Thresholds for Matchings in Random Bipartite Graphs with Applications to Hashing-Based Data Structures

Michael Rink

Dissertation zur Erlangung des akademischen Grades
Doctor rerum naturalium (Dr. rer. nat.)
vorgelegt der Fakultät für Informatik und Automatisierung
der Technischen Universität Ilmenau
am 13.01.2014, verteidigt am 16.09.2014

1. Gutachter: Univ.-Prof. Dr. Martin Dietzfelbinger (wiss. Betreuer), TU Ilmenau
2. Gutachter: Univ.-Prof. Dr. Konstantinos Panagiotou, LMU München
3. Gutachter: Prof. Rasmus Pagh, PhD., IT University of Copenhagen

urn:nbn:de:gbv:ilm1-2014000557

To Anja.

Preface

This thesis presents some results on matchings in random bipartite graphs which are at the core of several hashing-based data structures that were proposed in recent years. The results were obtained during my work on *Modern Hashing*, a project founded by a research grant from the DFG. It is my aim to put them into a broader context, show connections between them, as well as emphasize the simplicity and power of the covered hashing-based data structures.

Summary

We study randomized algorithms that take as input a set S of n keys from some large universe U or a set of n key-value pairs, associating each key from S with a specific value, and build a data structure that solves one of the following tasks. On calling the operation `lookup` for a key $x \in U$:

- ▷ Decide set membership with respect to S (membership tester).
- ▷ If $x \in S$, then return the value associated with x. If $x \in U - S$, then return either some specific value $\perp$ interpreted as "$x \notin S$" (dictionary), or some arbitrary value (retrieval data structure).
- ▷ If $x \in S$, then return a natural number associated with x, from a range with size close to n, where for all elements of S the numbers are pairwise distinct. The numbers for elements $x \in U - S$ are arbitrary (perfect hash function).

The data structures that we cover are variations of *cuckoo hashing* and *Bloomier filter* which have the same simple structure. They consist of a table with m cells, each capable of holding entries of size r bits, as well as a constant number of hash functions, which are used to map the elements from U to a constant number of table cells. Assuming *fully random hash functions*, we will discuss how the data structures can be constructed in time linear in n, and what load n/m or space consumption $m \cdot r$ can be achieved in trade-off with the time for `lookup`.
This leads to the question whether a random bipartite graph with n nodes (keys) on the left, m nodes (cells) on the right, and edges determined by the hash functions, asymptotically almost surely has a *matching* of a certain type, and furthermore, how such a matching can be calculated efficiently.
The focus of this thesis concerns:

- ▷ optimizing the lookup time for dictionary and membership tester (based on cuckoo hashing) with respect to:
  - (i) a bound on the average number of cell probes,
  - (ii) the expected number of page accesses in a setting where the table (memory) is subdivided into pages of the same size.
- ▷ minimizing the space usage of the retrieval data structure and perfect hash function (based on Bloomier filter), when for each key the number of cell probes for `lookup` is upper bounded by a constant.
- ▷ an efficient simulation of fully random hash functions as needed by the data structures.

Zusammenfassung

Wir betrachten randomisierte Algorithmen, die bei Eingabe einer Menge S von n Schlüsseln aus einem großen Universum U, oder einer Menge von n Schlüssel-Wert-Paaren bei der jedem Schlüssel aus S ein spezifischer Wert zugeordnet ist, eine Datenstruktur für eine der folgenden Aufgaben bauen. Bei Aufruf der Operation `lookup` für einen Schlüssel $x \in U$:

- ▷ Entscheide, ob x Element der Menge S ist (Mengenzugehörigkeitstester).
- ▷ Falls $x \in S$, gib den Wert zurück welcher x zugeordnet ist. Ansonsten gib den Wert $\perp$ mit der Interpretation „$x \notin S$“ zurück (Wörterbuch) oder gib einen beliebigen Wert zurück (Retrieval-Datenstruktur).
- ▷ Falls $x \in S$, gib eine von x abhängige natürliche Zahl zurück, wobei für alle $x \in S$ diese Zahlen paarweise verschieden sind und ihr Wertebereich in etwa die Größe n hat. Für Elemente $x \in U - S$ dürfen beliebige Zahlen zurückgegeben werden (perfekte Hashfunktion).

Die Datenstrukturen die wir behandeln sind Varianten von *Cuckoo-Hashing* und des *Bloomier-Filters*, welchen die gleiche einfache Struktur zu Grunde liegt. Sie bestehen aus einer Tabelle mit m Zellen der Breite r Bits sowie einer konstanten Anzahl an Hashfunktionen, welche benutzt werden, um die Elemente aus U auf eine konstante Anzahl an Tabellenzellen abzubilden. Unter der Annahme *voll zufälliger Hashfunktionen* beschäftigen wir uns damit, wie diese Datenstrukturen in Linearzeit (bezüglich n) konstruiert werden können und welche Ladung n/m bzw. welcher Platzverbrauch $m \cdot r$ in Abhängigkeit vom Zeitbedarf für `lookup` erreicht werden kann. Dies führt zu der Frage, ob ein bipartiter Zufallsgraph mit n Knoten (Schlüsseln) links und m Knoten (Zellen) rechts und Kanten bestimmt mittels der Hashfunktionen ein *Matching* eines bestimmten Typs hat und darüber hinaus, wie solch ein Matching effizient berechnet werden kann.

Schwerpunkte der Dissertation sind:

- ▷ die Optimierung der Lookup-Zeit für Wörterbuch und Mengenzugehörigkeitstester (basierend auf Cuckoo-Hashing) bezüglich:
  - (i) einer Beschränkung der durchschnittlichen Anzahl an Zellenzugriffen,
  - (ii) der erwarteten Anzahl an Seitenzugriffen unter der Annahme, dass die Tabelle (Speicher) in Speicherseiten gleicher Größe unterteilt ist.
- ▷ die Minimierung des Platzbedarfs von Retrieval-Datenstruktur und perfekter Hashfunktion (basierend auf dem Bloomier-Filter), wobei für jeden Schlüssel die Anzahl der Zellenzugriffe nach oben durch eine Konstante beschränkt ist.
- ▷ eine effiziente Simulation von voll zufälligen Hashfunktionen, welche von allen behandelten Datenstrukturen vorausgesetzt werden.

Acknowledgment

Personal Undoubtedly, there are a lot of people who have helped and supported me in various ways in the process of writing this dissertation; first and foremost my adviser, Martin Dietzfelbinger, who always found the time for answering my questions and willingly discussed all ideas brought forward — the good, the bad, and the flawed. I want to thank him for his endless patience and all of the guidance that he has given me.

I am also much obliged to my former colleagues: Martin Aumüller and Sascha Grau, who helped me numerous times and always lended an ear to me; Ulf Schellbach, my first office mate, and Christopher Mattern, who followed Ulf, for creating an amicable atmosphere and for showing me what attention to detail really means; Petra Schüller and Jana Kopp, who aided me with the practical problems of everyday life; Martin Huschenbett, Roy Mennicke, and Raed Jaberi for their support and encouragement; Michael Brinkmeier, who was the first to arouse my interest in theoretical computer science; Dietrich Kuske and Manfred Kunde for their advice.

Furthermore, I would like to express my appreciation to the external reviewers Konstantinos Panagiotou and Rasmus Pagh for willingly spending their valuable time in assessing the thesis submission.

I am grateful to all the people whose work and research directly contributed to or influenced this thesis. Especially, I would like to mention authors, not named above, of the papers I was involved during my study: Michael Mitzenmacher, Hendrik Peilke, Andreas Goerdt, and Andrea Montanari.

Finally, I want to thank my parents Gabriele and Joachim as well as my brother Thomas for always believing in me.

And last but surely not least, I am deeply indebted to my partner Anja and my little daughter Tamara who readily accepted my frequent absence from their lives. Anja's unremittingly support gave me the time for writing and she constantly encouraged me not to lose sight of the goal; without her help, this dissertation would not have been possible.

Institutional This work was supported by DFG, grants DI 412/10-1 and DI 412/10-2.

Previously Published Work and Contributions by Others

The main results presented in this thesis previously appeared in several peer-reviewed papers [DR09, ADR09, DGM+10, DMR11a, DR12a, Rin13]. Most of them are accessible online on `http://arXiv.org` in an extended or full version [DGM+09, DMR11b, DR12b, Rin12]. In order to appropriately weight my contribution to each of the multi-author papers so far as it is relevant for this thesis, the following list acknowledges *contributions* of the other authors in which *I was not or only partially involved.*

Section 3.4 The generalized selfless algorithm from [DGM+10] was found in joint work with Martin Dietzfelbinger.

Section 3.5 All results from [DR12a] are joint work.

Section 3.6 The idea for the new cuckoo graph presented in [DMR11a] is solely by Michael Mitzenmacher.

Section 4.1 The experimental results from [ADR09] on the basis of pseudoinverses are by Martin Aumüller.

Sections 4.4 and 4.5 My main contributions to [DGM+10] concerned the placement algorithm and experiments, but not the theoretical results that set the starting point for [Rin13].

Section 5.4 The proof of the main theorem of [DR09] is joint work.

Contents

1

Introduction

We consider four fundamental static data structure problems know as *membership*, *perfect hashing*, *dictionary* and *retrieval*, see, e. g., [Pag01b, DMPP06]. At their core they can be seen as tasks of representing a function

$$f\colon U \to V\,,$$

via building a *static data structure* $\mathfrak{D}$ that supports a "lookup" operation for elements from U according to

$$\texttt{lookup}(\mathfrak{D}, x) \coloneqq f(x) \text{ for all } x \in U.$$

We assume that U and V are *finite*. Let $S = \{x_0, x_1, \ldots, x_{n-1}\} \subseteq U$ be a set of n keys, and let $g = \{(x_0, v_0), (x_1, v_1), \ldots, (x_{n-1}, v_{n-1})\} \subset S \times V$ be a set of n key-value pairs. Then the four problems can be specified as follows:

Membership — Given S, represent the characteristic function of S, i. e., $f(x) = 1$ if $x \in S$ and $f(x) = 0$ if $x \in U - S$. We refer to the data structure as *membership tester*.

Perfect Hashing — Given S and an integer $m \geqslant n$, represent an arbitrary but fixed function $f\colon U \to \{0, 1, \ldots, m-1\}$ that is injective on S, i. e., for all $x, x' \in S$ with $x \neq x'$ we have that $f(x) \neq f(x')$. We refer to the data structure as *perfect hash function*.

Dictionary — Given g and an element $\perp \in V$ with $\perp \neq g(x)$ for all $x \in S$, represent the function $f\colon U \to V$ with $f(x) = g(x)$ for all $x \in S$ and $f(x) = \perp$ for all $x \in U - S$. The value $\perp$ is interpreted as "$x \notin S$". We refer to the data structure as *dictionary*.

Retrieval — Given g, represent an arbitrary function $f\colon U \to V$ with $f(x) = g(x)$ for all $x \in S$, i. e., the function values for elements from $U - S$ are not relevant. We refer to the data structure as *retrieval data structure*.

Needless to say, the problem of implementing the (abstract) data structures becomes only interesting if the available resources are limited. Usually, there are constraints

concerning the consumption of space, time, and random bits during and after the construction.

In this thesis we will consider *variants* of known efficient data structures that are based on the use of "random functions", commonly denoted as *hash functions*. These *hashing-based* data structures essentially consist of a table with m cells, each capable of holding entries of size r bits, as well as of a constant number of hash functions, that are used to map elements from U to a constant number of table cells. We study what load factor n/m or space consumption $m \cdot r$ these randomized data structures can achieve asymptotically almost surely, and moreover, how they can be constructed efficiently. This translates to the questions if their underlying *bipartite graphs admit a matching* of a certain type and how such a matching can be calculated efficiently.

1.1. Dictionary and Membership

A standard data structure to solve the membership or dictionary problem is a *hash table*. Different forms of hash tables have been widely used and studied for decades, see, e. g., [Knu98, page 547ff.]. For the static case Fredman, Komlós, and Szemerédi gave the first construction[1], now usually referred as *FKS scheme*, that guarantees worst case constant lookup time and simultaneously needs space only proportional to the space usage of the stored keys or key-value pairs, respectively [FKS82, FKS84]. Later their approach was extended to the dynamic case [DKM+88, DKM+94], allowing updates in constant expected amortized time. In recent years a new hash table, called *cuckoo hashing*, attracted much attention. The static version was analyzed by Pagh in [Pag01b] and the dynamic version was introduced by Pagh and Rodler in [PR01, PR04]. Cuckoo hashing has the same theoretical properties as the FKS scheme, in the static as well as in the dynamic case, but is easier to implement and uses very little space; the load factor n/m is about 0.5. This efficiency and simplicity gave rise to many variants, which likewise the *standard* cuckoo hashing are appealing from a theoretical point of view, see, e. g., [Mit09], as well as from a practical point of view, see, e. g., [EMM06, Ros07, ASA+09]. One particularly interesting generalization is d-*ary cuckoo hashing* [FPSS03, FPSS05], which allows to increase the load factor arbitrary close to 1 at the price of an increased but constant lookup time; the case $d = 2$ corresponds to standard cuckoo hashing.

Structure and Function The basic structure of d-ary cuckoo hashing is simple. There are two parts, a table of m cells, where each cell can hold exactly one entry of a fixed type, and a mapping that associates each $x \in U$ with d fully random table cells. For a successful construction, each key $x \in S$ (membership) or each key-value pair $(x, g(x))$ (dictionary) must be placed into one of its associated table cells. A lookup for

[1] Actually this data structure can be interpreted as *perfect hash function*.

a key $x \in U$ probes the table cells associated with x and returns 1 if x is found, or $g(x)$ if $(x, g(x))$ is found, respectively. Otherwise, it returns 0 or $\perp$, meaning that $x \notin S$. The association of keys with table cells can be represented by a random d-left regular bipartite graph with n left nodes and m right nodes, where a successful construction is possible if and only if this graph admits a *left-perfect matching*, i. e., a matching that covers all left nodes. Clearly, the probability of a successful construction, short *success probability*, is a function of the parameters n, m, and d.

Worst-case Number of Cells Probes For the case $d = 2$ the success probability of d-ary cuckoo hashing can be determined very precisely [DK12], and for the case $d \geqslant 3$ at least asymptotically, see, e. g., [FP12, FM12, DGM+10]. It follows from these results that for fixed $d \geqslant 2$ and increasing load factor $c = n/m$ there is a sharp phase transition between the situations that there is (asymptotically almost surely) a left-perfect matching and there is (asymptotically almost surely) no left-perfect matching, and furthermore, the transition points or *load thresholds* $\hat{c}(d)$, respectively, are exactly known. In consequence, the trade-off between asymptotic maximum load of d-ary cuckoo hashing, which is just below $\hat{c}(d)$, and the worst-case number of cell probes for a lookup, which is d, is well understood. For example, with $d = 4$ one can already achieve a load factor of about 0.97 and with $d = 5$ even about 0.99.

Average Worst-case Number of Cell Probes The more general situation, where the left nodes are allowed to have different degrees, is also settled in the sense that sharp load thresholds for the existence of left-perfect matchings can be determined [DGM+10]. In this context the question arises which distribution maximizes the success probability given some constraint on the left degree. If one aims for maximum degree d, then clearly one cannot do better than giving each left node degree d. However, allowing individual left degrees, and given a target average left degree $\bar{d} \in \mathbb{N}$, then some irregular distributions could not only give a higher success probability, compared to the $\bar{d}$-regular case, but could also lead to a larger threshold.

Overview Over Result 1. We show that irregularity does not help to improve the success probability and hence gives no improvement with respect to the load thresholds. More precisely, assuming that the degree of each left node x is a random variable D_x with almost arbitrary probability mass function, we prove that if the average expected degree $\bar{d} = (1/n) \cdot \sum_{x \in S} \mathrm{Exp}(D_x)$ is integral, then it is optimal if each left node has degree $\bar{d}$, and if $\bar{d}$ is non-integral, then it is asymptotically optimal if the left nodes have only two possible degrees, $\lfloor \bar{d} \rfloor$ and $\lceil \bar{d} \rceil$, which according to [DGM+10] leads to a threshold between the thresholds for the $\lfloor \bar{d} \rfloor$-left regular and $\lceil \bar{d} \rceil$-left regular case.

Linear Time Algorithms Beside the existence of a matching, we are interested in its efficient computation. For the case $d = 2$, Pagh showed how to determine a matching,

if it exists, in expected linear time via a reduction to 2-SAT [Pag01b, Section 2.3]. For $d > 5$ and a load factor at some distance to the corresponding theoretical threshold $\hat{c}(d)$, Fotakis et al. proved that a matching can be computed in expected linear time via a modified breadth first search algorithm that determines shortest augmenting paths up to a certain length [FPSS05, Theorem 1]. According to these two results, there is a gap that we intend to close.

Overview Over Result 2. Derived from the *selfless algorithm* by Sanders [San04], we develop a simple matching algorithm with linear running time and give experimental evidence that this algorithm is likely to find a matching, if such one exists, up to the theoretical threshold value $\hat{c}(d)$ for all $d \geqslant 2$. Additionally, we give a further generalization for determining a placement in the situation that each $x \in S$ must store k items, each into a different table cell, where each cell can hold ℓ items. For fixed d, k, and ℓ with $d > k \geqslant 1$ and $\ell \geqslant 1$ there is a sharp phase transition with respect to the probability that a legal placement exits and the load thresholds $\hat{c}_{k,\ell}(d)$ are known, see, e. g., [Lel12a, Lel12b]. This very general variant of the selfless algorithm also runs in linear time and is likely to work up to these thresholds.

Remark. After the development of our *generalized selfless algorithm*, a novel algorithm for finding left-perfect matchings (in the static and dynamic case), called *local search allocation*, was proposed by Khosla [Kho13]. On input of a sparse random d-left regular bipartite graph, *local search allocation* has linear running time with high probability if $c \leqslant \hat{c}(d) - \varepsilon$, for $d \geqslant 3$ and constant $\varepsilon > 0$, and always finds a matching if the graph admits one, whereas the *generalized selfless algorithm* has worst-case linear running time and the best we can hope for is a proof that it finds a matching with high probability if there exists one. Note that it was shown in [CSW07] that the original *selfless algorithm*, which corresponds to our algorithm with parameters $d = 2$, $k = 1$, and $\ell \geqslant 2$, obtains a placement asymptotically almost surely if $c \leqslant \hat{c}_{k,\ell}(d) - \varepsilon$ for constant $\varepsilon > 0$.

Average Number of Page Accesses Concerning the lookup operation of d-ary cuckoo hashing, up to now our focus was on the number of cell probes. This seems a reasonable cost criteria under the assumption that cell probes dominate the running time and that each single probe approximately needs the same time. However, this assumption does not hold in general, especially if we have to take a memory hierarchy into account. A standard setting is when the memory, which in our simplified view is equivalent with the table, is *subdivided into pages* or sub-tables, respectively, and the number of (different) pages that have to be accessed for lookup becomes the primary cost. In this case it is a drawback that the cells that are associated with keys from U are spread uniformly at random throughout the table. A natural modification would be to evenly split the key set S into subsets $S_0, S_1, S_2, \ldots$ and then, for each subset S_i, to keep a separate hash table in page i. This limits the number of pages examined to

one. However, a remaining problem is that the most overloaded page limits the load of the entire data structure. This could significantly affect the maximum achievable load, at least if one excludes large pages, i. e., pages of size $m^\delta, \delta \in (0,1)$, for constant δ.

Overview Over Result 3. We demonstrate in experiments with small pages and each key confined to one page, that the maximum achievable load factor is quite low. However, if each key has d cell choices on one page and a single cell choice on a second page, then one can determine an optimal placement such that the number of page accesses for lookup is close to 1. Simultaneously, the load factor is close to the situation where each key has $d+1$ random cell choices, even when using very small pages. Furthermore, we give a simple and fast algorithm, based on the *Hopcroft-Karp algorithm* [HK73], to determine such an optimal placement in the static case and demonstrate that a simple variation of the well-known *random walk* insertion procedure, see, e. g., [FPSS05, Experiments], allows to determine a near optimal placement in the dynamic case.

1.2. Retrieval and Perfect Hashing

Hash tables can be used straightforwardly to solve retrieval. If no two keys share the same table cell, as for example in the FKS scheme, then one stores for key $x \in S$ only its function value $g(x)$ in the table. Otherwise, it is required to store (some representation of) the keys in addition, as for example in the case of d-ary cuckoo hashing. An elegant static data structure for retrieval that works without solving perfect hashing in the first place, and without using any explicit representation of keys is the *immutable Bloomier filter* by Chazelle, Kilian, Rubinfeld, and Tal [CKRT04, Section 3.1], which is a kind of generalization of the classical Bloom filter[2] [Blo70].

The construction principle of the immutable Bloomier filter was known before, see, e. g., [MWHC96]. However, for their *mutable Bloomier filter* [CKRT04, Section 3.3] Chazelle et al. introduced a clever improvement that allows to solve perfect hashing very efficiently. Later this method was independently rediscovered, see, e. g., [BPZ07, BPZ13].

Structure and Function Both immutable and mutable Bloomier filter share their basic structure with d-ary cuckoo hashing. More precisely, we have n key-value pairs from $S \times V$, a table of m cells of capacity one, and a mapping that associates each $x \in U$ with d fully random table cells. For a successful construction one must fill the table with elements from V such that for each $x \in S$, the sum of the entries from its d associated table cells is exactly $g(x)$. Such a solution can be determined in linear time if $(V, \oplus)$ is an abelian group and the underlying random d-left regular bipartite graph has a certain left-perfect matching that we refer as *order generating matching*. An order generating matching allows at least one complete ordering π on the elements of

[2] The conceptual step from Bloom filter to Bloomier filter and the relationship with retrieval is discussed is Section 4.2.2.1.

$S = \{x_0, x_1, \ldots, x_{n-1}\}$, so that for all $0 \leqslant i < j < n$ the matched right node of $x_{\pi(i)}$ is not adjacent to a left node $x_{\pi(j)}$. Such an ordering translates into a permutation of the rows and columns of the adjacency matrix of the bipartite graph that leaves the matrix in row echelon form.

Now the main difference between immutable and mutable Bloomier filter lies in the function values. While in the case of retrieval these are given by the application, in the case of perfect hashing these are determined using an order generating matching. More precisely, for each $x \in S$ the value $g(x)$ is the index of its matching edge, where we assume that for each left node its incident edges are numbered consecutively from 0 to $d - 1$. A lookup for a key $x \in U$ probes the table cells associated with x (in arbitrary order) and computes the sum ν of its values. Assuming that the construction was successful, ν must be equal to $g(x)$ if $x \in S$. Otherwise ν has an arbitrary value. In the case of retrieval ν is returned, and in the case of perfect hashing the address that corresponds to the right node of the edge that has index ν is returned. As before, we focus on the success probability, which is equivalent to the probability that a d-left regular random bipartite graph with parameters n, m, and d has an order generating matching.

MAXIMUM LOAD Like in the case of general left-perfect matchings it is known that for increasing load factor $c = n/m$ there is a phase transition concerning the existence of an order generating matching. If c is below a certain threshold $\check{c}(d)$, then, asymptotically almost surely if $d \geqslant 3$ and with constant limiting probability if $d = 2$, there is an order generating matching; if c is above this threshold, then there is no order generating matching asymptotically almost surely, see, e. g., [MWHC96, Mol04]. However, in contrast to general left-perfect matchings, the thresholds $\check{c}(d)$ are not monotonically increasing with increasing d; the maximum is about 0.818 for $d = 3$.

