\; 2\sqrt{\epsilon \log_2(1/\delta)} \;\leq\; 1/4.$

3.4.4 The Majority of Majorities

We choose $f_1, \ldots, f_k$ randomly by setting $f_i(x) = 1$ with probability p independently for each $i \in [k]$ and each $x \in \{0,1\}^n$.

▶ Claim 16. For every $J \subseteq [\ell]$ of size t and every $x \in \{0,1\}^n$, we have $\Pr_{f_1,\ldots,f_k}[x \in X_{I_J}] \leq \epsilon'/4$.

Proof. Define $t' = \lceil t/2 \rceil$. Note that if $(\mathrm{Maj} \circ \mathrm{Maj}^t \circ f_{I_J})(x) = 1$ then there exists a subset $J' \subseteq J$ of size t' such that $(\mathrm{And} \circ \mathrm{Maj}^{t'} \circ f_{I_{J'}})(x) = 1$. Thus we have

$$\begin{aligned} &\Pr_{f_1,\ldots,f_k}\left[(\mathrm{Maj} \circ \mathrm{Maj}^t \circ f_{I_J})(x) = 1\right] \\ \leq\; &2^t \cdot \max_{J' \subseteq J \,:\, |J'| = t'} \Pr_{f_1,\ldots,f_k}\left[(\mathrm{And} \circ \mathrm{Maj}^{t'} \circ f_{I_{J'}})(x) = 1\right]. \end{aligned}$$

Consider an arbitrary $J' \subseteq J$ of size t'. Define $m = \left|\bigcup_{j \in J'} I_j\right|$ and notice that since $I_1, \ldots, I_\ell$ is an (ℓ, k, s, b)-design, by inclusion-exclusion we have

$$t's - \binom{t'}{2}b \;\leq\; m \;\leq\; t's. \tag{1}$$

Define $s' = \lceil s/2 \rceil$ and $q = 1/2 - t'b/2s$. If $(\mathrm{And} \circ \mathrm{Maj}^{t'} \circ f_{I_{J'}})(x) = 1$ then for each $j \in J'$ we have $\sum_{i \in I_j} f_i(x) \geq s'$ and so by inclusion-exclusion we have

$$\textstyle\sum_{i \in \bigcup_{j \in J'} I_j} f_i(x) \;\geq\; \left(\sum_{j \in J'} \sum_{i \in I_j} f_i(x)\right) - \binom{t'}{2}b \;\geq\; t's' - \binom{t'}{2}b \;\geq\; qt's \;\geq\; qm.$$

It follows that

$$\begin{aligned} \Pr_{f_1,\ldots,f_k}\left[(\mathrm{And} \circ \mathrm{Maj}^{t'} \circ f_{I_{J'}})(x) = 1\right] \;&\leq\; \Pr_{f_1,\ldots,f_k}\left[\textstyle\sum_{i \in \bigcup_{j \in J'} I_j} f_i(x) \geq qm\right] \\ &=\; \mathrm{Tail}(m, p, q) \\ &\leq\; 2^{-\mathrm{RE}(q\|p)m} \\ &=\; \left(2^{-\mathrm{RE}(1/2\|p)s}\right)^{(m/s)\cdot(\mathrm{RE}(q\|p)/\mathrm{RE}(1/2\|p))} \\ &\leq\; \left(\delta^{1/10}\right)^{(m/s)\cdot(\mathrm{RE}(q\|p)/\mathrm{RE}(1/2\|p))} \end{aligned}$$

where the third line follows by Lemma 8 and the fifth line follows by nonnegativity of RE and

$$2^{-\mathrm{RE}(1/2\|p)s} \;\leq\; 2^{-2(1/2-p)^2 s} \;\leq\; \delta^{\epsilon s/2} \;\leq\; \delta^{1/10}$$