Therefore, even more than in the case of d-ary cuckoo hashing the question arises whether an irregular left degree distribution could give an improvement over the regular case. Interestingly, in related random bipartite graphs that arise in the context of erasure correcting codes it was shown that carefully chosen irregular degree distributions can increase the probability of an order generating matching. These distributions are applied pairwise to left and right nodes, see, e. g., [LMSS01], or only to the right nodes, see, e. g., [Lub02].

Overview Over Result 4. We generalize the known threshold formula for order generating matchings to random bipartite graphs with irregular left degree distributions. For the case of two fixed left degrees, we show how to efficiently obtain a distribution that maximizes the possible load factor. Moreover, for a suitable choice of two degrees we obtain thresholds up to 0.920, significantly improving upon the maximum value obtainable in the d-left regular case for any d.

Improved Perfect Hash Function Using an irregular left degree distribution that increases the maximum possible load does not necessarily decrease the space usage of a mutable Bloomier filter. This is because the function values that we have to store are not fixed but depend on the graph structure and increase with increasing maximum left degree.

Overview Over Result 5. We generalize the standard mutable Bloomier filter construction by using two different bipartite graphs, one with regular left degree to obtain a left-perfect matching and another one with irregular left degree to obtain an order generating matching. In contrast to the (optimal) standard Bloomier filter, where one can store about $m = 1.23 \cdot n$ "trits", i. e., elements from $\{0, 1, 2\}$, this allows to reduce the table size to $m = 1.1 \cdot n$ trits. In addition, the range of the perfect hash function is decreased from $\acute{m} = 1.23 \cdot n$ to $\acute{m} = 1.1 \cdot n$. Furthermore, we experimentally show that our construction can be used to solve perfect hashing spending 1.76 bits per key for $n \geqslant 10^7$.

1.3. Uniform Hashing

For the analysis of the hashing-based data structures that we consider, we assume that the keys are mapped to table cells using hash functions that are *fully random*. A direct consequence is that the edges of the underlying bipartite graphs are chosen uniformly and independently at random. This is an idealized, but nevertheless common, assumption whose realization requires non-negligible additional space and which is highly non-trivial if one simultaneously aims for constant evaluation time of the hash functions.

Simulation of Full Randomness An, at least theoretical, justification for this assumption is given by an efficient simulation of full randomness. Here on input n a randomized algorithm builds a data structure that represents a hash function which can be evaluated in constant time and for each key set S of size up to n the function is fully random on S with high probability. A construction with asymptotically optimal space usage and constant, but very high, evaluation time was given by Pagh and Pagh [PP08]. An alternative approach allowing faster evaluation time, with space consumption essentially twice the information theoretical lower bound, was later proposed by Aumüller, Dietzfelbinger, and Woelfel [ADW12]. This shows that there is some room of further improvement.

Overview Over Result 6. We give an algorithm for the simulation of full randomness where the representation of the hash function needs asymptotically optimal space and the evaluation time is exponentially smaller compared with the construction by Pagh and Pagh. Our construction has the currently best performance characteristic.

1.4. Thesis Outline

In the next chapter we start by introducing some notation and terminology and subsequently describe the scheme that underlies all hashing-based data structures considered in this work, including its abstract graph, hypergraph, and matrix representations. In Chapter 3 we focus on how to efficiently solve dictionary and membership using variants of d-ary cuckoo hashing. The following Chapter 4 covers improvements of the Bloomier filter construction for solving retrieval and perfect hashing. Chapter 5 is devoted to uniform hashing, which is implicitly assumed in the two preceding chapters, and discusses an efficient approach for its simulation. The thesis finishes with concluding remarks and some open problems.

1.4.1. Structure of the Main Chapters

The Chapters 3 to 5, which constitute the central part of this thesis, share the same structure. First there is an *introduction* section that highlights some known facts, theorems, and problems in order to prepare the presentation of the new findings, which are stated in separate subsections; later, each result is discussed in detail in its own section. After the introduction, it follows a section entirely devoted to *further background and related work* that places the results in a broader context. Then some *basic facts and definitions* are given before the most important part, the sections containing the actual *proofs and experimental findings*, begins. Each chapter finishes with a short *conclusion*.

2

Preliminaries

After a discussion of some nomenclature[1] used throughout this thesis, we introduce the hashing scheme that is the basis of the data structures we cover in the following chapters.

2.1. Notation and Terminology

Sets For each integer $i \in \mathbb{N}$, we let $[i]$ be the set $\{0, 1, 2, \ldots, i-1\}$. The *symmetric difference* of two sets A and B is denoted by $A \oplus B$. The *power set* of any set A is written $\mathcal{P}(A)$. The set of all possible inputs or *keys* is denoted by U and the set of keys that actually occur in an application is denoted by S, $S \subseteq U$. We assume that U, called the *universe*, has a simple structure comparable with $[u]$ or $[c]^r$ for integers u, c, and r. The cardinality of U is an integer larger than 1; the cardinality of S is denoted by n.

Matrices Matrices $\boldsymbol{A} = (a_{i,j})_{i \in [k], j \in [l]}$ are written in bold and their indices start with zero. This also holds for the special cases of column vectors $\boldsymbol{a} = (a_j)_{j \in [l]} = (a_0, a_1, \ldots, a_{l-1})^\intercal$ and row vectors $\boldsymbol{a}^\intercal$.

Probability We will often consider events that depend on n, the number of relevant keys. If the probability of such an event $\mathcal{E}_n$ approaches 1 if n goes to infinity, i. e., $\Pr(\mathcal{E}_n) = 1 - o(1)$, then we say the event occurs *asymptotically almost surely* (a. a. s.). And if we can bound the probability even via $1 - O(1/n^\delta)$, for some constant $\delta > 0$, then we say the event occurs *with high probability* (w. h. p.). The probability distribution of a discrete random variable Y is characterized via its *probability mass function* ρ_Y, given by $\rho_Y(y) = \Pr(Y = y)$ for all y. In the case of special distributions we will make use of the following definitions. A random variable that is *fully random* follows a certain *uniform distribution*. The *binomial distribution* with sample size k and success probability p is denoted by $\mathrm{Bin}(k, p)$; and we write $\mathrm{Bin}[k, p]$ for a random variable that follows this distribution. Moreover, in order to indicate that Y is $\mathrm{Bin}(k, p)$-distributed, we use the notation $Y \sim \mathrm{Bin}(k, p)$. Analogously, $\mathrm{Po}(\lambda)$ denotes the *Poisson*

[1] A comprehensive list of the notation can be found in Appendix A.2.

distribution with parameter λ, $\mathrm{Po}[\lambda]$ is an anonymous random variable that follows this distribution, and $Y \sim \mathrm{Po}(\lambda)$ indicates that Y is $\mathrm{Po}(\lambda)$-distributed.

STATISTICS For any sequence of observations $y_0, y_1, \dots y_{k-1}$ let $\bar{y}$ denote the *sample mean* $\bar{y} = \frac{1}{k}\sum_{i\in[k]} y_i$ if $k > 0$, and let $s^2[y]$ denote the unbiased *sample variance* $s^2[y] = \frac{1}{k-1}\sum_{i\in[k]}(y_i - \bar{y})^2$ if $k > 1$.

COMPLEXITY For theoretical results, we quantify *space consumption* $\mathcal{S}$ in bits and *time consumption* $\mathcal{T}$ in word operations on a unit-cost RAM with word size $\Theta(\log|U|)$, see, e. g., [Hag98, Section 2].

PERFORMANCE For experimental results, we measured *space consumption* in bits and *time consumption* in seconds on an Intel Core i3-3225 CPU (x86-64) @ 3.30 GHz with 32 Kib L1d and L1i cache, 256 Kib L2 cache, as well as 3072 KiB L3 cache, running Linux version 3.7.10-1.1-desktop.

FUNCTIONS For any function $f = f(x, y)$ we will use $f(x)$ and $f(x, y)$ *synonymously*, if y is considered to be fixed. We write log for the *logarithm* to the base 2 and ln for the logarithm to the base e. A *hash function* can be any function that maps keys from U to some range R. Typically, it is assumed that the sequence of *hash values* $(h(x))_{x\in U}$ is a random variable that follows some probability distribution. The source of randomness can be due to the choice of the function, or due to the choice of S if we restrict ourselves to the sequence $(h(x))_{x\in S}$. A hash function h is *realized* via a data structure $\mathfrak{D}_h$ that supports an "evaluation" operation for elements from U according to

$$\texttt{evaluate}(\mathfrak{D}_h, x) \coloneqq h(x) \text{ for all } x \in U.$$

The *description size* of h is the space consumption of its data structure and the *evaluation time* of h is the time consumption of the evaluation operation in the worst-case over all x from U.

GRAPHS Unless specified otherwise, graphs are meant to be undirected and with multiple edges. An *undirected edge* between two nodes v, v' is denoted by $\{v, v'\}$, a *directed edge* from v to v' is denoted by (v, v'). Given an undirected graph $G = (V, E)$ with node set V and edge multiset E, for each $V' \subseteq V$ its *neighborhood* is defined as the node set

$$N(V') \coloneqq \left\{v \in V \mid \{v, v'\} \in E, v' \in V'\right\}.$$

A *bipartite graph* $G = (V, E)$ with disjoint, independent node sets L and R, $L \cup R = V$, and edge set $E \subseteq \left\{\{x, y\} \mid x \in L, y \in R\right\}$ is written as $G = (L \cup R, E)$. The set L is called

left node set, and its elements are called left nodes; the set R is called *right node set*, and its elements are called right nodes.

A *matching* M in $G = (V, E)$ is a subset of the edge multiset E with the property that the edges from M are pairwise disjoint. With respect to a given matching M we use the following definitions. An edge from M is called *matching edge*, an edge that does not belong to M is called *non-matching edge*; and similarly, a node is called *matched* if it is incident to a matching edge, otherwise it is called *unmatched*. An *alternating path* (or cycle) is a simple path (or cycle) where the edges are alternately matching and non-matching edges. An *augmenting path* is an alternating path that starts and ends with an unmatched node.

2.2. The Basic Scheme

Our starting point is a simple hashing scheme that resembles the structure of a Bloom filter [Blo70]. It consists of a universe U, a *table* $\mathbf{t} = (t_0, t_1, \ldots, t_{m-1})^\intercal$ with m *cells*, as well as d hash functions $h_0, h_1, \ldots, h_{d-1}$. The hash functions are used to build a mapping $h\colon U \to [m]^d$, via

$$x \mapsto \big(h_0(x), h_1(x), \ldots, h_{d-1}(x)\big)\,,$$

which associates each element from U with a *sequence of addresses* or indices of table cells, respectively. For each key $x \in U$ its *set of addresses* is defined as

$$A_x = \{h_i(x) \mid i \in [d]\}\,.$$

2.2.1. Variants

There are three common types of the basic scheme with respect to the mapping h, as illustrated in Figure 2.2.1. In this thesis we will mainly focus on the first two of them.

Type A The hash function values are *not restricted* (Figure 2.2.1 (a)) according to

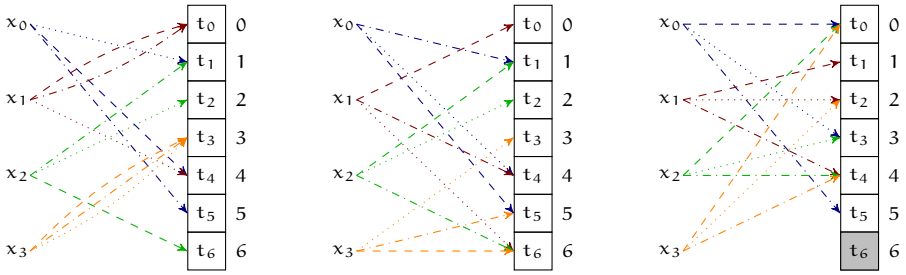
$$h_i\colon U \ni x \mapsto a \in [m] \text{ for all } i \in [d],$$

which implies that duplicate addresses are allowed.

Type B The hash addresses for each $x \in U$ are *pairwise distinct* (Figure 2.2.1 (b)), meaning that

$$h_i(x) \neq h_j(x) \text{ for all } x \in U \text{ and } 0 \leqslant i < j < d, \text{ where}$$
$$h_i\colon U \ni x \mapsto a \in [m] \text{ for all } i \in [d].$$

The mapping h for type B can be realized with only slightly more space and time consumption compared to a realization of h for type A, see Section 5.3.3.2 for details.



(a) type A (b) type B (c) type C

Figure 2.2.1.: Exemplary illustration of three different types of the basic scheme for four keys, $m = 7$, $d = 3$ and hash functions $h_0(\text{- - }\rightarrow), h_1(\cdots\rightarrow), h_2(\text{-}\cdot\text{-}\rightarrow)$. In the example of type C, cell t_6 is never addressed by $h(x)$ for all x from U.

TYPE C The hash addresses are from *disjoint ranges* (Figure 2.2.1 (c)) according to

$$h_i : U \ni x \mapsto a \in \{i \cdot \lfloor m/d \rfloor, i \cdot \lfloor m/d \rfloor + 1, \ldots, (i+1) \cdot \lfloor m/d \rfloor - 1\} \text{ for all } i \in [d].$$

This variant is the most convenient one for parallelization. Note that $m - d \cdot \lfloor m/d \rfloor = m \bmod d$ cells cannot be addressed.

2.2.2. Properties of the Hash Functions

ASSUMPTIONS In the subsequent Chapters 3 and 4 we assume that the hash functions are ideal in the following sense.

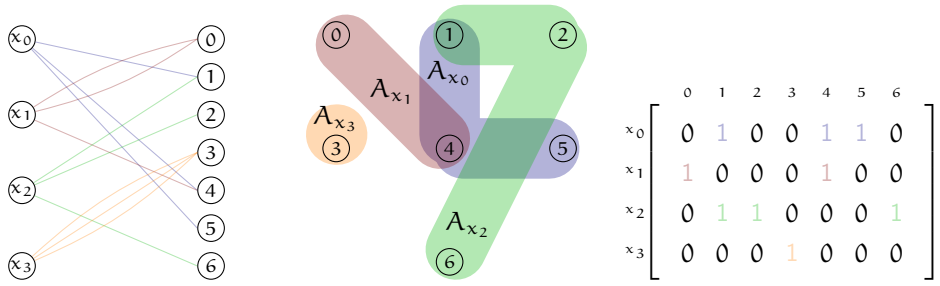
(i) For each $x \in S$ and each h_i, $i \in [d]$, the hash value $h_i(x)$ is a fully random variable and $h_0(x), h_1(x), \ldots, h_{d-1}(x)$ are independent. (The support of $(h_i(x))_{i \in [d]}$ depends on the type of the basic scheme.)

(ii) The evaluation of the hash functions can be done in constant time.

(iii) The space consumption is negligible.

A consequence of (i) is that the three types A, B, and C are hardly distinguishable if d is constant and n, m go to infinity. It follows from (ii) and (iii) that with d being a constant, the space consumption of the basic scheme is m times the size of a table cell and the evaluation time of h is constant too.

JUSTIFICATION Although widely used in the analysis of hashing-based algorithms, there is no known possibility to fulfill all three assumptions simultaneously without supposing that the keys from S are fully random, see Section 5.3.1. However, it is possible to realize hash functions that have property (i) with high probability and have constant evaluation time (ii) independently of the choice of S, while using space



(a) bipartite graph $G^d_{n,m}$ (b) hypergraph $H^d_{m,n}$ (c) binary matrix $\mathbf{M}^d_{n,m}$

Figure 2.3.2.: Bipartite graph, hypergraph, and matrix view of the basic scheme type A from Figure 2.2.1 (a) with $S = \{x_0, x_1, x_2, x_3\}$, $n = 4$, $m = 7$, and $d = 3$.

$\Omega(n \cdot \log m)$, which violates (iii). In Chapter 5 we discuss common methods for such realizations, and moreover, we contribute a new space and time efficient variant.

Remark. If we are only interested in asymptotic behavior, we can circumvent property (i) in our data structure setting via a method known as *split-and-share*. Here the key set S is *split* into small subsets by a splitting hash function and for each subset a separate data structure is built that *shares* its random mapping h with the other data structures. Since the mapping has description size $o(n \cdot \log m)$, property (iii) is met; details are given in Section 5.1.

2.3. Representations

More than the description of the basic scheme in terms of keys, cells, and hash functions we will stress its abstract representations as *bipartite graph*, *hypergraph*, and *binary matrix*. Figure 2.3.2 shows a concrete example and Table 2.3.1 provides an overview over the terms that correspond to each other.

! In this context we consider not the whole universe U but only the set S. However, under the assumption of ideal hash functions the representations are *random variables* which depend not on the concrete elements of S but on its size n.

Bipartite Graph The basic scheme naturally translates into a *bipartite graph*

$$G^d_{n,m} = (S \cup [m], E_G)$$

consisting of two disjoint node sets, a left node set S of size n and a right node set $[m]$, as well as an edge multiset

$$E_G = \big\{\{x, h_i(x)\} \mid x \in S, i \in [d]\big\}.$$

parameters	basic scheme	graph $G^d_{n,m}$	hypergraph $H^d_{m,n}$	matrix $\mathbf{M}^d_{n,m}$
n	keys	left nodes	hyperedges	rows
m	cells	right nodes	nodes	columns
d	addresses	left degree	edge size	row weight

Table 2.3.1.: Related terms in the different representations of the basic scheme.

The degree of each left node is d, i. e., the graph $G^d_{n,m}$ is d-left *regular*. By definition, the neighborhood $N(\{x\})$ is equal to A_x. Figure 2.3.2 (a) shows an example.

HYPERGRAPH Alternatively, the basic scheme can be seen as *hypergraph*

$$H^d_{m,n} = ([m], E_H)$$

with node set $[m]$ and hyper-edge multiset

$$E_H = \{A_x \mid x \in S\},$$

i. e., table cells correspond to nodes and keys correspond to edges. Figure 2.3.2 (b) shows an example. In the case of type B and type C all edges A_x have cardinality d, which makes $H^d_{m,n}$ a d-*uniform* hypergraph.

MATRIX Finally, we will interpret the basic scheme as random *binary matrix*

$$\mathbf{M}^d_{n,m} = (\mathbf{b}_x^\intercal)_{x\in S} \in \{0,1\}^{|S|\times m},$$

where $\mathbf{M}^d_{n,m}$ is the *adjacency matrix* of $G^d_{n,m}$ and the *incidence matrix* of $H^d_{m,n}$, respectively. That is, for each $x \in S$ the row vector $\mathbf{b}_x^\intercal = (b_{x,0}, b_{x,1}, \ldots, b_{x,m-1})$ is the characteristic vector of A_x defined as follows. For all $x \in S$ and $i \in [m]$ we have $b_{x,i} = 1$ if $i \in A_x$, and $b_{x,i} = 0$ otherwise. Figure 2.3.2 (c) shows an example. We refer to the number of 1's in a row as the row *weight*, which is *uniformly* d in the case of type B and type C.

2.3.1. CORES

With regard to the abstract representations of the basic scheme we will often consider the 2-*core* of $G^d_{n,m}$, $H^d_{m,n}$, and $\mathbf{M}^d_{n,m}$, respectively. The 2-core is the unique outcome of a simple procedure that successively removes from:

- $G^d_{n,m}$ a right node of degree 0, or a right node of degree 1 together with the adjacent left node and all edges incident to this left node,
- $H^d_{m,n}$ a node of degree 0, or a node of degree 1 together with its incident edge,

▷ $\mathbf{M}^{d}_{n,m}$ a column of weight 0, or a column of weight 1 together with the row that contributes the 1 to this column.

In this work, we will mainly stress the hypergraph view, and occasionally we will not only consider the 2-core but more general the ℓ_+-core, for non-zero $\ell_+ \in \mathbb{N}$, which with respect to hypergraphs is shortly defined as follows.

> DEFINITION 1: (ℓ_+-CORE)
> The ℓ_+-core of a hypergraph is the largest induced sub-hypergraph (possibly empty) that has minimum degree at least ℓ_+.

The standard procedure for determining the ℓ_+-core of a hypergraph, usually called *peeling*, is given as Algorithm 1.

ALGORITHM 1: peeling

```
Input  : Hypergraph H = (V, E), core parameter ℓ+ ∈ ℕ with ℓ+ ⩾ 1.
Output : The ℓ+-core of H.
while ∃v ∈ V: |{e ∈ E | v ∈ e}| ⩽ ℓ+ − 1 do
    V ← V − {v};
    E ← E − {e ∈ E | v ∈ e};
end
return (V, E)
```

We conclude this section with a high level view of deep results concerning the *appearance* and *edge density* of the 2-core in random hypergraphs $H^{d}_{m,n}$, where edge density means ratio of the number of edges to the number of nodes.

REMARK. As we will discuss in detail later, the edge density of the ℓ_+-core is strongly related to the existence of an *orientation* of $H^{d}_{m,n}$ with the property that each edge is directed (assigned) to exactly one of its nodes and the maximum indegree of each node is $\ell = \ell_+ - 1$.

We start by defining two functions, called *key function*

$$\mathrm{key}(\lambda, d) = \frac{\lambda}{d \cdot \Pr\left(\mathrm{Po}[\lambda] \geqslant 1\right)^{d-1}},$$

and *density function*

$$\mathrm{dens}(\lambda, d) = \mathrm{key}(\lambda, d) \cdot \frac{\Pr\left(\mathrm{Po}[\lambda] \geqslant 1\right)^{d}}{\Pr\left(\mathrm{Po}[\lambda] \geqslant 2\right)} = \frac{\lambda \cdot \Pr\left(\mathrm{Po}[\lambda] \geqslant 1\right)}{d \cdot \Pr\left(\mathrm{Po}[\lambda] \geqslant 2\right)}.$$

For $\lambda > 0$ and fixed $d \geqslant 3$ the key function is strictly convex with *global minimum*

$$\check{c}(d) \coloneqq \min_{\lambda \in (0,\infty)} \mathrm{key}(\lambda, d),$$

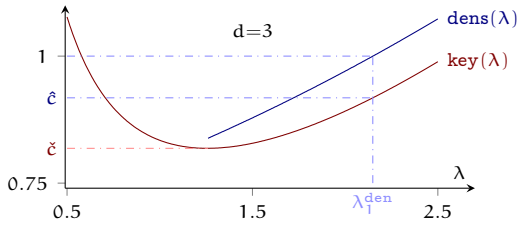


Figure 2.3.3.: Visualization of the definition of 2-core thresholds $\check{c}(3)$ and $\hat{c}(3)$.

and the density function is strictly monotonically increasing for increasing λ. The following tight connection between these two functions and the edge density of the 2-core of $H^d_{m,n}$ was shown for type B, e. g., in [Coo04, Mol04, Kim06] as well as [BK06, $d = 4$], and was recently proven also for type C in [BWZ12]; it also holds for type A.

Let $d \geqslant 3$. If the edge density $c = n/m$ of $H^d_{m,n}$ is smaller than $\check{c}(d)$, then a. a. s. the random hypergraph has an empty 2-core. However, if c is larger than the global minimum of the key function, then there is exactly one $\lambda = \lambda_c^{key}$ to the right of $\arg\min_{\lambda\in(0,\infty)} \mathrm{key}(\lambda, d)$, such that $\mathrm{key}(\lambda_c^{key}, d) = c$, and a. a. s. in this case the hypergraph has a non-empty 2-core whose edge density is tightly concentrated around $\mathrm{dens}(\lambda_c^{key})$. Conversely, let $\hat{c}(d)$ be the value of the key function at the point where the density function equals 1, i. e.,

$$\hat{c}(d) := \mathrm{key}(\lambda_1^{den}, d), \text{ where } \lambda_1^{den} \in (0,\infty) \text{ is the solution of } \mathrm{dens}(\lambda, d) = 1.$$

It follows that if $c = \hat{c}(d)$, then the edge density of the 2-core of $H^d_{m,n}$ is tightly concentrated around 1. Figure 2.3.3 shows a plot of $\mathrm{key}(\lambda, d)$, $\mathrm{dens}(\lambda, d)$, $\check{c}(d)$, and $\hat{c}(d)$ for $d = 3$. In summary, we have the following theorem.

> Theorem 2.1 (Density of 2-Core, see, e. g., [BWZ12, Theorem 3])
> *Let $d \geqslant 3$ be constant and let $c = n/m$. Then for any constant $\varepsilon > 0$ a. a. s. the following holds:*
>
> *(i) If $c \leqslant \check{c}(d) - \varepsilon$, then $H^d_{m,n}$ has an empty 2-core.*
>
> *(ii) If $c \geqslant \check{c}(d) + \varepsilon$, then $H^d_{m,n}$ has a non-empty 2-core.*
>
> *Moreover, if $c \geqslant \check{c}(d) + \varepsilon$, then the edge density of the 2-core of $H^d_{m,n}$ is tightly concentrated around $\mathrm{dens}(\lambda_c^{key})$, where λ_c^{key} is the unique solution of $\mathrm{key}(\lambda) = c$. In particular, for any constant $\varepsilon > 0$ a. a. s. we have:*
>
> *(i) If $c \leqslant \hat{c}(d) - \varepsilon$, then the 2-core of $H^d_{m,n}$ has edge density smaller than 1.*
>
> *(ii) If $c \geqslant \hat{c}(d) + \varepsilon$, then the 2-core of $H^d_{m,n}$ has edge density larger than 1.*

With this background we are ready to discuss efficient solutions of data structure problems using the basic scheme.

3

Dictionary and Membership

In this chapter we cover solutions of the membership and dictionary problem based on d-ary cuckoo hashing [FPSS05]. The two data structure problems can be briefly restated as follows.

Membership — Given S, build a data structure $\mathfrak{D}$ that on $\texttt{lookup}(\mathfrak{D}, x)$ returns 1 if $x \in S$, and returns 0 if $x \in U - S$.

Dictionary — Given $g = \{(x_0, v_0), (x_1, v_1), \ldots, (x_{n-1}, v_{n-1})\} \subset S \times (V - \{\perp\})$, build a data structure $\mathfrak{D}$ that on $\texttt{lookup}(\mathfrak{D}, x)$ returns $g(x)$ if $x \in S$, and returns $\perp$ otherwise.

3.1. Cuckoo Hashing Variants

We start with a short description of (standard) d-ary cuckoo hashing and present a *new construction algorithm*. This algorithm runs in linear time, and we give experimental evidence that it builds cuckoo hashing data structures up to the theoretical load thresholds with low failure rate. It follows a discussion of a generalization called *irregular cuckoo hashing*. In this context we state the main result of this chapter, which essentially says that a left-irregular degree distribution of the underlying graph does not help to increase the load factor of standard d-ary cuckoo hashing. The last part of this section is devoted to a new cuckoo hashing variant, called *cuckoo hashing with pages*, that is adapted for settings where memory hierarchy must be taken into account.

3.1.1. d-Ary Cuckoo Hashing

Standard d-ary cuckoo hashing is a *multiple-choice* hash table on top of the basic scheme considered in Section 2.2. Its construction principle can be described as follows.

3.1.1.1. Construction

Building a static dictionary or membership tester $\mathfrak{D}$ using the basic scheme is straightforward. For ease of discussion, we assume that there is some placeholder $\breve{x} \notin U$ with

description size at most $\lceil \log|U| \rceil$. First we initialize the table $\mathbf{t} = (t_0, t_1, \ldots, t_{m-1})$ by storing in each cell the same entry, $(\check{x}, v)$, for arbitrary $v \in V$, if we want a dictionary; and only $\check{x}$ if we want a membership tester. Afterwards, we store for each $x \in S$ one specific entry in a cell t_a whose address a is from A_x, the set of addresses associated with x. This entry is the key-value pair $(x, g(x))$ in the case of a dictionary, and only the key x otherwise.

If each table cell can hold only one entry, then in order that $\mathfrak{D}$ can be built successfully, it is *necessary and sufficient* that there exists an injective mapping from S to $[m]$ such that for all $x \in S$ the image of x under this mapping is an address from A_x. This is the case if and only if the random bipartite graph $G^d_{n,m}$ that underlies the basic scheme admits a *left-perfect matching*, i. e., a set of pairwise disjoint edges that cover all left nodes. Given such a matching M, for each matching edge $\{x, a\} \in M$ the entry associated with x is stored in cell t_a.

A bipartite graph has a left-perfect matching if and only if it fulfills a weak expansion property [Hal35, follows by Theorem 1], known as *marriage condition*.

> Definition 2: (Marriage Condition)
> A bipartite graph $G = (L \cup R, E)$ fulfills the *marriage condition*, if for each subset L' of L the size if its neighborhood $N(L')$ is at least as large as the size of L',
>
> $$\forall L' \subseteq L\colon |L'| \leqslant |N(L')|\,. \tag{MC}$$

A left-perfect matching in $G^d_{n,m}$ corresponds to an *orientation* of the edges of the hypergraph $H^d_{m,n}$, such that each edge is directed to exactly one of its nodes and each node has indegree at most one. Furthermore, it corresponds to an $n \times n$ submatrix of the binary matrix $\mathbf{M}^d_{n,m}$ with *non-zero permanent*, i. e., the submatrix contains a permutation matrix. Figure 3.1.1 shows an example.

3.1.1.2. Lookup Operation

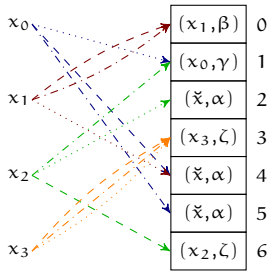
The lookup operation only needs to compare a given key $x \in U$ with the entries of $\mathbf{t}$ addressed by A_x in an arbitrary order. For all $x \in U$ we define

$$\texttt{lookup}(\mathfrak{D}, x) \coloneqq \begin{cases} v, & \text{if } \exists a \in A_x \quad : t_a = (x, v) \\ \bot, & \text{if } \neg\exists a \in A_x \exists v\colon t_a = (x, v) \end{cases},$$

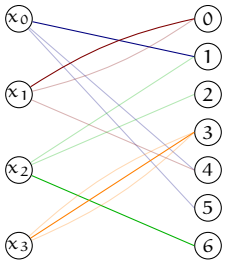
if $\mathfrak{D}$ is a dictionary, and analogously

$$\texttt{lookup}(\mathfrak{D}, x) \coloneqq \begin{cases} 1, & \text{if } \exists a \in A_x\colon t_a = x \\ 0, & \text{otherwise} \end{cases},$$

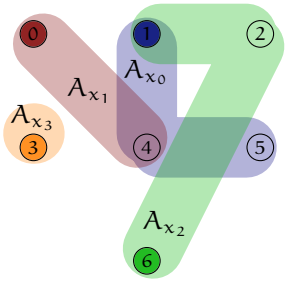
if $\mathfrak{D}$ is a membership tester.



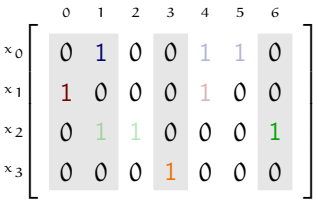
(a) dictionary build upon the basic scheme as given in Figure 2.2.1 (a), $\check{x} \notin U$ is a placeholder



(b) bipartite graph representation, a left-perfect matching is indicated via dark edges



(c) corresponding hypergraph, the orientation of the edges is indicated by dark nodes



(d) matrix representation, the gray submatrix with non-zero permanent corresponds to the matching

Figure 3.1.1.: A dictionary for $\{(x_0,\gamma),(x_1,\beta),(x_2,\zeta),(x_3,\zeta)\} \cup \{(x,\perp) \mid x \in U - S\}$, realized with the basic scheme (type A), as well as its corresponding graph, hypergraph, and matrix representations.

3.1.1.3. Success Probability

The probability of a successful construction is the probability that $G^d_{n,m}$ has a left-perfect matching, which is a function of the parameters n, m, and d. If n and m are fixed, then with increasing d this probability increases. We consider the situation that d is given due to the application and ask for the success probability as function of the load factor $c = n/m$ for $n \to \infty$.

Case $d = 1$ If each left node has only degree one, then $G^d_{n,m}$ admits a left-perfect matching if and only if each right node is incident to at most one edge. Note that in this case there is no difference between type A, B, and C. Commonly discussed in the context of the well known *birthday paradox*, this probability is exactly $m^{\underline{n}}/m^n$, which however is very low even for small n. Asymptotically, we have that if $n(m) = \omega(\sqrt{m})$,

then w. h. p. $G^d_{n,m}$ has no left-perfect matching, see Section 3.3.2. Hence in the case $d = 1$ the space consumption of the data structures is at least a factor of n away from the optimum, which is generally not very interesting. However, by increasing d only by one this factor is reduced to a constant.

CASE $d = 2$ If each left node has degree two, then one observes a sharp phase transition for increasing load factor $c = n/m$ as stated in the following theorem.

> THEOREM 3.1 (CUCKOO HASHING, SEE, E. G., [PAG01B, SECTION 2.3])
> *Let $d = 2$ and let $c = n/m$. Then for any constant $\varepsilon > 0$ w. h. p. the following holds:*
>
> *(i) If $c \leqslant 0.5 - \varepsilon$, then $G^d_{n,m}$ admits a left-perfect matching.*
>
> *(ii) If $c \geqslant 0.5 + \varepsilon$, then $G^d_{n,m}$ admits no left-perfect matching.*

The theorem follows for type A, e. g., from [DK12, Theorem 4], and by a modification of the proof of [CS97, Theorem 9][1]; and for type C, e. g., from [Pag01b, Section 2.3][2], [DM03, Theorem 1], and [DK12, Theorem 1]. It also holds for type B.

REMARK. The bipartite graph $G^d_{n,m}$ admits no left-perfect matching, if the graph $H^d_{m,n}$ has a component that is not a tree nor unicyclic. This is related to appearance of the *giant component* in $H^d_{m,n}$, which is known to happen (for type A, B, and C) if the edge density passes 0.5.

CASE $d \geqslant 3$ Increasing the left degree further allows an arbitrary load factor $c < 1$. For the extreme case $c = 1$, it is well known that one needs a left degree of $d = \Omega(\log n)$ in order that $G^d_{n,m}$ has a left perfect matching a. a. s., see, e. g., [Riv78, Section 3]. This is directly related to the classical *coupon collectors problem*, where one derives an upper and lower bound of $\Theta(n \log n)$ samples needed to obtain n different coupons, see, e. g., [ER61, Equation 2 with $m = 1$]. For the case $c \in (0.5, 1)$, Fotakis et al. [FPSS05, Lemmas 1 and 2] showed that $d = \Theta\left(\log\left(\frac{1}{1-c}\right)\right)$ is necessary and sufficient. Similarly to the case $d = 2$, there is a sharp phase transition. The points of transition are the thresholds $\hat{c}(d)$ where the 2-core density of $H^d_{m,n}$ switches from below 1 to above 1, see Section 2.3.1; some threshold values are given in Table 3.1.1.

> THEOREM 3.2 (d-ARY CUCKOO HASHING, SEE, E. G., [FP12, THEOREM 1.1])
> *Let $d \geqslant 3$ be a constant integer and let $c = n/m$. Then for any constant $\varepsilon > 0$ a. a. s. the following holds:*
>
> *(i) If $c \leqslant \hat{c}(d) - \varepsilon$, then $G^d_{n,m}$ admits a left-perfect matching.*
>
> *(ii) If $c \geqslant \hat{c}(d) + \varepsilon$, then $G^d_{n,m}$ admits no left-perfect matching.*

[1] In the journal version [CS01] a different proof strategy is used, which is not directly applicable.
[2] This results assumes not fully random, but weaker $O(\log n)$-*wise independent* hash functions.

The theorem was proven independently and almost simultaneously by Frieze and Melsted [FM09, FM12] (type A), Fountoulakis and Panagiotou [FP10, FP12] (type B), and Dietzfelbinger et al. [DGM+10] (type B). The equality of the 2-core density thresholds for types A, B, and C, see Theorem 2.1, indicate that the theorem also holds for type C.

d	3	4	5	6	7	8	9
$\hat{c}(d)$	0.91794	0.97677	0.99244	0.99738	0.99906	0.99966	0.99988

Table 3.1.1.: Thresholds $\hat{c}(d)$ for the existence of left-perfect matchings in $G^d_{n,m}$, rounded to the nearest multiple of 10^{-5}.

According to Section 2.3.1, we have $\hat{c}(d) = \mathrm{key}(\lambda_1^{\mathrm{den}}, d)$, where λ_1^{den} is required to fulfill $\mathrm{dens}(\lambda_1^{\mathrm{den}}, d) = 1$. Now let

$$f(\lambda) = \lambda \cdot \left(1 - \frac{\lambda}{e^\lambda - 1}\right)^{-1} \quad \text{and}$$
$$\tilde{\mathrm{key}}(\lambda, d) = \frac{\mathrm{key}(\lambda, d)}{\mathrm{dens}(\lambda, d)} = \frac{1 - e^{-\lambda} - \lambda \cdot e^{-\lambda}}{(1 - e^{-\lambda})^d}\,.$$

Note that $f(\lambda)$ is strictly increasing for increasing $\lambda > 0$, $f(\lambda_1^{\mathrm{den}}) = d$, $\frac{\partial \tilde{\mathrm{key}}}{\partial \lambda}(\lambda_1^{\mathrm{den}}, d) = 0$, and $d - 1 < \lambda_1^{\mathrm{den}} < d$. From this, one can conclude that

$$d - \min\left\{1, d^2/(e^{d-1} - 1)\right\} < \lambda_1^{\mathrm{den}} < d\,,$$

which implies in combination with

$$\hat{c}(d) = \mathrm{key}(\lambda_1^{\mathrm{den}}, d) = \tilde{\mathrm{key}}\,(\lambda_1^{\mathrm{den}}, d) > 1 - e^{-\lambda_1^{\mathrm{den}}} - \lambda_1^{\mathrm{den}} \cdot e^{-\lambda_1^{\mathrm{den}}}$$

that the threshold function $\hat{c}(d)$ monotonically increases and converges to 1 exponentially fast in d, cf., [FP12, Section 1].

3.1.1.4. Space and Time Consumption

Let $\varepsilon \in (0, 1)$ and $m = (1 + \varepsilon) \cdot n$. Under the assumptions made in Section 2.2.2 concerning the space and time needs of the hash functions, we have that both data structures can be constructed a. a. s. with

space usage $(1 + \varepsilon) \cdot |S| \cdot \left(\lceil \log|U| \rceil + \lceil \log|V| \rceil\right)$ (dictionary),
$(1 + \varepsilon) \cdot |S| \cdot \lceil \log|U| \rceil$ (membership tester),

lookup time $O\big(\log(1/\varepsilon)\big)$.

Assuming that $\log|U| \gg \log|S|$, the space consumption is roughly a factor of $1 + \varepsilon$ away from the information theoretical minimum, see Section 3.3.1.

3.1.1.5. Linear Time Algorithms

Up to now we did not concern ourselves with the task of computing a left-perfect matching in $G^d_{n,m}$. There are a variety of matching algorithms with running time polynomial in n, but since we focus on constant left degree d, the sparsity of the graph allows to solve the problem actually in (expected) linear time, independent of the type (A, B, or C) of $G^d_{n,m}$.

Case $d = 2$ For the case $d = 2$, Pagh discussed how to determine a matching, in expected linear time, via a reduction to 2-SAT [Pag01b, Section 2.3], as shown in Algorithm 2. For this, each key $x \in S$ is associated with a binary random variable X that indicates which edge $e = \{x, h_X(x)\}$, for $X \in \{0, 1\}$, becomes a matching edge. Then a 2-SAT instance F is generated, in expected linear time, with clauses that encode legal choices of matching edges for pairs of distinct keys. If a satisfying assignment φ of F exists, it can be determined in linear time and defines a matching M according to $M = \big\{\{x, h_{\varphi(X)}(x)\} \mid x \in S\big\}$.

Algorithm 2: `matching_via_2-SAT`

```
Input   : Bipartite graph G = (L ∪ R, E) and functions h_0, h_1, where
          E = {{x, h_b(x)} | x ∈ L, b ∈ {0,1}}.
Output  : Left-perfect Matching M in G if it exists, otherwise ∅.
Require : Function solve_2-SAT(F), that returns a satisfying assignment φ if F is
          a satisfiable 2-SAT instance, and otherwise returns ∅.
F ← ⋀_{x∈L} (X_x ∨ ¬X_x);
foreach a ∈ [m] do
    foreach {x, y} ∈ (N({a}) choose 2) do
        if h_0(x) = h_0(y) then F ← F ∧ ( X_x ∨  Y_y);
        if h_0(x) = h_1(y) then F ← F ∧ ( X_x ∨ ¬Y_y);
        if h_1(x) = h_0(y) then F ← F ∧ (¬X_x ∨  Y_y);
        if h_1(x) = h_1(y) then F ← F ∧ (¬X_x ∨ ¬Y_y);
    end
end
M ← ∅;
φ ← solve_2-SAT(F);                                   // black-box solver
if φ ≠ ∅ then M ← {{x, h_{φ(X_x)}(x)} | x ∈ S};
return M;
```

Case $d \geqslant 3$ For standard d-ary cuckoo hashing, Fotakis et al. presented an augmenting path algorithm that determines a matching in $G^d_{n,m}$ in expected linear time for all $c \in (0.5, 1)$ under the condition that $d \geqslant 5 + 3 \cdot \ln(\frac{c}{1-c})$ [FPSS05, Theorem 1]. That is,

for fixed d the load factor must be sufficiently far away from the theoretical threshold value $\hat{c}(d)$. This gap we would like to close.

We hope to improve on this result using a new algorithm, called *generalized selfless algorithm*, derived from the *selfless algorithm* by Sanders [San04].

Remark. Meanwhile, Khosla proposed a new algorithm, called *local search allocation*, for determining a left-perfect matching in $G^d_{n,m}$ [Kho13], see also Section 1.1. This algorithm always finds a matching if the graph admits one and has linear running time with high probability if $c \leqslant \hat{c}(d) - \varepsilon$, for $d \geqslant 3$ and constant $\varepsilon > 0$.

The generalized selfless algorithm allows to attack the following more general problem. Given a hypergraph H and parameters k and ℓ. Assign each hyperedge of H to k of its nodes (pairwise distinct), with the constraint that the maximum number of assignments for each node must not exceed ℓ. This can be seen as directing each hyperedge to k of its nodes, such that the maximum indegree of each node is at most ℓ. Hence, we call a legal assignment of edges to nodes a (k, ℓ)-*orientation* of H.

Determining a $(1, 1)$-orientation of $H^d_{m,n}$ corresponds exactly to our problem of finding a left-perfect matching in $G^d_{n,m}$.

Remark. An orientation for $k \geqslant 2$ allows to redundantly store entries in a hash table. This can be used in order to build data structures that achieve a high failure resistance or support efficient parallelization of operations, see, e. g., [AP11, $3 \leqslant d = 2 \cdot k - 1$].

We will discuss the details of the algorithm and our results (all experimental) shortly. But first, we summarize a few facts concerning the orientability of $H^d_{m,n}$.

Orientability Results obtained by researchers in the last decade show that for constant parameters (d, k, ℓ) there is a sharp phase transition from the situation that there is a. a. s. a (k, ℓ)-orientation of the random hypergraph $H^d_{m,n}$ to the situation that there is a. a. s. no (k, ℓ)-orientation. We refer to the corresponding thresholds $\hat{c}_{k,\ell}$ as *orientation thresholds*, and define the corresponding *normalized load thresholds* as $\hat{c}_{k,\ell}/\ell$.

Remark. With respect to the basic scheme, assuming that each table cell has a capacity of ℓ entries, the *load factor* $c = n/m$ is the ratio of keys to table cells, the *normalized load factor* $c/\ell = n/(m \cdot \ell)$ is the quotient of keys and overall capacity, while the *capacity utilization* $c \cdot k/\ell = (n \cdot k)/(m \cdot \ell)$ is the total number of entries divided by the overall capacity.

For all sequences (d, k, ℓ) of parameters $d > k \geqslant 1$ and $\ell \geqslant 1$ the orientation thresholds are known. (They are derived for $H^d_{m,n}$ type B and A, but are expected to hold for type C too.) Beside the results mentioned above, for the cases $d = 2$, $k = 1$, $\ell = 1$, e. g., [PR04], and $d \geqslant 3$, $k = 1$, $\ell = 1$ [FM12, FP12, DGM+10], the main contributions are the following. The thresholds for $d = 2$, $k = 1$, $\ell \geqslant 2$ were found independently and simultaneously by Fernholz and Ramachandran [FR07] as well as Cain, Sanders, and Wormald [CSW07]. The thresholds for all $d \geqslant 3$, $k = 1$, $\ell \geqslant 1$ were derived

by Fountoulakis, Khosla, and Panagiotou [FKP11] filling the gap in a result by Gao and Wormald [GW10] that holds for large enough ℓ. And a unifying theorem that gives the thresholds for all $d > k \geqslant 1$, $\ell \geqslant 1$, with $\max\{d-k, \ell\} \geqslant 2$ was derived by Lelarge [Lel12a, Lel12b] improving upon the result by Gao and Wormald [GW10] which assumes that ℓ is a large constant depending on d and k. The remaining values for "extreme orientability", i. e., values $d \geqslant 3$, $d = k+1$, $\ell = 1$, were shown to be equivalent to the known thresholds for the appearance of the complex (or giant) component[3] by Loh and Pagh [LP12, Section 2].

For the case $d = k + 1 \geqslant 2$ and $\ell = 1$ the *orient* algorithm by Loh and Pagh finds a (k, ℓ)-orientation of $H^d_{m,n}$ (if such an orientation exists) and has linear running time w. h. p. if $c = n/m = (1-\varepsilon) \cdot \hat{c}_{k,\ell}(d)$ for any constant $\varepsilon \in (0, 0.5)$, see [LP12, Theorem 1.1 and Section 3.1].

Results

We propose a new algorithm, called *generalized selfless algorithm*, for determining orientations of (random) hypergraphs and present strong experimental evidence that for any sequence $(d, k, \ell) \in \mathbb{N}^3$ and sufficiently large n this algorithm is likely to determine a (k, ℓ)-orientation of $H^d_{m,n}$ if $c \leqslant \hat{c}_{k,\ell} - \varepsilon$, for an arbitrary constant $\varepsilon > 0$. In particular, this includes finding a left-perfect matching in $G^d_{n,m}$. Pseudocode is given as Algorithm 3. With proper implementation, linear running time on input $H^d_{m,n}$ is guaranteed.

Remark. The generalized selfless algorithm is a greedy algorithm based on the *selfless algorithm* by Sanders [San04]. More precisely, for input parameters $d = 2$ and $k = 1$ the Algorithm 3 corresponds to the selfless algorithm. It was proven by Cain, Sanders, and Wormald that for constant $\ell \geqslant 2$ the selfless algorithm a. a. s. finds a legal orientation of $H^d_{m,n}$ (type A and B) if $c \leqslant \hat{c}_{1,\ell}(2) - \varepsilon$, for an arbitrary constant $\varepsilon > 0$, see [CSW07, Theorems 2.2 and 2.3].