which holds by Fact 7, Claim 15, and $\epsilon \leq 1/20$. We have

$$m/s \;\geq\; t' - (t')^2 b/2s \;\geq\; t'/2 \;\geq\; t/4 \tag{2}$$

by (1) and $b \leq s/t'$ (which can be shown using the final inequality in Claim 15). We also have $t'b/2s \leq \frac{1}{8}\sqrt{\epsilon \log_2(1/\delta)}$ and thus $q - p \geq \frac{3}{4}(1/2 - p)$ by Claim 15. Hence by Fact 7 we have

$$\mathrm{RE}(q\|p)/\,\mathrm{RE}(1/2\|p) \;\geq\; \tfrac{(q-p)^2}{2(1/2-p)^2} \;\geq\; \tfrac{(\frac{3}{4}(1/2-p))^2}{2(1/2-p)^2} \;\geq\; 1/4. \tag{3}$$

Using (2) and (3) we get

$$\Pr_{f_1,\ldots,f_k}\left[(\mathrm{And} \circ \mathrm{Maj}^{t'} \circ f_{I_{J'}})(x) = 1\right] \;\leq\; \left(\delta^{1/10}\right)^{(t/4)\cdot(1/4)} \;=\; \delta^{t/160}.$$

We conclude that $\Pr_{f_1,\ldots,f_k}[x \in X_{I_J}] \;\leq\; 2^t \cdot \delta^{t/160} \;\leq\; \epsilon'/4$. This finishes the proof of Claim 16. ◀

3.4.5 Putting It All Together

For every $j \in [\ell]$ and every $x \in \{0,1\}^n$, we have $\Pr_{f_1,\ldots,f_k}[x \in X_{I_j}] = \mathrm{Tail}(s, p, 1/2) = \delta/2$. Standard relative-error forms of the Chernoff bound give

$$\begin{aligned}
&\Pr_{f_1,\ldots,f_k}\left[|X_{I_j}| < (\delta/2 - \epsilon'/4)2^n\right] &&\leq\; e^{-2^n(\epsilon')^2/16\delta}\\
&\Pr_{f_1,\ldots,f_k}\left[|X_{I_j}| > (2\delta/3)2^n\right] &&\leq\; e^{-2^n\delta/54}\\
&\Pr_{f_1,\ldots,f_k}\left[|X_{I_J}| > (\epsilon'/2)2^n\right] &&\leq\; e^{-2^n\epsilon'/12}
\end{aligned}$$

where the latter holds for each $J \subseteq [\ell]$ of size t, using Claim 16. Thus by a union bound we have

$$\begin{aligned}
&\Pr_{f_1,\ldots,f_k}\begin{bmatrix}(\delta/2 - \epsilon'/4)2^n \leq |X_{I_j}| \leq (2\delta/3)2^n \text{ for all } j \in [\ell] \text{ and}\\ |X_{I_J}| \leq (\epsilon'/2)2^n \text{ for all } J \subseteq [\ell] \text{ of size } t\end{bmatrix}\\
&\geq 1 - \ell \cdot e^{-2^n(\epsilon')^2/16\delta} - \ell \cdot e^{-2^n\delta/54} - \tbinom{\ell}{t} \cdot e^{-2^n\epsilon'/12}\\
&> 0
\end{aligned}$$

since $2^n \geq \frac{t \log_2 k}{\epsilon\delta^2(\epsilon')^2}$. Fix a choice of $f_1, \ldots, f_k$ such that the above event occurs.

For every $J^* \subseteq [\ell]$ of size $2t - 1$, there is no $a \in \{0,1\}^\alpha$ such that $\mathrm{Dec}(f_1, \ldots, f_k, a)$ simultaneously covers D_{I_j} for all $j \in J^*$, because otherwise for some $J \subseteq J^*$ of size t, either $\mathrm{Dec}(f_1, \ldots, f_k, a)$ or its complement would simultaneously positively cover D_{I_j} for all $j \in J$, which would contradict Claim 14.