We summarize our findings in the following conjecture.

> Conjecture 3.1.1 (Generalized Selfless Algorithm). Let d, k, and ℓ be integer constants with $d > k \geqslant 1$, $\ell \geqslant 1$ and let $c = n/m$. Then for any constant $\varepsilon > 0$ a. a. s. the following holds. If $c \leqslant \hat{c}_{k,\ell} - \varepsilon$, then the generalized selfless algorithm applied to $H^d_{m,n}$ with parameters k, ℓ directs each edge of $H^d_{m,n}$ towards k pairwise distinct nodes, such that the maximum indegree of each node is at most ℓ.

It follows a high level description of Algorithm 3. Later, in combination with the presentation of our experimental results in Section 3.5, we will discuss an implementation that has running time linear in n. In order to describe the algorithm compactly, we use the following definitions:

[3] The existence and non-existence can be shown to occur not only a. a. s. but w. h. p..

ALGORITHM 3: `generalized_selfless`

Input : Undirected hypergraph $H = (V, E)$ with parameters $k \geqslant 1$ and $\ell \geqslant 1$.
Purpose : Direct each hyperedge from E to k pairwise distinct nodes from V, such that the maximum indegree of each node is at most ℓ.
Output : Either H with legal orientation, or $\emptyset$ if no such orientation is found.
Require : Functions $\mathrm{inc}\colon V \to [\ell + 1]$, $\mathrm{req}\colon E \to [k + 1]$, $E_{\mathrm{pot}}\colon V \to \mathcal{P}(E)$, and $V_{\mathrm{pot}}\colon E \to \mathcal{P}(V)$ as defined in Section 3.1.1.5.

$V' \leftarrow V$;
while $V' \neq \emptyset$ do
  $V' \leftarrow \{v \in V \mid |E_{\mathrm{pot}}(v)| > 0, \mathrm{inc}(v) < \ell\}$;
  randomly choose $v \in V'$ with smallest ($\hat{=}$highest) priority $\mathrm{prio}(v)$, according to

$$\mathrm{prio}(v) = \begin{cases} 0, & \text{if } |E_{\mathrm{pot}}(v)| + \mathrm{inc}(v) \leqslant \ell \\ \sum_{e \in E_{\mathrm{pot}}(v)} \frac{\mathrm{req}(e)}{|V_{\mathrm{pot}}(e)|} + \mathrm{inc}(v), & \text{otherwise} \end{cases};$$

  if $\mathrm{prio}(v) > \ell$ then return $\emptyset$; `// failure`
  if $\mathrm{prio}(v) = 0$ then
    direct all edges $e \in E_{\mathrm{pot}}(v)$ to v; `// selfless step`
  else
    randomly choose edge $e \in E_{\mathrm{pot}}(v)$ from those with smallest $|V_{\mathrm{pot}}(e)|$;
    direct e towards v;
  end
end
if $\exists e \in E\colon \mathrm{req}(e) > 0$ then return $\emptyset$; `// failure`
return H;

(i) The number of edges already directed towards a node v is denoted by $\mathrm{inc}(v)$. A node v is *saturated* if $\mathrm{inc}(v) = \ell$, otherwise it is *unsaturated*.

(ii) The number of nodes to which an edge e must still be directed is denoted by $\mathrm{req}(e)$; initially $\mathrm{req}(e) = k$. An edge e is *settled* if $\mathrm{req}(e) = 0$, otherwise it is *unsettled*.

(iii) The set of unsettled edges incident to a node v that are not directed towards v, called *potential edges*, is denoted by $E_{\mathrm{pot}}(v)$.

(iv) The set of unsaturated nodes to which an edge e can be directed, called *potential nodes*, is denoted by $V_{\mathrm{pot}}(e)$.

The algorithm works in rounds where in each round at least one edge is directed to a node. Consider a fixed round $i \geqslant 0$. Let V' be the set of unsaturated nodes that have potential edges. Each node v from V' is assigned a non-negative priority $\mathrm{prio}(v)$ as follows.

- ▷ The priority is 0 if the number of potential edges plus the number of incoming edges is at most ℓ, i. e., $|E_{\mathrm{pot}}(v)| + \mathrm{inc}(v) \leqslant \ell$.
- ▷ Otherwise the priority is the expected indegree of v assuming that in this round all potential edges of v would be directed uniformly at random. The probability that a potential edge e of v is directed towards v is $\mathrm{req}(e)/|V_{\mathrm{pot}}(e)|$, hence the expected indegree of v is

$$\sum_{e \in E_{\mathrm{pot}}(v)} \frac{\mathrm{req}(e)}{|V_{\mathrm{pot}}(e)|} + \mathrm{inc}(v)\,.$$

Among the nodes from V' a node v with smallest priority is chosen uniformly at random. If this node has a priority larger than ℓ, then the algorithm stops and returns $\emptyset$, which means failure. If v has priority 0, then the node "selflessly" takes all its potential edges upon himself, i. e., they are directed towards v. Otherwise a potential edge e of v with smallest number of potential nodes $|V_{\mathrm{pot}}(e)|$ is selected and directed towards v. As before ties are broken by randomization. If V' has no elements left and there is an unsettled edge, then the algorithm returns $\emptyset$. If V' is empty and all edges are settled, then the algorithm returns H.

The experimental results together with a more detailed description of the algorithm are presented in Section 3.4.

3.1.2. Irregular $\bar{d}$-Ary Cuckoo Hashing

If we aim for a given worst-case number of cell probes d for lookup, then we cannot do better than giving each left node exactly the same degree d. But what happens if we weaken this restriction slightly? Assume that we focus on bounding the number of

cell probes on average over all $x \in S$. For this we modify the basic scheme such that for each $x \in U$ a randomized algorithm determines some individual number of hash functions d_x, and change the sequence of addresses to

$$h(x) = \big(h_0(x), h_1(x), \ldots, h_{d_x-1}(x)\big) .$$

Hence, for each $x \in U$ the number of addresses becomes a random variable D_x that follows some probability mass function $\rho_x = \rho_{D_x}$ (on the natural numbers). Let Δ_x be the mean of D_x and let $\bar{d}$ be the average mean over all $x \in S$, i. e.,

$$\Delta_x = \mathrm{Exp}(D_x) = \sum_{d \in \mathbb{N}} d \cdot \rho_x(d) \text{ for all } x \in U,$$

$$\bar{d} = \frac{1}{n} \cdot \sum_{x \in S} \Delta_x .$$

We define $G^{\bar{d}}_{n,m} = G^{\bar{d}}_{n,m}\big((\rho_x)_{x \in S}\big)$ to be the bipartite graph representation of this new scheme, i. e., we apply the following substitution

$$G^{d}_{n,m} \rightsquigarrow G^{\bar{d}}_{n,m}\big((\rho_x)_{x \in S}\big) ,$$

where the uniform left degree d is replaced by the average expected left degree $\bar{d}$ produced by the sequence of probability mass functions $(\rho_x)_{x \in S}$. Analogously, we obtain the generalization for the hypergraph and matrix representations, according to

$$H^{d}_{m,n} \rightsquigarrow H^{\bar{d}}_{m,n}\big((\rho_x)_{x \in S}\big) \text{ and } \mathbf{M}^{d}_{n,m} \rightsquigarrow \mathbf{M}^{\bar{d}}_{n,m}\big((\rho_x)_{x \in S}\big) .$$

Now given $\bar{d}$ the question arises which sequence of probability mass functions $(\rho_x)_{x \in S}$ for the random variables $(D_x)_{x \in S}$ maximizes the probability that the random graph $G^{\bar{d}}_{n,m}\big((\rho_x)_{x \in S}\big)$ has a left-perfect matching. We call such a sequence *optimal*. Note that there must be some optimal sequence for compactness reasons.

Remark. Without any modification our (greedy heuristic) Algorithm 3 can be used to try to determine a left-perfect matching in $G^{\bar{d}}_{n,m}$.

3.1.2.1. Identical Distribution

A very natural setting is when all random variables $(D_x)_{x \in S}$ are independent and have the *same probability mass function*.

Remark. In this case the random algorithm that determines for each $x \in U$ the individual left degree can be realized with at most two additional independent and fully random hash functions, if one applies for example the *alias method* [Gen03, pages 133ff.].

For type B of $G^{\bar{d}}_{n,m}$ it is optimal if the degree of each node is concentrated around $\bar{d}$; that means that

$$\rho(\lfloor \bar{d} \rfloor) = 1 - (\bar{d} - \lfloor \bar{d} \rfloor), \text{ and } \rho(\lfloor \bar{d} \rfloor + 1) = \bar{d} - \lfloor \bar{d} \rfloor$$

is an *optimal probability mass function* [DGM+10, follows from Proposition 4]. Furthermore, Dietzfelbinger et al. [DGM+10] generalized Theorem 2.1 to determine values $\hat{c}(\bar{d})$ where a. a. s. the edge density of the 2-core of $H^{\bar{d}}_{m,n}\big((\rho_x)_{x\in S}\big)$ is below 1 if c is lower than $\hat{c}(\bar{d})$, and above 1 if c is greater than $\hat{c}(\bar{d})$. They applied these thresholds to obtain the following generalization of Theorem 3.2 concerning the probability of the existence of a left-perfect matching.

> Theorem 3.3 ($\bar{d}$-Ary Cuckoo Hashing, [DGM+10, Theorem 3])
> *Let $(\rho_x)_{x\in S}$ be an optimal sequence under the assumption of identical distributions with constant $\bar{d} > 2$ and let $c = n/m$. Then for any constant $\varepsilon > 0$ a. a. s. the following holds:*
>
> *(i) If $c \leqslant \hat{c}(\bar{d}) - \varepsilon$, then $G^{\bar{d}}_{n,m}\big((\rho_x)_{x\in S}\big)$ (type B) admits a left-perfect matching.*
>
> *(ii) If $c \geqslant \hat{c}(\bar{d}) + \varepsilon$, then $G^{\bar{d}}_{n,m}\big((\rho_x)_{x\in S}\big)$ (type B) admits no left-perfect matching.*

The proof concerning the thresholds $\hat{c}(\bar{d})$ for non-integral $\bar{d}$ is not elaborated in all details in [DGM+09, DGM+10]. One approach, derived from the integral case that is discussed in Section 2.3.1, is to study the generalized *key function*

$$\operatorname{key}(\lambda, \bar{d}) = \frac{\lambda}{\Big(\alpha \cdot \lfloor \bar{d} \rfloor \cdot \big(\Pr\left(\operatorname{Po}[\lambda] \geqslant 1\right)\big)^{\lfloor d \rfloor - 1} + (1-\alpha) \cdot \lceil \bar{d} \rceil \cdot \big(\Pr\left(\operatorname{Po}[\lambda] \geqslant 1\right)\big)^{\lceil d \rceil - 1}\Big)},$$

as well as the generalized *density function*

$$\operatorname{dens}(\lambda, \bar{d}) = \operatorname{key}(\lambda, \bar{d}) \cdot \frac{\alpha \cdot \Pr\left(\operatorname{Po}[\lambda] \geqslant 1\right)^{\lfloor \bar{d} \rfloor} + (1-\alpha) \cdot \Pr\left(\operatorname{Po}[\lambda] \geqslant 1\right)^{\lceil \bar{d} \rceil}}{\Pr\left(\operatorname{Po}[\lambda] \geqslant 2\right)},$$

for $\alpha = 1 - (\bar{d} - \lfloor \bar{d} \rfloor)$.

If the minimum degree $\lfloor \bar{d} \rfloor$ of each node is at least 3 then, as shown later in Section 4.1.2, the threshold for the *appearance* of a 2-core is $\min_{\lambda\in(0,\infty)} \operatorname{key}(\lambda, \bar{d})$. Hence, analogously to the definition of $\hat{c}(d)$ for integral d, one gets

$$\hat{c}(\bar{d}) \coloneqq \operatorname{key}(\lambda_1^{\text{den}}, \bar{d}),$$

where λ_1^{den} is the smallest value for which it holds that $\lambda_1^{\text{den}} \geqslant \arg\min_{\lambda\in(0,\infty)} \operatorname{key}(\lambda, \bar{d})$ and $\operatorname{dens}(\lambda_1^{\text{den}}, \bar{d}) = 1$.

The same approach can be used in the case that $\lfloor \bar{d} \rfloor = 2$. However, it appears that if $\bar{d} \in (2, 2.25]$, then we have $\arg\inf_{\lambda\in(0,\infty)} \operatorname{key}(\lambda, \bar{d}) = 0$ and there is either no or only a non-positive solution λ for $\operatorname{dens}(\lambda, \bar{d}) = 1$. Accordingly, we probably have

$$\hat{c}(\bar{d}) \coloneqq \lim_{\lambda\to 0} \operatorname{key}(\lambda, \bar{d}) = \frac{1}{2 \cdot \big(1 - (\bar{d} - \lfloor \bar{d} \rfloor)\big)} = \frac{1}{2 \cdot \alpha},$$

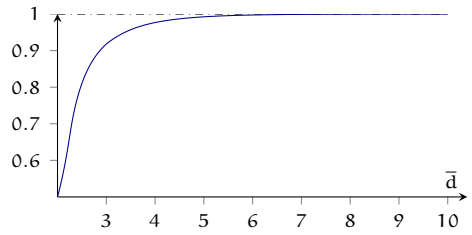


Figure 3.1.2.: Thresholds $\hat{c}(\bar{d})$ for the existence of left-perfect matchings in random bipartite graphs $G^{\bar{d}}_{n,m}\big((\rho_x)_{x\in S}\big)$ where all left degrees are concentrated around $\bar{d}$.

for all $2 < \bar{d} \leqslant 2.25$, cf., [DGM+09, end of Section 4.1]. The interpretation is that in this case, if c is below the threshold $\hat{c}(\bar{d})$, then a. a. s. the 2-core is empty and if c is above the threshold, then a. a. s. the 2-core is non-empty and has already a density of at least 1. Figure 3.1.2 depicts some thresholds $\hat{c}(\bar{d})$.

3.1.2.2. Almost Arbitrary Distributions

Beyond identically distributed random variables $(D_x)_{x\in S}$, in [DGM+10, Proposition 4] it was shown that if the sequence of *individual means* $(\Delta_x)_{x\in S}$ is fixed, then it is optimal (for type B) if the sequence of probability mass functions $(\rho_x)_{x\in S}$ concentrates the degree around the individual means $(\Delta_x)_{x\in S}$. However, it is not a priori clear, given $\bar{d}$, which sequence $(\Delta_x)_{x\in S}$ maximizes the probability of a left-perfect matching.

Results

! Under the assumption that the random variables $(D_x)_{x\in S}$ are independent we will show the following for $G^{\bar{d}}_{n,m}$ of type A.

For each $\bar{d} \geqslant 2$, there is an optimal sequence of probability mass functions that concentrates the degree of the left nodes around $\lfloor \bar{d} \rfloor$ and $\lceil \bar{d} \rceil$. Furthermore, if $\bar{d}$ is an integer we can explicitly determine this optimal sequence. In the case that $\bar{d}$ is non-integral we will identify a tight condition that an optimal sequence must meet.

> **Theorem 3.4 (Optimality of Concentration [DR12a, Theorem 1])**
> *Let $2 \leqslant n \leqslant m$, $2 \leqslant \bar{d}$, and let $(\rho_x)_{x\in S}$ be an optimal sequence for parameters $(n, m, \bar{d})$. Then the following holds for all $x \in S$:*
>
> *(i) If $\bar{d}$ is an integer, then $\rho_x(\bar{d}) = 1$.*
>
> *(ii) If $\bar{d}$ is non-integral, then $\rho_x(\lfloor \bar{d} \rfloor) \in [0, 1]$ and $\rho_x(\lceil \bar{d} \rceil) = 1 - \rho_x(\lfloor \bar{d} \rfloor)$.*

Note that Theorem 3.4 is not an asymptotic result, but holds for all n and m. The second statement is not entirely satisfying since it does not identify an optimal solution.

However, we will give strong evidence that in the situation of Theorem 3.4 (ii) there is no single, simple description of the distributions that are optimal, for arbitrary feasible node set sizes.

Since the case $\bar{d} = 2$ is completely settled by Theorem 3.4 (i), we then focus on the cases where $\bar{d} > 2$, with the additional condition that the number of left nodes is linear in the number of right nodes, that is, $n = c \cdot m$ for a constant $c > 0$. We show that for sufficiently large n all sequences that meet the condition of Theorem 3.4 (ii) asymptotically lead to the same matching probability. Therefore, we call these sequences *near optimal*.

> Proposition 3.1 (Asymptotic Behavior & Thresholds [DR12a, Prop. 1])
> *Let $(\rho_x)_{x\in S}$ be a near optimal sequence with constant $\bar{d} > 2$ and let $c = n/m$. Then for any constant $\varepsilon > 0$ a. a. s. the following holds:*
>
> *(i) If $c \leqslant \hat{c}(\bar{d}) - \varepsilon$, then $G_{n,m}^{\bar{d}}((\rho_x)_{x\in S})$ (type A) admits a left-perfect matching.*
>
> *(ii) If $c \geqslant \hat{c}(\bar{d}) + \varepsilon$, then $G_{n,m}^{\bar{d}}((\rho_x)_{x\in S})$ (type A) admits no left-perfect matching.*

This is a consequence of Theorem 3.3. Hence, the thresholds $\hat{c}(\bar{d})$ are exactly the same as defined above for $G_{n,m}^{\bar{d}}((\rho_x)_{x\in S})$ (type B).

So in the case that $n = c \cdot m$ all near optimal sequences exhibit essentially the same behavior in terms of matching probability, at least asymptotically. We will, however, give strong evidence that there are only two sequences that can be optimal, where the decision which one it is depends on the ratio c.

> Conjecture 3.1.2 (Essentially Two Strategies [DR12a, Conjecture 1]).
> Let $(\rho_x)_{x\in S}$ be an *optimal sequence* for parameters $(n, m, \bar{d})$ in the situation of Theorem 3.4 (ii) for constant $c = n/m > 0$ and constant $\bar{d} > 2$. Let $\alpha = \lceil \bar{d} \rceil - \bar{d}$.
>
> (i) If $c < \hat{c}(\bar{d})$, then we have $\rho_x(\lfloor \bar{d} \rfloor) = 1$ for $\alpha \cdot n$ nodes and $\rho_x(\lceil \bar{d} \rceil) = 1$ for $(1 - \alpha) \cdot n$ nodes (assuming that $\alpha \cdot n$ is an integer).
>
> (ii) If $c > \hat{c}(\bar{d})$, then we have $\rho_x(\lfloor \bar{d} \rfloor) = \alpha$ and $\rho_x(\lceil \bar{d} \rceil) = 1 - \alpha$ for all $x \in S$.

This means the following: If c is smaller than the threshold, i. e., $G_{n,m}^{\bar{d}}$ has a matching a. a. s., then it is optimal to fix the degrees of the left nodes; if c is larger than the threshold, then it is optimal to let each left node choose its degree at random from $\lfloor \bar{d} \rfloor$ and $\lceil \bar{d} \rceil$, by identical, independent experiments. Hence, the number of edges should be either fixed to $\bar{d} \cdot n$ a priori, or should be $\lfloor \bar{d} \rfloor \cdot X + \lceil \bar{d} \rceil \cdot (n - X)$ for a binomially distributed random variable X with sample size n and success probability $\lceil \bar{d} \rceil - \bar{d}$. The reasoning will be that seemingly if $c < \hat{c}(\bar{d})$, then the probability for a matching to exist as a function of the ratio of degree $\lfloor \bar{d} \rfloor$ nodes among all nodes is in essence concave (so a single value is better than all averages), while if $c > \hat{c}(\bar{d})$, then the matching probability is in essence convex (so an average over several values will be larger than any single value).

The proofs of Theorem 3.4 and Proposition 3.1 are given in Section 3.5 together with an in-depth discussion of Conjecture 3.1.2.

3.1.3. d-ary Cuckoo Hashing with Pages

Perhaps the most significant downside of both cuckoo hashing variants discussed so far is that they potentially require checking multiple cells distributed uniformly at random throughout the table. Hence, in settings where such random cell probes are expensive they become a less attractive choice to solve the membership and dictionary problem. One such standard setting is when memory is subdivided into (not too small) blocks of fixed size, called *pages*, and the dominating cost factor for lookup is the number of block accesses, which is about $(d+1)/2$ on average if we apply d-ary cuckoo hashing.

A natural approach to minimize this number is to split the key set S into disjoint subsets S_i of about the same size, using a fully random hash function, and then keep a separate cuckoo hash table in each page. This limits the number of pages examined to one and maintains the constant lookup time; such a scheme has been utilized in previous work, see, e. g., [ASA+09]. However, this approach has the following downsides:

(i) If the pages are small, say at most $\operatorname{poly}\log(m)$, then fluctuations in the size of the key sets S_i can significantly affect the maximum achievable load.

(ii) If the pages are large, more precisely of size m^δ for $\delta \in (0,1)$, then we obtain the same asymptotic load thresholds as in the setting without pages, but usually fail to model real-world page and memory sizes.

We generalize the above approach by modifying the basic scheme such that most of the d cell choices associated with a key x correspond to one single page, called *primary page*, and the rest of the cell choices (usually just one) correspond to a different page, called *backup page*. In detail, we use the following modification of the basic scheme. Let $d_b, d_p \in \mathbb{N}$ with $d = d_p + d_b$. The table t is subdivided into p pages (sub-tables) of equal size $\check{m} = \lfloor m/p \rfloor$. We have d hash functions $h_0, h_1, \dots, h_{d-1}$ with range restricted to $[\check{m}]$, i. e.,

$$h_i \colon U \to [\check{m}] \text{ for all } i \in [d].$$

The first d_p of them are used for the primary page and the last d_b of them are used for the backup page. In addition, we have two fully random hash functions

$$h_p, h_b \colon U \to [p],$$

in order to determine the address offset for the primary and for the backup page. Now the sequence of addresses is changed to

$$h(x) = \big(\underbrace{h_0(x)+i, h_1(x)+i, \dots, h_{d_p-1}+i}_{d_p \text{ addresses}}, \underbrace{h_{d_p}+j, h_{d_p+1}+j, \dots, h_{d-1}+j}_{d_b \text{ addresses}} \big),$$

$$\text{with } i = h_p(x) \cdot \check{m} \text{ and } j = h_b(x) \cdot \check{m}.$$

For type B we require in addition that the hash values are pairwise distinct according to

$$h_p(x) \neq h_b(x) \text{ for all } x \in U,$$
$$h_i(x) \neq h_j(x) \text{ for all } x \in U \text{ and } 0 \leqslant i < j < d_p, \text{ as well as } d_p \leqslant i < j < d.$$

Type C is realized analogously to type B via subdividing the pages further into disjoint sections. Note that in the special case of $\breve{m} = m$ and $d_b = 0$ we have exactly our basic scheme. As before, we adjust the graph, hypergraph, and matrix representations so that they reflect the modifications, denoted by the following substitutions

$$G^d_{n,m} \rightsquigarrow G^{d_p,d_b}_{n,(m,\breve{m})},\ H^d_{m,n} \rightsquigarrow H^{d_p,d_b}_{(m,\breve{m}),n}, \text{ and } \boldsymbol{M}^d_{n,m} \rightsquigarrow \boldsymbol{M}^{d_p,d_b}_{n,(m,\breve{m})}.$$

Related Paging Models Directly related to our model of cuckoo hashing with pages is the model *CUCKOO-CHOOSE-K* by Porat and Shalem [PS12, page 348]. Here each key is associated with $d_0 \cdot d_1$ random addresses, divided into d_0 groups of size d_1 each, where each group fully belongs to a page. That is, the model essentially corresponds to our model for $d_0 = 2$ (number of pages) and $d_1 = d_p = d_b$ (addresses per page). Porat and Shalem provide experimental results concerning the maximum load factor and they derive lower bounds on the maximum load factor for constant page size and parameters $(d_0 - 1) \cdot d_0 \cdot d_1 \geqslant 3$. However, their model is not designed for minimizing the number of page accesses which is our driving motivation.

For constant page size $\breve{m}$ our paging model is related to *blocked cuckoo hashing*, an extension of d-ary cuckoo hashing where the table consists of m blocks of ℓ cells each, each key is associated with d randomly chosen blocks, and the entry for each key can be stored in any of the ℓ cells of its associated blocks. There are two main variants: non-overlapping blocks and contiguous overlapping blocks.

In the case of *non-overlapping blocks*, see [DW05, DW07] and [Pan05, $\ell = 2$], we have $m \cdot \ell$ table cells that are *partitioned* into blocks of ℓ (contiguous) cells each. The construction of the data structure translates into the problem of finding an orientation of the random hypergraph $H^d_{m,n}$, with m nodes, which correspond to blocks, and n edges, which correspond to keys, such that the indegree of each node is at most ℓ. As we have discussed in Section 3.1.1.5, for constant d and ℓ as well as increasing $c = n/m$ there is a sharp phase transition from the situation that there is a. a. s. an orientation to the situation that there is a. a. s. no orientation; and the *orientation thresholds* $\hat{c}_{k,\ell} = \hat{c}_{1,\ell}$ are known. Recall that the *normalized load thresholds* are defined as $\hat{c}_{1,\ell}/\ell$, since each block can hold ℓ entries.

The model of m non-overlapping blocks of size ℓ and $d = 2$ hash functions corresponds to our paging model (type B) for the following parameters: m table cells of capacity ℓ entries each, divided into pages of size $\breve{m} = 1$, using one constant (hash) function to address the only table cell of each page, i. e., $d_p = d_b = 1$, or alternatively, $m \cdot \ell$ table cells, each of capacity 1 entry, divided into pages of size $\breve{m} = \ell$, using ℓ constant (hash) functions to address the ℓ table cells of each page, i. e., $d_p = d_b = \ell$.

A distinction of non-overlapping blocks and pages was made by Porat and Shalem in their model *CUCKOO-DISJOINT*, see [PS12, page 349]. Here we have $m \cdot \ell$ cells of capacity one, partitioned into m blocks of ℓ contiguous cells each, and each $\check{m}$ blocks build a separate page. Analogously to the model *CUCKOO-CHOOSE-K* described above, each key is associated with d_0 randomly chosen groups of d_1 contiguous cells, where each group fully belongs to a page, which is essentially the same as if each key is associate with d_0 randomly chosen pages and on each page the key is associated with one random block and its $d_1 = \ell$ cells. Therefore, independently of the page size, the maximum achievable load is the same as in the case of non-overlapping blocks of size ℓ without pages and $d = d_0$ hash functions.

Loosely related is the variant of *contiguous overlapping blocks*, which can be seen as a combination of cuckoo hashing and linear probing [Wei04, Section 3.2.1]. Here the analysis seems significantly harder compared to non-overlapping blocks and up to now only lower bounds for load thresholds are known, see [LP09] and [Bey12, $\ell = 2$]. Porat and Shalem proposed the model *CUCKOO-OVERLAP*, analogously to their two other model described above, as a combination of contiguous overlapping blocks with paging, where each block is entirely contained in a page [PS12, page 348]; they provide experimental results concerning the maximum achievable load as a function of the page size.

Results

! We focus on random bipartite graphs $G^{d_p,d_b}_{n,(m,\check{m})}$ type B with $d = d_p + d_b = 4$. All of the following results are experimental.

With small page size $\check{m}$ and $d_p = 4$, $d_b = 0$, i. e., each key confined to one page, we find that the maximum achievable load factor is far below $\hat{c}(4)$. However, if we set $d_p = 3$ and $d_b = 1$, i. e., each key is given three addresses on a primary page and a fourth on a backup page, then the load factor is quite close to $\hat{c}(4)$. The load factor does not shrink if we set $d_p = d_b = 2$. Intuitively, a second page for each key allows overloaded pages to constructively distribute load to underloaded pages. Backed up by additional experiments for different d, we believe in general the following is true.

> Conjecture 3.1.3 (Cuckoo Hashing with Pages [DMR11a, implicit]). Let $d_p, d_b \in \mathbb{N}$ be non-zero constants with $d_p + d_b \geqslant 3$ and let $c = n/m$ as well as $\check{m} = \operatorname{poly}\log(m)$. Then for any constant $\varepsilon > 0$ a. a. s. the following holds:
>
> (i) If $c \leqslant \hat{c}(d) - \varepsilon$, then $G^{d_p,d_b}_{n,(m,\check{m})}$ (type B) admits a left-perfect matching.
>
> (ii) If $c \geqslant \hat{c}(d) + \varepsilon$, then $G^{d_p,d_b}_{n,(m,\check{m})}$ (type B) admits no left-perfect matching.

So, in order to obtain a reasonable load factor, we must access two pages for lookup in the worst-case. We distinguish between keys that use their primary page, called

primary keys, and keys that use their backup page, called *backup keys*.

Maximizing the number of primary keys can be done straightforward using a minimum *weighted matching* algorithm. We present a simple and efficient variant based on the *Hopcroft-Karp algorithm* [HK71, HK73]. Applying this algorithm, we find that with $d_p = 3$, $d_b = 1$, and a load factor of 0.95 only about 3 percent of keys are backup keys (with suitable page sizes). That is, for most of the keys a lookup requires only a single page access.

Moreover, we show that a simple variation of the well-known *random walk* insertion procedure, see, e. g., [FMM11, Algorithm 2] and [FPS10, Section 2.1], allows nearly the same level of performance with online placement of keys (including scenarios with alternating insertion and deletions). Our experiments consistently yield that at most 5 percent of keys needs to be placed on a backup page with these parameters. This provides a tremendous reduction of the number of page accesses required for successful searches.

For unsuccessful searches, spending a little more space for Bloom filters [Blo70] on each page leads to an even smaller number of accesses to backup pages. This is done via storing for each page all the keys from S in a separate Bloom filter that have this page as primary page but are placed in their backup pages. An unsuccessful search for a key $x \in U - S$ will access the backup page of x only in the case that the Bloom filter associated with the primary page of x confirms the set membership (false positive answer).

The experimental results and the algorithms for static and dynamic insertion are presented in Section 3.6.

3.1.4. Overview of the Chapter

The next section gives some background information on hashing and load balancing. It follows Section 3.3 which covers the standard space lower bounds for dictionary and membership tester, as well as some facts about matchings and weighted matchings in bipartite graphs since this is needed for the discussion of cuckoo hashing with pages. In the subsequent three sections we focus on our results, where Section 3.4 treats the generalized selfless algorithm, Section 3.5 covers irregular cuckoo hashing, and Section 3.6 deals with cuckoo hashing with pages. The chapter finishes with some concluding remarks.

3.2. Further Background and Related Work

We now give some background on hash tables and load balancing, which however, is not required to follow the rest of the chapter. A reader familiar with these topics can skip this section.

The dictionary and membership tester variants described in Section 3.1 are representatives of the most common hashing-based structures, known as *hash tables*. Basically, a hash table consists of a table $\mathbf{t}$ with m cells and a mapping h

$$h\colon U \to \bigcup_{i=1}^{<\infty} [m]^i \ ,$$

based on one hash function h or several hash functions $h_0, h_1, h_2, \ldots$, where each key $x \in S$ is stored in a cell t_a whose address a is from the sequence $h(x)$, in the case of a dictionary together with satellite data $g(x)$. Basically, hash tables differ in their mapping as well as in the capacity and structure of the table cells. Moreover, often one considers the dynamic case, that is S and g change over time, which makes strategies for dynamically *inserting*, *updating*, and *deleting* entries main distinctive factors.

! For the rest of Section 3.2 the following holds. The statements with respect to the performance of the hash tables are valid under the assumption that the hash functions h and h_i, $i \geqslant 0$, are fully random; and events that depend on n occur asymptotically almost surely.

3.2.1. Traditional Hash Tables

A rough classification concerning the table $\mathbf{t}$ itself distinguishes between *open hashing*, which means that the number of stored entries is not limited by the table size, and *closed hashing*, which means that each cell can hold a constant number of entries. Typical representatives are

separate chaining [Knu98, page 520]. This hash table scheme uses the mapping

$$h\colon U \to [m],\ h(x) \coloneqq \big(h(x)\big) \ .$$

Each table cell holds a pointer to a linked list of entries and a key is inserted at the front of the list, whose pointer is stored in the cell with address $h(x)$.

linear probing [Knu98, page 526]. Here each cell can store one entry and we have

$$h\colon U \to [m],\ h(x) \coloneqq \big(h(x), h(x) \oplus 1, h(x) \oplus 2, \ldots, h(x) \oplus (m-1)\big) \ ,$$

were $\oplus$ is addition in $\mathbb{Z}_m$. Insert and lookup follow the probe sequence given by $h(x)$ until x or an empty cell is found.

The classical hash table schemes like separate chaining, linear probing, quadratic probing, and variants, see, e. g., [CLRS01, pages 224ff.] and [Knu98, pages 513ff.], have in common that lookups cannot guaranteed to be in constant time. Actually, for each hash table where all probe sequences start at some random position and the whole

search is solely determined by this starting value, one needs to test at least $\Omega(\frac{\log n}{\log \log n})$ entries for lookup in the worst case, independent of the concrete insertion procedure. This lower bound is a consequence of the result that if one randomly throws n balls into n bins, then the maximum number of balls that fall into a bin is $\frac{\ln n}{\ln \ln n} \cdot (1 + o(1))$, see, e. g., [Gon81, Theorem 6 and Table VI] and [RS98, Section 1.1 and Theorem 1].

3.2.2. Multiple Choice Hash Tables

According to [Mit09, Introduction], in the recent years "[...] the field of hashing, which has enjoyed a long and rich history in computer science [...], has also enjoyed something of a theoretical renaissance [...]", based on what is called the *power of multiple choices*, see, e. g., [Mit91].

This process was triggered by the surprising insight that if one throws n balls into n bins where each ball is associated with two random bins and is placed in the least loaded one, then the maximum load of a bin is exponentially reduced in comparison with the situation where each ball has only one random choice. The concrete maximum load depends on the tie-breaking rule for equally loaded bins. If ties are broken uniformly at random, then the maximum load drops to $\frac{\ln \ln n}{\ln 2} + \Theta(1)$ [ABKU94], [ABKU99, Theorem 1.1]. If one uses a deterministic and asymmetric tie-breaking rule, the maximum load can be reduced even further, namely to $\frac{\ln \ln n}{2 \cdot \ln 1.618} + O(1)$ [Vö99], [Vö03, Theorem 1]. This was the starting point for the design of new hash tables that can be subsumed under the term *multiple choice hash tables*, see, e. g., [KM08], which have the advantages of

- ▷ worst-case constant lookup time,
- ▷ high space efficiency, and
- ▷ good parallelizability.

The results on two random choices naturally translate into a hash table that extends separate chaining, known as

2-way chaining [ABKU99, Section 5], with improvement from [BM01, Section II, 2-left scheme]. Analogously to separate chaining, each table cell holds a pointer to a linked list of entries, but now a key is associated with two lists according to

$$h_0, h_1 : U \to [\lfloor m/2 \rfloor], \; h(x) := \big(h_0(x), h_1(x) + \lfloor m/2 \rfloor\big) .$$

An entry that is associated with a key x is stored at the front of the shortest list given by $h_0(x)$ and $h_1(x)$, and if both lists have the same length, then either ties are broken[4] uniformly at random (symmetric random rule) or one always uses the list given by $h_0(x)$ (asymmetric deterministic rule). The lookup procedure searches the two lists in alternating order.

[4] First *2-way chaining* was defined to break ties randomly [ABKU94], later deterministically, see, e. g., [BM01, PR01].

With 2-way chaining, a lookup only needs to test $\Theta(\log\log n)$ entries in the worst case, if the load factor $c = n/m$ is constant. Increasing the number of choices per key further leads to

d-*way chaining* [BM01, Section II], [MV99, Section 3], [MV02]. A generalization of 2-way chaining, where each key has $d \geqslant 2$ independent random choices and the mapping is defined as

$$h_0, h_1, \ldots, h_{d-1} \colon U \to [\lfloor m/d \rfloor],\ h(x) \coloneqq \big(h_i(x) + i \cdot \lfloor m/d \rfloor\big)_{i \in [d]} .$$

The asymmetric tie-breaking rule prefers the list given by the hash function with the smallest index.

The step from 2 to 3 choices or more decreases the maximum load only by a small factor, since in general for $d \geqslant 2$ and $m = n$, the maximum load is $\frac{\ln\ln n}{\ln d} + O(1)$ [ABKU99, Theorem 1.1] (symmetric random rule) or smaller than $\frac{\ln\ln n}{(d-1)\cdot\ln 2}$ [Vö03, Theorem 1] (asymmetric deterministic rule), respectively.

In the static case, that is if S is given in advance and one determines an optimal assignment from keys to table cells, then the maximum load can be even reduced to a constant, see, e. g. [ABKU99, follows by Lemma 6.1]. More precisely, one only needs cells of capacity one if each key has $d = 2$ two random choices and the load factor c is at most $1/2 - \varepsilon$ for an arbitrary constant $\varepsilon > 0$, see, e. g., [Pag01b, Section 2.3][5], [DM03, Theorem 1], and very precise [DK12, Theorems 1 and 4].

Remark. More general results give an upper bound for the smallest maximum load of $\lceil n/m \rceil + 1$ in the case of an arbitrary load factor c [SEK00, Theorem 3.1], [SEK03, Theorem 4], and of $\lceil n/m \rceil$ if c is bounded from below by $\Omega(\log m)$ [CRS03, Theorem 2].

In order to achieve two probes for lookup also in the dynamic case while maintaining a constant load factor just under $1/2$, one has to apply an insertion procedure that moves entries around the table. A straightforward solution, first suggested for a related hashing scheme, is to solve a matching (or network flow) problem on each insertion [Riv78, GM79, Section 4]. However, a more practical and very prominent approach is a variant of the *last-come-first-served* strategy [PM89], which gave the resulting hash table the unique name

cuckoo hashing [PR01, PR04]. Like 2-way chaining we use the mapping

$$h_0, h_1 \colon U \to [\lfloor m/2 \rfloor],\ h(x) \coloneqq \big(h_0(x), h_1(x) + \lfloor m/2 \rfloor\big) ,$$

but now each cell can hold only one entry. The insertion procedure imitates the nesting habits of a cuckoo. For simplicity assume that entries are keys only. On insertion of a key x the cell given by $h_0(x)$ is probed. If the cell is empty, then x is stored there and insertion terminates. Otherwise, x displaces the previous stored key x', which now must be inserted in the same way as x, but in the cell given by $h_1(x')$, and so on.

[5] This result assumes not fully random, but weaker $O(\log n)$-*wise independent* hash functions.

Cuckoo hashing naturally extends to more than two hash functions, which leads to

d-ary cuckoo hashing [FPSS03, FPSS05]. This variant of cuckoo hashing uses $d \geqslant 2$ hash functions, and the mapping generalizes to[6]

$$h_0, h_1, \ldots, h_{d-1} \colon U \to [m], \ h(x) \coloneqq \big(h_0(x), h_1(x), \ldots, h_{d-1}(x)\big) .$$

We discussed d-ary cuckoo hashing in detail in Section 3.1 including variants that allow the number of hash functions depend on the key, or subdivide the table into pages or blocks. Other notable variants augment the data structure with a small auxiliary memory, called *stash*, or with a *queue*. The stash is used to increase the probability of a successful construction [KMW08, KMW09, Kut10, Aum10, ADW12, ADW13], where the queue in addition is used to maintain a dynamic insertion procedure that needs constant time in the worst-case [KM07, ANS09, ANS10].

3.2.3. External Memory Hash Tables

Hashing is well suited for realizing large data structures that have to be stored in external memory. In a standard model [AV88] it is assumed that the external memory is accessed in the form of pages, each page spanning $\check{m}$ entries (records), and the internal memory can hold at least one page. Time complexity is primarily determined by the number of page I/Os, i. e., write and read accesses, since in many cases they are assumed to dominate the time for the other (memory word) operations.

We consider the situation that the m cells of an external memory hash table are partitioned into $p = m/\check{m}$ pages. Each cell has capacity of one entry, which we assume to be a key (membership) or a key-value pair (dictionary). If n entries are stored, the load factor is $c = n/m = n/(p \cdot \check{m})$, where in the case of open hashing, see Section 3.2.1, the total number of pages used can be larger than p. Hash functions are ideal, i. e., in particular, their internal memory consumption is ignored.

Traditional Hash Tables Traditional hash tables like *linear probing* and *separate chaining* can be easily extended to this setting. The adjustments are as follows:

external separate chaining [Knu98, page 542]. This hash table scheme maps keys to pages via

$$h \colon U \to [m/\check{m}], \ h(x) \coloneqq \big(h(x)\big) .$$

Each of the $p = m/\check{m}$ pages is the start of a separate linked list of additional pages and a key is stored in the list with address $h(x)$. For c bounded away by a constant from 1, most of the lists consist only of one page. Hence, in this case the total number of pages is close to p.

[6] Cuckoo hashing was defined using pairwise distinct hash values (type C), whereas d-ary cuckoo hashing was first defined in a simplified variant allowing duplicate hash values (type A).

external linear probing [Knu98, page 543]. Here the probe sequence for a key goes linearly from page to page according to

$$h\colon U \to [m/\breve{m}],\ h(x) \coloneqq \big(h(x), h(x) \oplus 1, h(x) \oplus 2, \ldots, h(x) \oplus (p-1)\big)\ ,$$

where $\oplus$ is addition in $\mathbb{Z}_p$. Insert and lookup follow $h(x)$ until a non-empty page or a page with an entry corresponding to x is found.

Knuth showed that both variants work well, see [Knu98, pages 541ff.] and, e. g., [Pag03, page 28]. More precisely, with load factor $c = 1-\varepsilon$, for constant $\varepsilon > 0$, for both schemes the expected average number of page accesses for a lookup is $1+2^{-\Omega(\breve{m})}$. This requires internal memory of $O(1)$ pages only. Interestingly, if linear probing is not adjusted to paging, i. e., one essentially ignores pages, then the expected average number of page access for lookup is $1 + O(c/\breve{m})$, see [QM98, Section 6] and [PWYZ14, Theorem 1].

Combining a variant of chained hashing with new techniques, Jensen and Pagh [JP08] showed how to obtain an asymptotically optimal external memory hash table that utilizes internal memory in the size of $O(1)$ pages. Here for a load factor of $c = 1-O(\breve{m}^{-1/2})$ the expected number of page accesses is $1+O(\breve{m}^{-1/2})$ for successful and unsuccessful lookup, while insert, delete, and update need $1 + O(\breve{m}^{-1/2})$ page accesses amortized, in expectation. However, in all of the hash tables discussed so far there is always a small chance that a lookup needs two or more page accesses.

A worst case guarantee of one page access for lookup can be achieved at the price of additional internal memory depending on n, see, e. g., [GL88, Section 4] and [Lar88, Section 2] for schemes that work for constant $c = c(\breve{m}) < 1$ and need additional space of $\Theta(n/\breve{m})$, not including the space consumption for helper hash functions.

Remark. Schemes that directly map at most $\breve{m}$ keys to a single page realize a generalization of an injective function, see Section 4.2.1, in our context usually called $\breve{m}$-perfect hash function. For $c = 1$, i. e., $n = \breve{m}\cdot p$, Mairson proved that if $\breve{m}$ is constant, then at least $\Omega(n \cdot \log(\breve{m})/\breve{m})$ bits are necessary in the worst case[7] in order to build a $\breve{m}$-perfect hash function with such a minimal range [Mai83, Theorem 5].

Multiple Choice Hash Tables With external memory versions of multiple choice hash tables one can achieve two page accesses for lookup in the worst case. This can be realized without storing additional data structures that need internal memory that grows with n if one uses, e. g., a random walk insertion procedure.

For instance, *blocked cuckoo hashing* with non-overlapping blocks, see Section 3.1.3, can be directly applied to the external memory model using blocks of size $\ell = \breve{m}$ that correspond to pages. With $d = 2$ hash functions the lookup time is limited to at most two page access for each $x \in U$, and the load factor c is bounded by $\hat{c}_{1,\ell}(2)/\breve{m} - \varepsilon$, for constant $\varepsilon > 0$. This bound can be chosen arbitrary close to one using a large enough

[7] This is also a lower bound for $c = n/m < 1$, under the condition that $n/\breve{m}$ out of $m/\breve{m}$ pages get exactly $\breve{m}$ keys [Mai92].

but constant page size $\breve{m}$. Moreover, with a modified insertion procedure, the expected number of page accesses for a successful lookup can be made close to 1. This approach is discussed in Section 3.6.4.3.

As introduced in Section 3.1.3, another variant of cuckoo hashing, called d-*ary cuckoo hashing with pages*, also allows to realize an external memory hash table that guarantees at most two page accesses per lookup; and with a slight adjustment of the insertion procedure, achieves close to 1 page access for successful lookup in expectation. For this scheme experiments show that the load factor is more dependent on d, the number of hash functions used to address table cells inside pages, than on the page size $\breve{m}$, and, furthermore, can be made close to $\hat{c}(d)$. Since the potential positions of a key within a page are determined via hashing, searching for a key once its page is loaded in internal memory is fast. This can be advantageous in the case of larger pages. Cuckoo hashing with pages is discussed in detail in Section 3.6.

3.3. Basics

In this section we give the standard space lower bounds for dictionary and membership. Afterwards, we discuss how to determine (minimum weight) left-perfect matchings using augmenting path algorithms, which is required for Section 3.6.

3.3.1. Worst-Case Space Lower Bounds

In Section 3.1.1.4 we stated that, if one ignores the hash functions, then the space consumption of d-ary cuckoo hashing is $(1+\varepsilon)\cdot|S|\cdot\log|U|$ for a membership tester and $(1+\varepsilon)\cdot|S|\cdot(\log|U|+\log|V|)$ for a dictionary, where $\varepsilon>0$ is an arbitrary constant. If the universe U is large, then this is near optimal according to the following standard lower bound.

> Lemma 3.3.1 (Space Bounds for Dictionary and Membership). In order to solve membership for the keys from $S\subseteq U$ one needs at least $|S|\cdot(\log|U|-\log|S|)$ bits in the worst-case. For solving dictionary when all keys from S are associated with a value from V one needs at least $|S|\cdot\log|V|$ bits in addition in the worst-case.

Proof. The following is folklore.

Let $|U|=u$ and let $|S|=n$. For a binary encoding of the set S on needs at least $\log\left(\binom{u}{n}\right)$ bits in the worst-case, since there are $s=\binom{u}{n}$ sets of size n and any uniquely decodable binary code for these sets must have a codeword of length at least $\log(s)$, since otherwise the number of codewords would be too small to encode all sets. Similarly, one needs at least $n\cdot\log|V|$ bits in the worst-case to encode one of the possible $|V|^n$ sequences of values from V. Hence, it remains to show that $\log(s)\geqslant n\cdot(\log u-\log n)$.