Therefore for each $a \in \{0,1\}^\alpha$, the number of $j \in [\ell]$ such that D_{I_j} is covered by $\mathrm{Dec}(f_1, \ldots, f_k, a)$ is at most $2t - 2$. This implies that the number of $j \in [\ell]$ for which there exists an $a \in \{0,1\}^\alpha$ such that $\mathrm{Dec}(f_1, \ldots, f_k, a)$ covers D_{I_j} is at most $2^\alpha \cdot (2t-2) < k^{b/8} \leq \ell$ since $\alpha \leq (b/8)\log_2 k - \log_2 t - 1$. Thus there exists a $j \in [\ell]$ such that D_{I_j} is not covered by $\mathrm{Dec}(f_1, \ldots, f_k, a)$ for any $a \in \{0,1\}^\alpha$. By Claim 13, D_{I_j} has no dense model, so Dec is not an $(n, \epsilon, \delta, \epsilon', k, \alpha)$-reduction. This finishes the proof of Theorem 6.

Acknowledgments I thank Anand Bhaskar, Siu Man Chan, and Siu On Chan for helpful discussions, and anonymous reviewers for useful comments.

References

1 Boaz Barak, Moritz Hardt, and Satyen Kale. The uniform hardcore lemma via approximate Bregman projections. In *Proceedings of the 20th ACM-SIAM Symposium on Discrete Algorithms*, pages 1193–1200, 2009.

2 Boaz Barak, Ronen Shaltiel, and Avi Wigderson. Computational analogues of entropy. In *Proceedings of the 7th International Workshop on Randomization and Computation*, pages 200–215, 2003.

3 Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *Proceedings of the 49th IEEE Symposium on Foundations of Computer Science*, pages 293–302, 2008.

4 Paul Erdős, Péter Frankl, and Zoltán Füredi. Families of finite sets in which no set is covered by the union of r others. *Israel Journal of Mathematics*, 51(1-2):79–89, 1985.

5 Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the 43rd ACM Symposium on Theory of Computing*, pages 99–108, 2011.

6 Timothy Gowers. Decompositions, approximate structure, transference, and the Hahn–Banach Theorem. *Bulletin of the London Mathematical Society*, 42(4):573–606, 2010.

7 Timothy Gowers and Julia Wolf. Linear forms and higher-degree uniformity for functions on $\mathbb{F}_p^n$. *Geometric and Functional Analysis*, 21(1):36–69, 2011.

8 Timothy Gowers and Julia Wolf. Linear forms and quadratic uniformity for functions on $\mathbb{F}_p^n$. *Mathematika*, 57(2):215–237, 2012.

9 Ben Green and Terence Tao. The primes contain arbitrarily long arithmetic progressions. *Annals of Mathematics*, 167(2):481–547, 2008.

10 Chi-Jen Lu, Shi-Chun Tsai, and Hsin-Lung Wu. Complexity of hard-core set proofs. *Computational Complexity*, 20(1):145–171, 2011.

11 Ilya Mironov, Omkant Pandey, Omer Reingold, and Salil Vadhan. Computational differential privacy. In *Proceedings of the 29th International Cryptology Conference*, pages 126–142, 2009.

12 Omer Reingold, Luca Trevisan, Madhur Tulsiani, and Salil Vadhan. Dense subsets of pseudorandom sets. In *Proceedings of the 49th IEEE Symposium on Foundations of Computer Science*, pages 76–85, 2008.

13 Terence Tao and Tamar Ziegler. The primes contain arbitrarily long polynomial progressions. *Acta Mathematica*, 201:213–305, 2008.

14 Luca Trevisan, Madhur Tulsiani, and Salil Vadhan. Regularity, boosting, and efficiently simulating every high-entropy distribution. In *Proceedings of the 24th IEEE Conference on Computational Complexity*, pages 126–136, 2009.

15 Jiapeng Zhang. On the query complexity for showing dense model. Technical Report TR11-038, Electronic Colloquium on Computational Complexity, 2011.

Index of Authors