In order to bound $\log(s)$ from below we use that

$$\log\left(\binom{u}{n}\right) = \log\left(\prod_{i=0}^{n-1}\frac{u-i}{n-i}\right) = \sum_{i=0}^{n-1}\log\left(\frac{u-i}{n-i}\right).$$

Since the sequence $\log\left(\frac{u-i}{n-i}\right), 0 \leqslant i \leqslant n-1$, is monotonically increasing for $u > n$, we can derive a lower bound via straightforward integration as follows:

$$\begin{aligned}
\log\left(\binom{u}{n}\right) &= \log\left(\frac{u}{n}\right) + \sum_{i=1}^{n-1}\log\left(\frac{u-i}{n-i}\right) \geqslant \log\left(\frac{u}{n}\right) + \int_0^{n-1}\log\left(\frac{u-x}{n-x}\right)dx \\
&= \log\left(\frac{u}{n}\right) + \left[(u-n)\cdot\log\left(\frac{u-n}{n-x}\right) - (u-x)\cdot\log\left(\frac{u-x}{n-x}\right)\right]_0^{n-1} \\
&= n\cdot(\log u - \log n) + \underbrace{(u-n+1)\cdot\log\left(\frac{u}{u-n+1}\right) - \log n}_{=:T}.
\end{aligned}$$

The summand T is non-negative, since using Bernoulli's inequality we get

$$T = \log\left(\left(1+\tfrac{n-1}{u-n+1}\right)^{u-n+1}\right) - \log n \geqslant \log\left(1+\tfrac{(u-n+1)\cdot(n-1)}{u-n+1}\right) - \log n = 0.$$

Hence, the lemma follows. ■

3.3.2. Left-Perfect Matchings

A necessary and sufficient condition for the existence of a left-perfect matching in a bipartite graph $G = (L \cup R, E)$ is given by the marriage condition (MC), which states that each non-empty set of left nodes must have a neighborhood of at least equal size. This is extremely clearly in the case that G is 1-left-regular like $G_{n,m}^d$ for $d = 1$. But as we have mentioned in Section 3.1.1.3, such a graph is not very interesting for us because of the following well known property.

> Lemma 3.3.2 ("Birthday Paradox"). Let $d = 1$ and let $n(m) = \omega(\sqrt{m})$. Then w. h. p. $G_{n,m}^d$ admits no left-perfect matching.

Proof. The following is folklore.

The probability that $G_{n,m}^d$ admits a left-perfect matching is exactly

$$\frac{m^{\underline{n}}}{m^n} = \exp\left(\sum_{i=1}^{n-1}\ln\left(1-\frac{i}{m}\right)\right) =: p(n,m).$$

Since $f(x) = 1 - \frac{x}{m}$ is monotonically decreasing for $0 \leqslant x \leqslant n-1 < m$, where $x \in \mathbb{R}$, we have that the probability p can be bounded from above via

$$p(n,m) \leqslant \exp\left(\int_0^{n-1}\ln\left(1-\frac{x}{m}\right)dx\right) =: P(n,m).$$

Due to integration by parts and using that $\frac{x}{m-x} = -1 + \frac{m}{m-x}$, we get

$$\begin{aligned} P(n,m) &= \exp\left(\left[x \cdot \ln\left(1-\frac{x}{m}\right) - x - m \cdot \ln(m-x)\right]_0^{n-1}\right) \\ &= \exp\Bigg(\underbrace{(n-1)\cdot \ln\left(1-\frac{n-1}{m}\right)}_{=:P_0(n,m)} + \underbrace{m \cdot \ln\left(\frac{m}{m-(n-1)}\right) - (n-1)}_{=:P_1(n,m)}\Bigg). \end{aligned}$$

Let $n = \sqrt{z \cdot m}$ for $z \geqslant 1$. By substituting m with n^2/z, we determine the limits

$$\begin{aligned} \lim_{n\to\infty} P_0(n, n^2/z) &= \lim_{n\to\infty} \ln\left(1-\frac{n-1}{n^2/z}\right)^{n-1} = -z \\ \lim_{n\to\infty} P_1(n, n^2/z) &= \lim_{n\to\infty} \frac{\ln\left(\frac{n^2/z}{n^2/z-(n-1)}\right) - \frac{n-1}{n^2/z}}{\frac{1}{n^2/z}} = \frac{z}{2}. \end{aligned}$$

Therefore, we have that $\lim_{n\to\infty} P(n, n^2/z) = e^{-\frac{z}{2}}$, which implies the lemma. ■

With turning to bipartite graphs of larger left degree the determination of a left-perfect matching is no longer trivial but remains relatively easy. A standard method, not only for bipartite but for general graphs, is to start with an empty matching M and then iteratively find an augmenting path P with respect to M and replace M by the symmetric difference $P \oplus M$. Here and in the following, a path in this context is considered as set of edges. The algorithm stops, if there is no augmenting path with respect to M. It is a classical result by Berge that at this point M has the largest possible size.

> Theorem 3.5 (Maximum Cardinality Matching [Ber57, Theorem 1])
> *A matching M in a graph G is a matching of maximum cardinality if and only if G has no augmenting path with respect to M.*

Since in a bipartite graph G each edge is incident to a left node, a maximum cardinality matching in G has also the maximum number of matched left nodes.

3.3.3. Minimum Weight Left-Perfect Matchings

In Section 3.6 we will consider matchings in weighted bipartite graphs, that is graphs $G = (L \cup R, E)$ with an additional *weight function*

$$\omega\colon E \to \mathbb{R}.$$

Our aim is to determine matchings with maximum cardinality that have minimum weight, where we define the weight of any edge set $E' \subseteq E$ as

$$\omega^+(E') \coloneqq \sum_{e\in E'} \omega(e).$$

In order to obtain a *minimum weight maximum cardinality matching* one can use the augmenting path approach discussed above. Given a matching M one chooses an augmenting path P with respect to M that minimizes the weight $\omega^+(M \oplus P)$, or with other words, the *incremental weight* of P with respect to M, defined as

$$\omega^{\pm}_M(P) \coloneqq \omega^+(P - M) - \omega^+(P \cap M) = \omega^+(M \oplus P) - \omega^+(M)\,,$$

must be minimal. This is subsumed in the following standard lemma.

LEMMA 3.3.3 (OPTIMALITY OF MINIMUM ADDITIONAL WEIGHT, E. G., [KOZ91, LEMMA 19.8]). Let M be a minimum weight matching of size $|M|$ and let P be an augmenting path for M of minimum incremental weight among all augmenting paths for M. Then the matching $M \oplus P$ is a minimum weight matching of size $|M| + 1$.

On that basis, Algorithm 4 shows a simple method to determine a maximum cardinality matching of minimum weight. Now a minor observation, which will become useful for

ALGORITHM 4: `min_weight_max_card_matching`

Input : Bipartite graph $G = (V, E)$, and weight function $\omega\colon E \to \mathbb{R}$.
Output : Maximum cardinality matching $M \subseteq E$ of minimum weight.
Require : Function `min_add_weight_aug_path(G, M)` that returns an augmenting path P in G with respect to M that has minimum incremental weight $\omega^{\pm}_M(P)$ if it exists, and otherwise returns $\emptyset$.

$M \leftarrow \emptyset$; $P \leftarrow \emptyset$;
repeat
  $P \leftarrow$ `min_add_weight_aug_path(G, M)`;
  $M \leftarrow M \oplus P$;
until $P = \emptyset$;
return M;

us, is that with this strategy of choosing augmenting paths the incremental weight for bipartite graphs can only increase with each round.

LEMMA 3.3.4 (INCREASING ADDITIONAL WEIGHT). Let G be a bipartite graph and let M_i be the matching M in Algorithm 4 on input G at the point where $|M| = i$, and let P_i be the augmenting path for M_i. Furthermore, let s be the size of the resulting matching. Then the following holds:

$$\omega^{\pm}_{M_0}(P_0) \leqslant \omega^{\pm}_{M_1}(P_1) \leqslant \omega^{\pm}_{M_2}(P_2) \leqslant \ldots \leqslant \omega^{\pm}_{M_{s-1}}(P_{s-1})\,.$$

PROOF. Assume for a contradiction we have $\omega^{\pm}_{M_{i+1}}(P_{i+1}) < \omega^{\pm}_{M_i}(P_i)$ for fixed $i \geqslant 0$.

Recall that P_i is an augmenting path with respect to M_i and P_{i+1} is an augmenting path with respect to $M_{i+1} = M_i \oplus P_i$. There are two cases.

Case (i): P_{i+1} is also an augmenting path with respect to M_i. It follows that P_i and P_{i+1} must be node disjoint and hence $\omega^{\pm}_{M_i}(P_{i+1}) = \omega^{\pm}_{M_{i+1}}(P_{i+1}) < \omega^{\pm}_{M_i}(P_i)$. But then in round i the algorithm would have chosen P_{i+1} instead of P_i, a contradiction.

Case (ii): P_{i+1} is not an augmenting path with respect to M_i and becomes an augmenting path for M_{i+1} via switching the edges of P_i from matched to unmatched and vice versa.

For ease of discussion, we define G_j to be the directed bipartite graph that results from G by orienting each matching edge of M_j from right to left and each non-matching edge from left to right for all $j \in [s]$. So in G_i the path P_i is directed from left to right, starting at an unmatched left node v_0 and ending at an unmatched right node v_t. Let $v_0, v_1, \ldots, v_t$ be the sequence of nodes of P_i in G_i from left to right. In G_{i+1} the path P_i is directed from right to left starting at v_t and ending at v_0.

Since P_i and P_{i+1} are not node disjoint, it is possible that there exist cycles C in the undirected graph G such that $C = AB$ and A is subset of $P_i - P_{i+1}$ and B is a subset of $P_{i+1} - P_i$. For each such cycle there are two possibilities. Either C is a directed (and hence an alternating) cycle in G_i, called forward cycle, or C is a directed cycle in G_{i+1}, called backward cycle. In both cases C has even length.

Now consider G_{i+1}. We follow P_{i+1} from its unmatched left node to its unmatched right node and define v_a to be the node of P_i that is reached first by P_{i+1} and v_b to be the node of P_i that is reached last by P_{i+1}. Let $P_i = D_0 D_1 D_2$, where D_1 is the sub-path between v_a and v_b, D_0 starts at v_0, and D_2 ends at v_t; and let $P_{i+1} = E_0 E_1 E_2$, where E_0 starts at a free left node and ends at v_a, E_1 is the sub-path of P_{i+1} between v_a and v_b, and E_2 starts at v_b and ends at a free right node.

The sub-path E_1 of P_{i+1} uses at least one edge of P_i. Moreover, E_1 can leave and come back to P_i several times via forward and backward cycles.

> Claim 1 (Bypass Costs). For any such cycle C the following holds:
>
> (i) If C is a forward cycle, then $\omega^{\pm}_{M_{i+1}}(C \cap P_{i+1}) \geqslant \omega^{\pm}_{M_{i+1}}(C \cap P_i)$.
>
> (ii) If C is a backward cycle, then $\omega^{\pm}_{M_{i+1}}(C \cap P_{i+1}) \geqslant -\omega^{\pm}_{M_{i+1}}(C \cap P_i)$.

Proof of Claim. Let C be a directed cycle, either in G_i or in G_{i+1} and let $A = C \cap P_i$ and $B = C \cap P_{i+1}$. Path A starts at a right node v_l and ends at a left node v_k and B starts at v_k and ends at v_l but uses no other nodes of P_i in between.

If C is a directed cycle in G_i, then we have $l < k$ and it holds

$$\omega^+(A \cap M_i) + \omega^+(B \cap M_i) \leqslant \omega^+(A - M_i) + \omega^+(B - M_i)\,,$$

since otherwise, there is matching $M_i' = M_i \oplus C$ of size $|M_i'| = |M_i|$ and weight $\omega^+(M_i') < \omega^+(M_i)$, which contradicts the assumption that M_i has minimum weight among all matchings of size $|M_i|$. We are interested in the incremental weight of the part B of P_{i+1} with respect to M_{i+1}, i. e., in G_{i+1}. Since the change from G_i to G_{i+1}

reverses all edges from A while the orientation of the edges of B remains the same, the following holds:

$$\begin{aligned}\omega^{\pm}_{M_{i+1}}(B) = \omega^{\pm}_{M_i}(B) &= \omega^{+}(B - M_i) - \omega^{+}(B \cap M_i) \\ &\geqslant \omega^{+}(A \cap M_i) - \omega^{+}(A - M_i) \\ &= \omega^{+}(A - M_{i+1}) - \omega^{+}(A \cap M_{i+1}) = \omega^{\pm}_{M_{i+1}}(A)\,.\end{aligned}$$

Analogously, if C is a directed cycle in G_{i+1}, then we have $l > k$ and

$$\omega^{+}(A \cap M_{i+1}) + \omega^{+}(B \cap M_{i+1}) \leqslant \omega^{+}(A - M_{i+1}) + \omega^{+}(B - M_{i+1})\,,$$

from which it follows that $\omega^{\pm}_{M_{i+1}}(B) \geqslant -\omega^{\pm}_{M_{i+1}}(A)$, and hence the claim holds. □

In order to bound the incremental weight of P_{i+1}, we have to consider two cases.

$a > b$. From Claim 1 it follows that regardless of the number and order of forward and backward cycles that are used by the sub-path E_1 of P_{i+1}, the incremental weight of E_1 with respect to M_{i+1} cannot be smaller than the incremental weight of the sub-path D_1 of P_i; more formally

$$\omega^{\pm}_{M_{i+1}}(E_1) \geqslant \omega^{\pm}_{M_{i+1}}(D_1) = -\omega^{\pm}_{M_i}(D_1)\,.$$

Furthermore, it holds that $\omega^{\pm}_{M_i}(E_0) \geqslant \omega^{\pm}_{M_i}(D_0D_1)$, since otherwise E_0D_2 would be an augmenting path for M_i of lower incremental weight than P_i, which contradicts the minimality of $\omega^{\pm}_{M_i}(P_i)$. Symmetrically, we have that $\omega^{\pm}_{M_i}(E_2) \geqslant \omega^{\pm}_{M_i}(D_1D_2)$, because D_0E_2 is also an augmenting path for M_i. Using that $\omega^{\pm}_{M_i}(E_0) = \omega^{\pm}_{M_{i+1}}(E_0)$ and $\omega^{\pm}_{M_i}(E_2) = \omega^{\pm}_{M_{i+1}}(E_2)$, we conclude

$$\begin{aligned}\omega^{\pm}_{M_{i+1}}(P_{i+1}) &= \omega^{\pm}_{M_{i+1}}(E_0) + \omega^{\pm}_{M_{i+1}}(E_1) + \omega^{\pm}_{M_{i+1}}(E_2) \\ &\geqslant \omega^{\pm}_{M_i}(D_0D_1) - \omega^{\pm}_{M_i}(D_1) + \omega^{\pm}_{M_i}(D_1D_2) \\ &= \omega^{\pm}_{M_i}(D_0D_1D_2) = \omega^{\pm}_{M_i}(P_i)\,.\end{aligned}$$

$a < b$. Analogously to the first case, in consequence of Claim 1, we have that

$$\omega^{\pm}_{M_{i+1}}(E_1) \geqslant -\omega^{\pm}_{M_{i+1}}(D_1) = \omega^{\pm}_{M_i}(D_1)\,,$$

as well as $\omega^{\pm}_{M_i}(E_0) \geqslant \omega^{\pm}_{M_i}(D_0)$, and $\omega^{\pm}_{M_i}(E_2) \geqslant \omega^{\pm}_{M_i}(D_2)$. It follows that

$$\begin{aligned}\omega^{\pm}_{M_{i+1}}(P_{i+1}) &= \omega^{\pm}_{M_i}(E_0) + \omega^{\pm}_{M_{i+1}}(E_1) + \omega^{\pm}_{M_i}(E_2) \\ &\geqslant \omega^{\pm}_{M_i}(D_0) + \omega^{\pm}_{M_i}(D_1) + \omega^{\pm}_{M_i}(D_2) \\ &= \omega^{\pm}_{M_i}(D_0D_1D_2) = \omega^{\pm}_{M_i}(P_i)\,.\end{aligned}$$

In both cases we have a contradiction to the assumption that $\omega^{\pm}_{M_{i+1}}(P_{i+1}) < \omega^{\pm}_{M_i}(P_i)$. This finishes the proof of the lemma. ■

3.4. The Generalized Selfless Algorithm

In this section we discuss experimental results concerning the generalized selfless algorithm (Algorithm 3) — parts of them are presented in [DGM+09].

3.4.1. Graph Model

We consider hypergraphs $H^d_{m,n}$ of type B. These are random hypergraphs that consist of the node set $[m]$ as well as n hyperedges chosen uniformly at random from $\binom{[m]}{d}$ with replacement.

3.4.2. Problem Description

Our aim is to investigate the potential of Algorithm 3 for solving the following problem. Given $H^d_{m,n}$ and parameters k and ℓ. Find a (k,ℓ)-orientation of the hypergraph, i. e., an orientation of the edges such that each edge is directed to k pairwise distinct nodes and the indegree of each node is at most ℓ.

Remark. Of course, maximum flow algorithms, see, e. g., [AMO93, Chapters 6–8], and for $k = 1$ maximum cardinality matching algorithms, see, e. g., [AMO93, Section 12.3], can be used to solve this problem, cf., Section 3.6.3.1. However, their running time is not linear in the number of edges.

As we have discussed in Section 3.1.1.5 for constant (d,k,ℓ) there is an *orientation threshold* $\hat{c}_{k,\ell}(d)$ such that if the load factor, or *edge density*, $c = n/m$ is bounded away by a constant, then $H^d_{m,n}$ admits a (k,ℓ)-orientation a. a. s. if c is below the threshold; and $H^d_{m,n}$ admits no (k,ℓ)-orientation a. a. s. if c is above the threshold. Table 3.4.2 lists some values.

For $d = k+1 \geqslant 2$ and $\ell = 1$, the thresholds are simply given via

$$\hat{c}_{k,\ell}(d) = \frac{1}{d\cdot(d-1)}\,,$$

as shown, e. g., in [LP12, Section 2]. For the other relevant cases, i. e., $d > k \geqslant 1$, $\ell \geqslant 1$, with $\max\{d-k,\ell\} \geqslant 2$, they can be defined using two functions, cf., Section 2.3.1, which we call *key function*

$$\mathrm{key}(\lambda,d,k,\ell) = \frac{\lambda}{d\cdot \Pr\left(\mathrm{Bin}\left[d-1,\Pr(\mathrm{Po}[\lambda]<\ell)\right]<k\right)}\,, \tag{3.1}$$

and *density function*

$$\mathrm{dens}(\lambda,d,k,\ell) = \frac{\lambda\cdot\sum_{i=0}^{k-1}(k-i)\cdot\Pr\left(\mathrm{Bin}\left[d,\Pr(\mathrm{Po}[\lambda]<\ell)\right]\right)}{d\cdot\Pr\left(\mathrm{Po}[\lambda]\geqslant\ell+1\right)\cdot\Pr\left(\mathrm{Bin}\left[d-1,\Pr(\mathrm{Po}[\lambda]<\ell)\right]<k\right)}\,. \tag{3.2}$$

$k \setminus \ell$	1	2	3	4	5	6	7
1	0.91794	1.97640	2.99186	3.99701	4.99887	5.99957	6.99983
2	0.16667	0.75804	1.29856	1.82824	2.35160	2.87067	3.38660

(a) $d = 3$

$k \setminus \ell$	1	2	3	4	5	6	7
1	0.97677	1.99648	2.99939	3.99989	4.99998	6.00000	7.00000
2	0.37985	0.92664	1.45154	1.96692	2.47701	2.98385	3.48858
3	0.08333	0.44086	0.78921	1.13669	1.48303	1.82827	2.17251

(b) $d = 4$

Table 3.4.2.: Thresholds $\hat{c}_{k,\ell}(d)$ for the existence of a (k, ℓ)-orientation of $H^d_{m,n}$. The edge densities are rounded to five decimal places.

As shown in [Lel12a, Lel12b], $\hat{c}_{k,\ell}(d)$ is the value of the key function at the unique positive lambda where the density function equals ℓ, see Section 2.3.1, i. e.,

$$\hat{c}_{k,\ell}(d) = \text{key}(\lambda^{\text{den}}_{\ell}, d, k, \ell), \text{ where } \lambda^{\text{den}}_{\ell} \in (0, \infty) \text{ is the solution of } \text{dens}(\lambda) = \ell.$$

3.4.3. Algorithm

We give another description of the generalized selfless algorithm as alternative to Algorithm 3 in order to show how we can obtain linear running time. As before, the alternative algorithm works in rounds, however, now the hypergraph does not remain intact but is shrunk.

Input is a hypergraph $H = (V, E)$ with node set V, edge set E, and parameters

$\hat{d}$ — the maximum edge size,

k — the number of orientations per edge, at most the minimum edge size, and

ℓ — the maximum indegree per node.

The algorithm works as follows.

1. Set *scale* to the least common multiple of $2, 3, \ldots, \hat{d}$. This value is a scale factor used to obtain integer priorities.
2. Set the maximum priority *maxPrio* to $1 + \textit{scale} \cdot \ell$.
3. For each $v \in V$ let $\textit{inc}_v$ be a variable that stores the number of edges directed towards v. Initialize $\textit{inc}_v$ with 0 for all $v \in V$.
4. For each $e \in E$ let $\textit{req}_e$ be a variable that stores the number of nodes to which e must be directed. Initialize $\textit{req}_e$ with k for all $e \in E$.

5. For each node $v \in V$ let $unbPrio_v$ be a variable that holds its current *unbounded* priority, initialized as follows

$$unbPrio_v \leftarrow scale \cdot inc_v + \sum_{e \in E\colon v \in e} scale \cdot req_e/|e|\,.$$

6. Insert all nodes $v \in V$ with $\deg(v) > 0$ and $inc_v < \ell$ into a priority queue $\mathfrak{Q}$ according to their *bounded* priority $\mathrm{prio}(v)$, where

$$\mathrm{prio}(v) \coloneqq \begin{cases} 0, & \text{if } \deg(v) + inc_v \leqslant \ell \\ \min\{maxPrio, unbPrio_v\}, & \text{otherwise} \end{cases}.$$

(The priority queue and the data structures for the nodes and edges are sketched afterwards.)

7. While $\mathfrak{Q}$ is not empty do:
   a) Find a node $\check{v}$ in $\mathfrak{Q}$ with minimum priority; break ties by randomization.
   b) If $\mathrm{prio}(\check{v}) = maxPrio$, then return failure. Else
      i. Find an edge $\check{e}$ of minimum size $|\check{e}|$ that is incident with $\check{v}$; break ties by randomization.
      ii. Store: "$\check{e}$ is directed towards $\check{v}$".
      iii. Increment $inc_{\check{v}}$ by 1 and $unbPrio_{\check{v}}$ by $1 \cdot scale$. Decrement $req_{\check{e}}$ by 1.
      iv. Create an empty list $\mathfrak{L}_E$ for edges that have to be updated as well as an empty list $\mathfrak{L}_V$ for nodes that have to be updated.
      v. If $inc_{\check{v}} < \ell$, then enqueue only $\check{e}$ on $\mathfrak{L}_E$, otherwise collect all edges that are incident with $\check{v}$ in $\mathfrak{L}_E$.
      vi. For all e in $\mathfrak{L}_E$ do:
         A. Compute a decrement dec according to

$$dec \leftarrow \begin{cases} scale \cdot (req_e + 1)/|e| & \text{if } e = \check{e} \\ scale \cdot req_e/|e| & \text{otherwise} \end{cases}.$$

         B. For all v from e decrement their unbounded priority $unbPrio_v$ by dec.
         C. For all v from $e - \{\check{v}\}$ increment the unbounded priority $unbPrio_v$ by $scale \cdot req_e/(|e| - 1)$;
         D. Collect all nodes v from e in $\mathfrak{L}_V$.
         E. Remove $\check{v}$ from e. This reduces $|e|$ and $\deg(\check{v})$ by 1.
      vii. If $req_{\check{e}} = 0$, then remove $\check{e}$ from E. This will shrink the neighborhood of its nodes.

viii. Update the priority queue. For each node v from $\mathfrak{L}_V$ remove v from $\mathfrak{Q}$; if $\deg(v) > 0$ and $inc_v < \ell$, then calculate its new bounded priority $\text{prio}(v)$ and re-insert v into $\mathfrak{Q}$.

8. If there remain edges $e \in E$ with $req_e > 0$, then return failure.

3.4.3.1. Data Structures

The data structures for priority queue, nodes, and edges can be realized as follows.

Priority Queue The priority queue is an array $\mathfrak{Q}[0 \dots maxPrio]$ of size $maxPrio+1$. An element $\mathfrak{Q}[p]$ is an array of pointers to nodes that have priority p.

Nodes For each node v we store a pointer to its associated cell within $\mathfrak{Q}[\text{prio}(v)]$. Furthermore, we hold an array of pointers to its incident edges (edge list), and an array of pointers to its associated cells within the node lists of its incident edges (positions in node lists).

Edges For each edge e we store an array of pointers to its incident nodes (node list), as well as an array of pointers to its associated cells within the edge lists of its incident nodes (positions in edge lists).

Determining the minimum priority in step 7a can be realized via linear search for $p = 0, 1, 2, \dots$ until a non-empty array $\mathfrak{Q}[p]$ is found. The start value for the next round can be adjusted in step 7(b)viii. Removing an element from an array $\mathfrak{Q}[p]$ is done via overwriting the element with the last entry, adjusting the pointers, and removing the last entry. In order to remove a node from the node list of an edge and vice versa the edge from the edge list of the node, essentially the same procedure is used.

3.4.3.2. Running Time

For the analysis we assume that $(c, \hat{d}, k, \ell)$ is constant.

The initialization of $unbPrio_v$ for all $v \in V$ needs time $O(n)$, since each edge e has to be considered only $|e| \leqslant \hat{d}$ times. Since $maxPrio$ is a constant, finding a node $\check{v}$ in $\mathfrak{Q}$ with minimum priority $\text{prio}(\check{v})$ needs constant time. If $\text{prio}(\check{v}) < maxPrio$, then $\check{v}$ has only a constant number of incident edges. Hence, determining an edge of minimum size needs constant time too. Furthermore, the number of edges in $\mathfrak{L}_E$ and the number of nodes in $\mathfrak{L}_V$ that have to be updated in each round of the while loop is also bounded by a constant. Each update of a node with respect to its incident edges, its unbounded priority, and its position in the priority data structure needs only constant time. The same holds for each update of an edge. Since, in each round of the while loop either an edge is directed, or the algorithm stops, the total number of rounds is linear in n. Hence, the generalized selfless algorithm runs in time $O(n)$ in the worst-case on input $H^d_{m,n}$ for constant parameters $(c, d = \hat{d}, k, \ell)$.

3.4.4. Experiments

For each of the following experiments we used *pseudorandom* hypergraphs $H^d_{m,n}$ with node set $[m]$ and edge set $[n]$, where all edges were randomly chosen, according to type A or type B, via the pseudorandom number generator MT19937 "Mersenne Twister" of the GNU Scientific Library [GDT+11, page 199].

Setup and Measurements We made a selection of parameter vectors (m, d, k, ℓ). For each of those vectors we chose load factors $c_{\text{start}}, c_{\text{end}}$ that fulfill $\hat{c}_{k,\ell} \in [c_{\text{start}}, c_{\text{end}}]$ as well as $c_{\text{end}} - \hat{c}_{k,\ell} \approx \hat{c}_{k,\ell} - c_{\text{start}}$, and created a sequence of configurations $\big((c_i, m, d)\big)_{i\in[l]}$, where $c_i = c_{\text{start}} + i \cdot 10^{-4}$ and $l = \lfloor (c_{\text{end}} - c_{\text{start}}) \cdot 10^4 \rfloor + 1$. According to each configuration (c_i, m, d) we generated 100 pseudorandom hypergraphs $H^d_{m,n}$ with $n = c_i \cdot m$ edges. On each hypergraph we ran the generalized selfless algorithm in order to determine a (k, ℓ)-orientation. Finding a legal orientation was considered a success, finding no legal orientation was considered a failure. We measured the failure rate ξ_i among 100 hypergraphs as a function of c_i and tried to estimate the value of c, where the failure rate is about 0.5. For that, we fit the *sigmoid function*

$$\sigma(c; \gamma, \delta) = \frac{1}{1 + e^{-(c-\gamma)/\delta}} \tag{3.3}$$

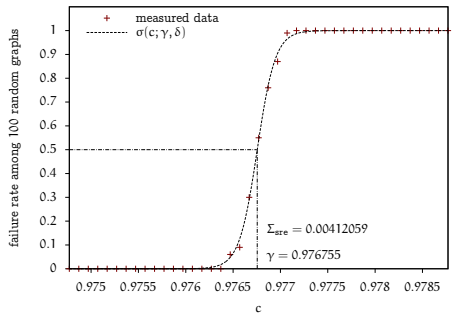
to the sequence of load factor and failure rate pairs $\big((c_i, \xi_i)\big)_{i\in[l]}$, via gnuplot [WKL+12], using the method of least squares. We determined values of γ and δ that lead to a local minimum of the *sum of squares of residuals*

$$\Sigma_{\text{sre}} = \sum_{i\in[l]} \big(\sigma(c_i; \gamma, \delta) - \xi_i\big)^2 . \tag{3.4}$$

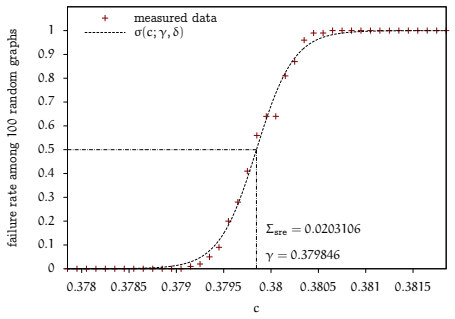
The point $c = \gamma$ is the inflection point of (3.3), which has the property that $\sigma(\gamma; \gamma, \delta) = 0.5$, for $\delta \neq 0$. Our interest is in the absolute difference between the inflection point given by the fit and the theoretical threshold value $\hat{c}_{k,\ell}$.

Results Here we investigate the gradient of the failure rate of the generalized selfless algorithm near the theoretical orientation thresholds.

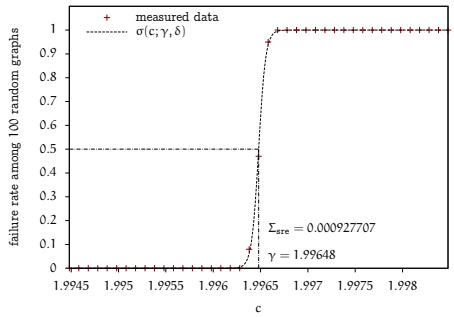
I. Table 3.4.3 gives experimental evidence that our simple algorithm is able to find *placements* for edge densities quite *close to the calculated thresholds* $\hat{c}_{k,\ell}$, for hypergraphs of type A and type B. These results substantiate Conjecture 3.1.1. The absolute difference of the inflection point of the fit function and the threshold $|\hat{c}_{k,\ell}(d) - \gamma|$, as well as the error of the fit Σ_{sre}, are very small in the "non-extreme cases", i. e., for $\max\{d - k, \ell\} \geqslant 2$, and slightly larger in the "extreme case", i. e., for $d - k = \ell = 1$. The slope of the sigmoid curve increases and simultaneously the error Σ_{sre} decreases with increasing d, increasing ℓ, and decreasing k. The transition from total success to total failure is much sharper in the non-extreme cases, see Figure 3.4.3, than in the extreme cases, see Figure 3.4.4.



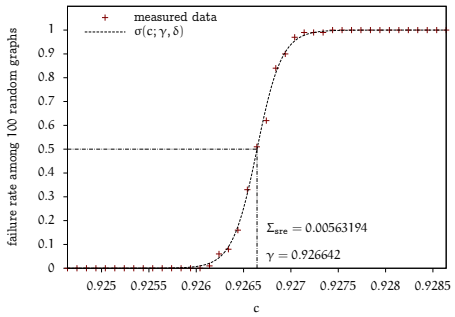
(a) $(d, k, \ell) = (4, 1, 1)$, 41 data points



(b) $(d, k, \ell) = (4, 2, 1)$, 41 data points

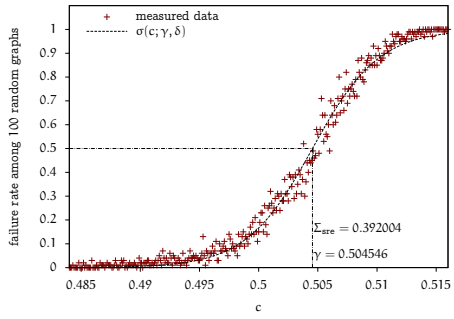


(c) $(d, k, \ell) = (4, 1, 2)$, 41 data points

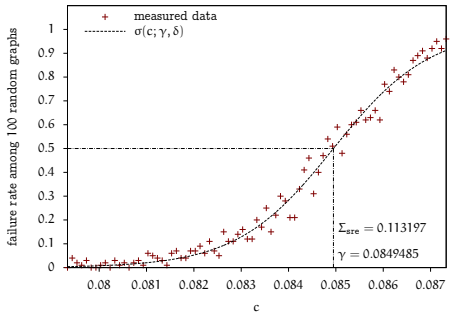


(d) $(d, k, \ell) = (4, 2, 2)$, 41 data points

Figure 3.4.3.: Failure rate of the generalized selfless algorithm when determining *non-extreme orientations* of pseudorandom hypergraphs $H^d_{m,n}$ with 10^6 nodes.



(a) $(d, k, \ell) = (2, 1, 1)$, 321 data points



(b) $(d, k, \ell) = (4, 3, 1)$, 81 data points

Figure 3.4.4.: Failure rate of the generalized selfless algorithm when determining *extreme orientations* of pseudorandom hypergraphs $H^d_{m,n}$ with 10^6 nodes.

(d, k, ℓ)	$[c_{start}, c_{end}]$	γ	$\lvert\hat{c}_{k,\ell}(d) - \gamma\rvert$	Σ_{sre}
$(2, 1, 1)$	$[0.48400, 0.51600]$	0.50454	$4540 \cdot 10^{-6}$	0.307141
$(3, 1, 1)$	$[0.91594, 0.91994]$	0.917902	$33 \cdot 10^{-6}$	0.0166131
$(4, 1, 1)$	$[0.97477, 0.97877]$	0.976756	$14 \cdot 10^{-6}$	0.00859291
$(4, 1, 2)$	$[1.99448, 1.99848]$	1.99648	$3 \cdot 10^{-6}$	0.000177911
$(4, 1, 3)$	$[2.99739, 3.00139]$	2.99938	$5 \cdot 10^{-6}$	$3.44092 \cdot 10^{-5}$
$(4, 2, 1)$	$[0.37785, 0.38185]$	0.379823	$25 \cdot 10^{-6}$	0.0186169
$(4, 2, 2)$	$[0.92464, 0.92864]$	0.926622	$22 \cdot 10^{-6}$	0.0174623
$(4, 2, 3)$	$[1.44954, 1.45354]$	1.45154	$2 \cdot 10^{-6}$	0.00926342
$(4, 3, 1)$	$[0.07933, 0.08733]$	0.0849781	$1645 \cdot 10^{-6}$	0.154957
$(4, 3, 2)$	$[0.43886, 0.44286]$	0.440838	$17 \cdot 10^{-6}$	0.0192226
$(4, 3, 3)$	$[0.78721, 0.79121]$	0.789234	$20 \cdot 10^{-6}$	0.00878215

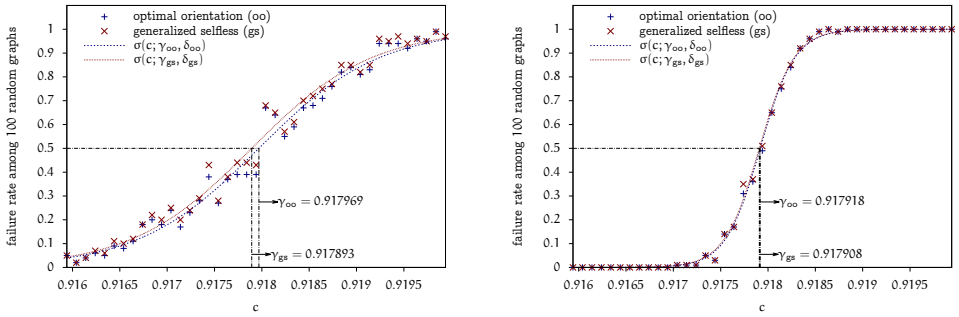
(a) type A

γ	$\lvert\hat{c}_{k,\ell}(d) - \gamma\rvert$	Σ_{sre}
0.504546	$4546 \cdot 10^{-6}$	0.392004
0.917933	$2 \cdot 10^{-6}$	0.00928238
0.976755	$15 \cdot 10^{-6}$	0.00412059
1.99648	$3 \cdot 10^{-6}$	0.000927707
2.99939	$5 \cdot 10^{-6}$	$1.03831 \cdot 10^{-5}$
0.379846	$2 \cdot 10^{-6}$	0.0203106
0.926642	$2 \cdot 10^{-6}$	0.00563194
1.45152	$22 \cdot 10^{-6}$	0.0123738
0.0849485	$1615 \cdot 10^{-6}$	0.113197
0.44084	$15 \cdot 10^{-6}$	0.0170778
0.789214	0	0.0204965

(b) type B

Table 3.4.3.: Approximation of orientation thresholds due to curve fitting of the failure rate of the generalized selfless algorithm using 100 pseudorandom hypergraphs $H^d_{m,n}$ with 10^6 nodes per data point.

II. Figure 3.4.5 depicts the failure rate of the generalized selfless algorithm and the failure rate of an *optimal placement algorithm* for $(d, k, \ell) = (3, 1, 1)$ and pseudorandom hypergraphs (type B) with 10^5 and 10^6 nodes. Clearly our algorithm can fail on hypergraphs that admit a matching but this becomes very rare for $m \geqslant 10^6$. The slope of the sigmoid curve increases and Σ_{sre} decreases with growing n, leading to a sharp transition from total success to total failure.



(a) $m = 10^5$ (b) $m = 10^6$

Figure 3.4.5.: *Comparison of the failure rate* between the generalized selfless algorithm and an optimal placement algorithm using pseudorandom hypergraphs $H^d_{m,n}$ type B with $(d, k, \ell) = (3, 1, 1)$.

3.5. Towards Optimal Degree Distributions for Irregular Cuckoo Hashing

In this section we prove Theorem 3.4, Proposition 3.1, and give evidence for Conjecture 3.1.2 — the proofs appeared in [DR12a].

3.5.1. Graph Model

We study graphs $G^{\bar{d}}_{n,m}\big((\rho_x)_{x\in S}\big)$ of type A which are *bipartite multigraphs* with left node set S, $|S| = n$, and right node set $[m]$, where each left node x from S has D_x right neighbors. The right neighbors are chosen at random with replacement from $[m]$, i. e., they are independent as well as uniformly distributed in $[m]$ and therefore may coincide. The number of choices D_x, which is the degree of x, is a random variable that follows some probability mass function ρ_x. For each x from S let $D_x \geqslant 1$ and let Δ_x be the mean of D_x, that is,

$$\Delta_x \coloneqq \mathrm{Exp}(D_x) = \sum_{d\geqslant 1} d \cdot \rho_x(d)\,.$$

Furthermore, let $\bar{d}$ be the average mean, i. e.,

$$\bar{d} \coloneqq \frac{1}{n} \cdot \sum_{x\in S} \Delta_x\,.$$

We assume that the random variables D_x, $x \in S$, are independent and $\bar{d}$ is a given constant. To simplify notation, for each set $S' \subseteq S$ we define $G_{S'}$ to be the induced bipartite subgraph of $G^{\bar{d}}_{n,m}$ with left node set S' and right node set $[m]$, so in particular we use G_S *synonymously* for $G^{\bar{d}}_{n,m}$. Furthermore, for each $S' \subseteq S$ we define $\mathcal{M}_{S'}$ as the event that $G_{S'}$ has a left-perfect matching. The probability of such an event is called *success probability*.

3.5.2. Problem Description

Our aim is to determine a sequence of probability mass functions $(\rho_x)_{x\in S}$ that maximizes the probability that the random graph $G^{\bar{d}}_{n,m}\big((\rho_x)_{x\in S}\big)$ of type A has a left-perfect matching. Such a sequence is called *optimal*.

3.5.3. Optimality of Concentration in a Unit Length Interval

Here we prove Theorem 3.4. Let n, m, and $\bar{d}$ be fixed and consider some arbitrary but fixed sequence of probability mass functions $(\rho_x)_{x\in S}$. We will show that if this sequence has certain properties contradicting one of the conditions in Theorem 3.4,

then we can apply a modification, obtaining a new sequence $(\rho'_x)_{x\in S}$ with the same average expected value $\bar{d}$, such that the probability that $G\big((\rho'_x)_{x\in S}\big)$ has a left-perfect matching is strictly higher than the probability that $G\big((\rho_x)_{x\in S}\big)$ has a left-perfect matching.

> Lemma 3.5.1 (Concentration Around the Mean, variant of [DGM+10, Proposition 4]). Let $(\rho_x)_{x\in S}$ be given. Let $z \in S$ be arbitrary but fixed. If in ρ_z two degrees with distance at least 2 have non-zero probability then $(\rho_x)_{x\in S}$ is not optimal.

The lemma was stated in [DGM+10] and proven in [DGM+09] for a slightly different graph model. Its proof runs along the lines of [DGM+09]; it is given in Section 3.5.3.1 for completeness.

Lemma 3.5.1 implies that in an optimal sequence each left node has either a fixed degree (with probability 1) or two possible degrees with non-zero probability, where these degrees differ by 1. The lemma and [DGM+09, DGM+10] do not say anything about the relation between the degrees of different nodes. This follows next.

> Lemma 3.5.2 (Average Degrees are Close, [DR12a, Lemma 2]). Let $(\rho_x)_{x\in S}$ be given, where for each $x \in S$ the only degrees with non-zero probability are from $\{\lfloor\Delta_x\rfloor, \lceil\Delta_x\rceil\}$. Let $y, z \in S$ be arbitrary but fixed. If $\lfloor\Delta_y\rfloor$ and $\lfloor\Delta_z\rfloor$ have distance at least 2, or $\lceil\Delta_y\rceil$ and $\lceil\Delta_z\rceil$ have distance at least 2, then $(\rho_x)_{x\in S}$ is not optimal.

Lemma 3.5.2 is proved in Section 3.5.3.2. Using Lemma 3.5.2 one concludes that an optimal sequence restricts the means Δ_x, for each $x \in S$, to an open interval $(l-1, l+1)$ for some integer constant $l \geqslant 2$. Hence all degrees that appear with non-zero probability must be from $\{l-1, l, l+1\}$. With the help of the next lemma one concludes that actually two values are enough.

> Lemma 3.5.3 (Two Neighboring Degrees, [DR12a, Lemma 3]). Let $(\rho_x)_{x\in S}$ be given, where for each $x \in S$ the only degrees with non-zero probability are from $\{\lfloor\Delta_x\rfloor, \lceil\Delta_x\rceil\}$. Let $y, z \in S$ be arbitrary but fixed and assume that Δ_y and Δ_z are non-integral. If $\lceil\Delta_y\rceil$ and $\lfloor\Delta_z\rfloor$ have distance 2 then $(\rho_x)_{x\in S}$ is not optimal.

Lemma 3.5.3 is proved in Section 3.5.3.3. Combining Lemmas 3.5.1 to 3.5.3, we obtain the following for an optimal sequence. If $l \leqslant \bar{d} < l+1$, then we have $l \leqslant \Delta_x \leqslant l+1$ for all $x \in S$, and all degrees that appear with non-zero probability must be from $\{l, l+1\}$. If $\bar{d}$ is an integer, then by definition of $\bar{d}$ we have $\rho_x(\bar{d}) = 1$ for all $x \in S$. Hence Theorem 3.4 follows.

So, to complete the proof of Theorem 3.4, it remains to show Lemmas 3.5.1 to 3.5.3, which is done in Sections 3.5.3.1 to 3.5.3.3.

3.5.3.1. Degrees Must be Concentrated Around the Mean

In this section we prove Lemma 3.5.1. Let $(\rho_x)_{x\in S}$ be given and consider some z from S. We will show that if ρ_z gives positive weight to two degrees, say l and k, and it holds

that $l < \Delta_z < k$ as well as $k - l \geqslant 2$, then the probability that there is a left-perfect matching for the whole key set S cannot be maximal. More precisely we will show that modifying ρ_z to ρ'_z via

$$\begin{aligned} \rho'_z(k) &= \rho_z(k) - \varepsilon & \rho'_z(l) &= \rho_z(l) - \varepsilon \\ \rho'_z(k-1) &= \rho_z(k-1) + \varepsilon & \rho'_z(l+1) &= \rho_z(l+1) + \varepsilon\,, \end{aligned}$$

for $\varepsilon \in (0, \min\{\rho_z(l), \rho_z(k)\}]$, and $\rho'_z(d) = \rho_z(d)$ for all other values d, decreases the failure probability. That is

$$\Pr\left(\overline{\mathcal{M}}_S\right) > \Pr'\left(\overline{\mathcal{M}}_S\right)\,,$$

where $\Pr$ and $\Pr'$ refer to the probability spaces created by the sequences $(\rho_x)_{x\in S}$ and $((\rho_x)_{x\in S-\{z\}}, \rho'_z)$, respectively. Clearly, Δ_z and $\bar{d}$ remain unchanged.

For each element $x \in S - \{z\}$ we fix its degree and neighborhood $N_x = N(\{x\})$. The resulting graph $G_{S-\{z\}}$ can have zero, one or more left-perfect matchings. Let $B \subseteq T$ be the set of right nodes of $G_{S-\{z\}}$ that are matched in every left-perfect matching for $S - \{z\}$. Since there can be a left-perfect matching for S only if there is a left-perfect matching for $S - \{z\}$ it is sufficient to show that

$$\Pr\left(\overline{\mathcal{M}}_S \mid \mathcal{M}_{S-\{z\}}\right) > \Pr'\left(\overline{\mathcal{M}}_S \mid \mathcal{M}_{S-\{z\}}\right)\,. \tag{3.5}$$

Using the law of total probability we see that (3.5) is equivalent to

$$\begin{aligned} &\sum_{b=0}^{n-1} \Pr\left(\overline{\mathcal{M}}_S \mid \mathcal{M}_{S-\{z\}}, |B| = b\right) \cdot \Pr\left(|B| = b \mid \mathcal{M}_{S-\{z\}}\right) \\ &> \sum_{b=0}^{n-1} \Pr'\left(\overline{\mathcal{M}}_S \mid \mathcal{M}_{S-\{z\}}, |B| = b\right) \cdot \Pr'\left(|B| = b \mid \mathcal{M}_{S-\{z\}}\right)\,. \end{aligned} \tag{3.6}$$

For G_S to have a left-perfect matching there must be at least one node in the neighborhood N_z of z that is not an element of B. Therefore we have to show

$$\begin{aligned} &\sum_{b=0}^{n-1} \left[\sum_{d\geqslant 1} \rho_z(d) \cdot \left(\frac{b}{m}\right)^d\right] \cdot \Pr\left(|B| = b \mid \mathcal{M}_{S-\{z\}}\right) \\ &> \sum_{b=0}^{n-1} \left[\sum_{d\geqslant 1} \rho'_z(d) \cdot \left(\frac{b}{m}\right)^d\right] \cdot \Pr'\left(|B| = b \mid \mathcal{M}_{S-\{z\}}\right)\,. \end{aligned}$$

Clearly, $\Pr\left(|B| = b \mid \mathcal{M}_{S-\{z\}}\right)$ is not affected by changing ρ_z to ρ'_z and if $b = 0$ the modification from ρ_z to ρ'_z does not affect the failure probability. Hence it is sufficient to show for each $b > 0$ that

$$\sum_{d\geqslant 1} \rho_z(d) \cdot \left(\frac{b}{m}\right)^d > \sum_{d\geqslant 1} \rho'_z(d) \cdot \left(\frac{b}{m}\right)^d\,. \tag{3.7}$$

By definition of ρ'_z the right-hand side of (3.7) equals

$$\sum_{d \geqslant 1} \rho_z(d) \cdot \left(\frac{b}{m}\right)^d + \varepsilon \cdot \left(-\left(\frac{b}{m}\right)^l + \left(\frac{b}{m}\right)^{l+1} + \left(\frac{b}{m}\right)^{k-1} - \left(\frac{b}{m}\right)^k\right),$$

such that with $\varepsilon > 0$, it follows that (3.7) is equivalent to

$$\left(\frac{b}{m}\right)^l \cdot \left(1 - \frac{b}{m}\right) > \left(\frac{b}{m}\right)^{k-1} \cdot \left(1 - \frac{b}{m}\right),$$

which is true since $0 < b/m < 1$, $k - l \geqslant 2$. As the event $\{b > 0\}$ has positive probability, inequality (3.5) holds. This finishes the proof of Lemma 3.5.1. ■

3.5.3.2. Average Degrees of Different Nodes are Close

In this section we prove Lemma 3.5.2. Consider the probability mass functions ρ_y and ρ_z for the degrees D_y and D_z, respectively. By the hypothesis of the lemma, ρ_y and ρ_z are concentrated on two values each, i. e.,

$$\begin{aligned} \rho_y(k) &= p & \qquad \rho_z(l) &= q \\ \rho_y(k+1) &= 1-p & \qquad \rho_z(l+1) &= 1-q\,, \end{aligned}$$

with $p, q \in [0, 1]$. By the assumption, we may arrange things so that $k - l \geqslant 2$ and we have one of the following situations:

$$\begin{array}{lllll} \text{(I)} & k = \lfloor \Delta_y \rfloor & l = \lfloor \Delta_z \rfloor & p = 1 - (\Delta_y - \lfloor \Delta_y \rfloor) & q = 1 - (\Delta_z - \lfloor \Delta_z \rfloor) \\ \text{(II)} & k + 1 = \lceil \Delta_y \rceil & l + 1 = \lceil \Delta_z \rceil & p = \lceil \Delta_y \rceil - \Delta_y & q = \lceil \Delta_z \rceil - \Delta_z\,. \end{array}$$

We will show that changing ρ_y to ρ'_y and ρ_z to ρ'_z such that $\Delta'_y = \Delta_y - 1$ and $\Delta'_z = \Delta_z + 1$, via

$$\begin{aligned} \rho'_y(k-1) &= p & \qquad \rho'_z(l+1) &= q \\ \rho'_y(k) &= 1-p & \qquad \rho'_z(l+2) &= 1-q\,, \end{aligned}$$

will strictly increase the probability that G_S has a left-perfect matching, while it does not change $\bar{d}$. For this, we will show

$$\Pr\left(\overline{\mathcal{M}}_S\right) > \Pr'\left(\overline{\mathcal{M}}_S\right),$$

where, similarly as before, Pr and Pr′ refer to the probability spaces created by the sequences $(\rho_x)_{x \in S}$ and $((\rho_x)_{x \in S - \{y,z\}}, \rho'_y, \rho'_z)$, respectively. We fix the neighborhood N_x for the remaining elements $x \in S - \{y, z\}$ and therefore the graph $G_{S-\{y,z\}}$. Since there can be a left-perfect matching for S only if there is a left-perfect matching for $S - \{y, z\}$, it is sufficient to show that

$$\Pr\left(\overline{\mathcal{M}}_S \mid \mathcal{M}_{S-\{y,z\}}\right) > \Pr'\left(\overline{\mathcal{M}}_S \mid \mathcal{M}_{S-\{y,z\}}\right). \tag{3.8}$$

Let

$$\mathrm{Fail}(d_y, d_z) = \Pr\left(\overline{\mathcal{M}}_S \mid \mathcal{M}_{S-\{y,z\}}, D_y = d_y, D_z = d_z\right) . \tag{3.9}$$

Then (3.8) holds if and only if

$$\sum_{\substack{d_y \in \{k,k+1\} \\ d_z \in \{l,l+1\}}} \mathrm{Fail}(d_y, d_z) \cdot \rho_y(d_y) \cdot \rho_z(d_z) > \sum_{\substack{d_y \in \{k-1,k\} \\ d_z \in \{l+1,l+2\}}} \mathrm{Fail}(d_y, d_z) \cdot \rho'_y(d_y) \cdot \rho'_z(d_z) . \tag{3.10}$$

Note that if $k - l = 2$, then the summand regarding $d_y = k$ and $d_z = l + 1$ on the left-hand side is the same as the summand regarding $d_y = k - 1$ and $d_z = l + 2$ on the right-hand side. Hence, to prove (3.10) it is sufficient to show that for

$$\begin{aligned} k - l \geqslant 3 : \quad & \mathrm{Fail}(k, l+1) > \mathrm{Fail}(k-1, l+2) , \text{ and for} \\ k - l \geqslant 2 : \quad & \mathrm{Fail}(k+1, l) > \mathrm{Fail}(k, l+1) \\ & \wedge\ \mathrm{Fail}(k+1, l+1) > \mathrm{Fail}(k, l+2) \\ & \wedge\ \mathrm{Fail}(k, l) > \mathrm{Fail}(k-1, l+1) . \end{aligned}$$

This is subsumed by the following condition:

$$\mathrm{Fail}(k, l) > \mathrm{Fail}(k-1, l+1) , \text{ for } k - l \geqslant 2 . \tag{3.11}$$

To prove (3.11), consider the fixed graph $G_{S-\{y,z\}}$. We classify the right nodes of $G_{S-\{y,z\}}$ according to the following three types:

- ▷ We call v *blocked* if v is matched in all left-perfect matchings of $G_{S-\{y,z\}}$.
- ▷ We call v *free* if v is never matched in any left-perfect matching of $G_{S-\{y,z\}}$.
- ▷ We call v *half-free* if v is neither blocked nor free.

Let B be the set of blocked nodes, let F be the set of free nodes, and let $F_\bullet$ be the set of half-free nodes. Elements of $\overline{B} = F \cup F_\bullet$ are called *non-blocked* nodes. For a moment consider only the non-blocked nodes. For each right node set $V \subseteq \overline{B}$ let H_V be an auxiliary graph with node set V that has an edge between two nodes $v_1, v_2 \in V$ if and only if there exists a left-perfect matching for $G_{S-\{y,z\}}$ in which v_1 and v_2 simultaneously are *not* matched. Now consider the full set $\overline{B}$ of non-blocked nodes and the corresponding graph $H_{\overline{B}}$. We define $\sim$ as the following binary relation: $v_1 \sim v_2$, for nodes v_1 and v_2, if $\{v_1, v_2\}$ is *not* an edge in $H_{\overline{B}}$. The following observation is crucial.

Claim 2 (Independent Sets). The relation $\sim$ (no edge) is an equivalence relation.

Proof of Claim. Clearly $\sim$ is reflexive and symmetric. Assume for a contradiction $\sim$ is not transitive. That is, we have three nodes v_1, v_2 and v_3 with $v_1 \sim v_3$ and $v_2 \sim v_3$

but $v_1 \not\sim v_2$. Let $V = \{v_1, v_2, v_3\}$ and consider the graph H_V. By definition of $\sim$ the edge $\{v_1, v_2\}$ is present in H_V, while the other edges $\{v_1, v_3\}$ and $\{v_2, v_3\}$ are not.

If there is a free node in V, then H_V is connected since $V \subseteq F \cup F_\bullet$ and by definition of the edge set of H_V, leading to a contradiction.

Therefore it remains to consider the case where all nodes of V are half-free nodes. Let M and M' be two matchings in $G_{S-\{y,z\}}$, where in M node v_3 is unmatched but nodes v_1 and v_2 are matched, and in M' node v_3 is matched but nodes v_1 and v_2 are unmatched. Matching M exists since as a non-blocked node, v_3 is unmatched in at least one matching, and in such a matching nodes v_1 and v_2 must be matched because of the absence of edges $\{v_1, v_3\}$ and $\{v_2, v_3\}$ in H_V. Likewise matching M' exists, since the edge $\{v_1, v_2\}$ in H_V assures a matching where v_1 and v_2 are unmatched, and under this assumption the absence of edge $\{v_1, v_3\}$ implies that the half-free node v_3 is matched.

Now consider the bipartite multigraph $M \cup M'$ consisting of all edges from both left-perfect matchings and the corresponding nodes. We can make the following observations about $M \cup M'$:

- ▷ Nodes on the left side have degree 2, since both matchings are left-perfect.
- ▷ Nodes on the right side have degree 1 or 2; in particular, v_1, v_2, and v_3 have degree 1.

It follows that $M \cup M'$ is a union of node-disjoint paths and cycles of even length, where on all paths and cycles edges from M and M' alternate. Nodes v_1 and v_2 must be at the ends of two distinct paths, since both are incident to M-edges. Node v_3 must be at the end of a path as well, incident to an M'-edge.

Without loss of generality, we may assume that v_1 and v_3 do not lie on the same path. (If they do, we know that v_2 lies on a different path, and we swap the names of v_1 and v_2.) Starting from M', we get a new left-perfect matching in which neither v_1 nor v_3 are matched by replacing the M'-edges on the path with v_3 by the M-edges on this path. Therefore there must be an edge $\{v_1, v_3\}$ in H_V, which contradicts our assumption, proving the claim. □

From the definition of $\sim$ and Claim 2 it follows that the right node set T of $G_{S-\{y,z\}}$ can be subdivided into disjoint segments $B \cup I_1 \cup I_2 \cup \ldots = T$, where B is the set of blocked nodes and $I_1, I_2, \ldots$ are the equivalence classes of $\sim$, which is equivalent to saying they are the maximal independent sets in $H_{\overline{B}}$. (Note that each free node leads to a one-element set I.) With this characterization of $H_{\overline{B}}$ we can express the event that for fixed neighborhoods N_x, $x \in S - \{y, z\}$, that admit a left-perfect matching for $G_{S-\{y,z\}}$ there is no left-perfect matching for G_S as follows:

$$\{N_y \subseteq B\} \cup \{N_z \subseteq B\} \cup \bigcup_j \{(N_y \cup N_z) \subseteq (B \cup I_j)\}. \tag{3.12}$$

Let $\mathcal{BI}_{S-\{y,z\}}(b, r, i_1, \ldots, i_r)$ be the event that $G_{S-\{y,z\}}$ has $|B| = b$ many blocked nodes and r (non-empty) maximal independent sets $I_1, I_2, \ldots, I_r$ according to the

definition above, with $|I_j| = i_j$ and $i_1 \leqslant i_2 \leqslant \ldots \leqslant i_r$. Let

$$\begin{aligned}&\mathrm{fail}(d_y, d_z, b, r, i_1, \ldots, i_r) = \\ &\quad \Pr\left(\overline{\mathcal{M}}_S \mid \mathcal{M}_{S-\{y,z\}}, D_y = d_y, D_z = d_z, \mathcal{BI}_{S-\{y,z\}}(b, r, i_1, \ldots, i_r)\right) .\end{aligned}$$

From (3.12) we obtain the following, by the principle of inclusion-exclusion:

$$\begin{aligned}\mathrm{fail}(d_y, d_z, b, r, i_1, \ldots, i_r) = &\left(\frac{b}{m}\right)^{d_y} + \left(\frac{b}{m}\right)^{d_z} - \left(\frac{b}{m}\right)^{d_y} \cdot \left(\frac{b}{m}\right)^{d_z} \\ &+ \sum_{j=1}^{r} \left[\left(\frac{i_j + b}{m}\right)^{d_y} - \left(\frac{b}{m}\right)^{d_y}\right] \cdot \left[\left(\frac{i_j + b}{m}\right)^{d_z} - \left(\frac{b}{m}\right)^{d_z}\right] .\end{aligned}$$

Using the law of total probability we can rewrite the value $\mathrm{Fail}(d_y, d_z)$ (3.9) as follows:

$$\mathrm{Fail}(d_y, d_z) = \sum_{(b, r, i_1, \ldots, i_r)} \mathrm{fail}(d_y, d_z, b, r, i_1, \ldots, i_r) \cdot \Pr(\mathcal{BI}_{S-\{y,z\}}(b, r, i_1, \ldots, i_r) \mid \mathcal{M}_{S-\{y,z\}}) .$$

We will abbreviate $\mathrm{fail}(d_y, d_z, b, r, i_1, \ldots, i_r)$ by $\mathrm{fail}(d_y, d_z)$ for the rest of Section 3.5. In order to prove (3.11) it is sufficient to show

$$\mathrm{fail}(k, l) > \mathrm{fail}(k-1, l+1) , \tag{3.13}$$

for each $\mathcal{BI}$-vector $(b, r, i_1, \ldots, i_r)$. Let $\iota_j = i_j/m$ and let $\beta = b/m$. Thus,

$$\mathrm{fail}(k, l) = \beta^k + \beta^l - \beta^{k+l} + \sum_{j=1}^{r} \left[(\iota_j + \beta)^k - \beta^k\right] \cdot \left[(\iota_j + \beta)^l - \beta^l\right] . \tag{3.14}$$

Hence, inequality (3.13) holds if and only if

$$\begin{aligned}\beta^k + \beta^l - \beta^{k-1} - \beta^{l+1} > \sum_{j=1}^{r} \Big(&\left[(\iota_j + \beta)^{k-1} - \beta^{k-1}\right] \cdot \left[(\iota_j + \beta)^{l+1} - \beta^{l+1}\right] \\ &- \left[(\iota_j + \beta)^k - \beta^k\right] \cdot \left[(\iota_j + \beta)^l - \beta^l\right] \Big)\end{aligned}$$

or, equivalently,

$$(1 - \beta) \cdot (\beta^l - \beta^{k-1}) > \sum_{j=1}^{r} \iota_j \cdot \underbrace{\left[\beta^l \cdot (\iota_j + \beta)^{k-1} - \beta^{k-1} \cdot (\iota_j + \beta)^l\right]}_{=: f(l, k, \iota_j, \beta)} . \tag{3.15}$$

Note that if $r = 1$ there is no left-perfect matching for G_S. Hence we are only interested in the case $r \geqslant 2$, which implies that $i_j < m - b$ and $\iota_j < 1 - \beta$, respectively. Consider the right-hand side of (3.15). The expression within the square brackets increases monotonically with increasing ι_j, since we have

$$\frac{\partial f(l, k, \iota_j, \beta)}{\partial \iota_j} = (k-1) \cdot \beta^l \cdot (\iota_j + \beta)^{k-2} - l \cdot \beta^{k-1} \cdot (\iota_j + \beta)^{l-1} ,$$

and $\frac{k-1}{l} \cdot (\iota_j + \beta)^{k-l-1} > \beta^{k-l-1}$ holds because of $k - l \geqslant 2$ and $\iota_j + \beta > \beta$. Therefore replacing ι_j with $1-\beta$ within f and using that $\sum_{j=1}^{r} \iota_j = 1 - \beta$ strictly increases the right-hand side of (3.15) and yields the left-hand side of (3.15). But since we assume $\iota_j < 1-\beta$ the strict inequality holds. Due to the fact that the event $\{r \geqslant 2\}$ has positive probability Lemma 3.5.2 follows. ■

3.5.3.3. Optimal Distributions Use Only Two Neighboring Degrees

In this section we prove Lemma 3.5.3. Consider the probability mass functions ρ_y and ρ_z for the degrees D_y and D_z respectively. Let $\lfloor \Delta_y \rfloor = l$ and $\lfloor \Delta_z \rfloor = l - 1$ as well as $p = 1 - (\Delta_y - \lfloor \Delta_y \rfloor)$ and $q = 1 - (\Delta_z - \lfloor \Delta_z \rfloor)$. By the hypothesis of the lemma we have

$$\begin{aligned} \rho_y(l) &= p & \rho_z(l-1) &= q \\ \rho_y(l+1) &= 1-p & \rho_z(l) &= 1-q\,, \end{aligned}$$

with $p \in (0,1)$ and $q \in (0,1)$. To prove Lemma 3.5.3 we will show that changing ρ_y to ρ'_y and ρ_z to ρ'_z, via

$$\begin{aligned} \rho'_y(l) &= p + \varepsilon & \rho'_z(l-1) &= q - \varepsilon \\ \rho'_y(l+1) &= 1-p-\varepsilon & \rho'_z(l) &= 1-q+\varepsilon\,, \end{aligned}$$

for some small perturbation $\varepsilon \neq 0$ will strictly increase the probability that G_S has a left-perfect matching, while it does not change $\bar{d}$. As in the proof of Lemma 3.5.2 we will obtain that

$$\Pr\left(\overline{\mathcal{M}}_S\right) > \Pr'\left(\overline{\mathcal{M}}_S\right)\,,$$

proving that $(\rho_x)_{x \in S}$ cannot be optimal. As before we fix the neighborhood N_x for the remaining elements $x \in S - \{y,z\}$ and therefore the graph $G_{S-\{y,z\}}$. As in Lemma 3.5.2, we conclude that it is sufficient to show that for some perturbation term $\varepsilon \neq 0$ we have

$$\sum_{\substack{d_y \in \{l,l+1\} \\ d_z \in \{l-1,l\}}} \text{Fail}(d_y,d_z) \cdot \rho_y(d_y) \cdot \rho_z(d_z) > \sum_{\substack{d_y \in \{l,l+1\} \\ d_z \in \{l-1,l\}}} \text{Fail}(d_y,d_z) \cdot \rho'_y(d_y) \cdot \rho'_z(d_z)\,.$$

Subtracting the left-hand side from right-hand side gives the equivalent formulation

$$\begin{aligned} \left[-\varepsilon^2 - \varepsilon \cdot (p-q)\right] \cdot \underbrace{\left[\text{Fail}(l,l-1) + \text{Fail}(l+1,l) - \text{Fail}(l,l) - \text{Fail}(l+1,l-1)\right]}_{=:K_0} \\ -\varepsilon \cdot \underbrace{\left[\text{Fail}(l+1,l-1) - \text{Fail}(l,l)\right]}_{=:K_1} < 0\,, \end{aligned}$$

or, abbreviated:

$$-\varepsilon^2 \cdot K_0 - \varepsilon \cdot \underbrace{\left[(p-q) \cdot K_0 + K_1\right]}_{=:L} < 0\,. \tag{3.16}$$

From (3.11), which was proven in Lemma 3.5.2, it follows that $K_1 > 0$. There are three cases.

$K_0 = 0$. Since we have $K_1 > 0$, it is easy to see that (3.16) holds for $\varepsilon > 0$.

$K_0 > 0$. Regardless whether L is zero, positive, or negative, (3.16) holds for some small $\varepsilon \neq 0$.

$K_0 < 0$. The only critical case would be $L = 0$, but we will show that $K_1 > -K_0$ and therefore $L > 0$, implying that (3.16) holds for small $\varepsilon > 0$.

Claim 3 (There is No Critical Case). $K_1 > -K_0$.

Proof of Claim. Inequality $K_1 > -K_0$ holds if and only if

$$\mathrm{Fail}(l+1, l) + \mathrm{Fail}(l, l-1) > 2 \cdot \mathrm{Fail}(l, l)\,.$$

As before we will simply show the sufficient condition

$$\mathrm{fail}(l+1, l) + \mathrm{fail}(l, l-1) > 2 \cdot \mathrm{fail}(l, l)\,.$$

Using (3.14) in combination with the substitutions $\iota_j = i_j/m$ and $\beta = b/m$ the condition can be written as

$$\begin{aligned}(1-\beta)^2 \cdot \left[\beta^{l-1} - \beta^{2l-1}\right] > \sum_{j=1}^{r} (1-\beta)^2 \cdot \left[(\iota_j+\beta)^l \cdot \beta^{l-1} - \beta^{2l-1}\right] \\ - \sum_{j=1}^{r} [1-(\iota_j+\beta)]^2 \cdot \left[(\iota_j+\beta)^{2l-1} - (\iota_j+\beta)^{l-1} \cdot \beta^l\right].\end{aligned}$$

Note that the subtrahend of the right-hand side is non-negative. Hence it is sufficient to show that

$$(1-\beta)^2 \cdot \left[\beta^{l-1} - \beta^{2l-1}\right] > (1-\beta)^2 \cdot \sum_{j=1}^{r} (\iota_j+\beta)^l \cdot \beta^{l-1} - r \cdot (1-\beta)^2 \cdot \beta^{2l-1}\,. \quad (3.17)$$

Bounding $\sum_{j=1}^{r} (\iota_j+\beta)^l$ using the binomial theorem gives

$$\begin{aligned}\sum_{j=1}^{r} (\iota_j+\beta)^l &= \sum_{j=1}^{r} \sum_{i=0}^{l} \binom{l}{i} \cdot \iota_j^i \cdot \beta^{l-i} = r \cdot \beta^l + \sum_{i=1}^{l} \binom{l}{i} \cdot \beta^{l-i} \cdot \sum_{j=1}^{r} \iota_j^i \\ &< r \cdot \beta^l + \sum_{i=1}^{l} \binom{l}{i} \cdot \beta^{l-i} \cdot \left[\sum_{j=1}^{r} \iota_j\right]^i = (r-1) \cdot \beta^l + 1\,, \end{aligned} \quad (3.18)$$

where the last step follows from $\sum_{j=1}^{r} \iota_j = 1-\beta$. Using (3.18) to estimate $\sum_{j=1}^{r} (\iota_j+\beta)^l$ by $(r-1) \cdot \beta^l + 1$ in (3.17), followed by an obvious calculation, shows that (3.17) holds and thus the claim. □

This finishes the proof of the lemma and hence completes the proof of Theorem 3.4. ■

3.5.4. Essentially Two Different Strategies

In this section, we give evidence for Conjecture 3.1.2, which says that essentially two types of degree distributions may be optimal, if the ratio n/m is fixed to some value $c \neq \hat{c}(\bar{d})$: one in which all keys are given fixed degrees l or $l+1$, and one in which each node chooses one of l and $l+1$ at random, governed by the same distribution on $\{l, l+1\}$. We indicate under what circumstances the one or the other is best.

Assume we are in the situation of Theorem 3.4 (ii), i. e., $l < \bar{d} < l+1$ for some integer constant $l \geqslant 2$ and $\rho_x(l) \in [0,1]$ and $\rho_x(l+1) = 1 - \rho_x(l)$, for each x from S. Let y and z be two arbitrary but fixed elements of S with

$$\begin{aligned} \rho_y(l) &= p & \rho_z(l) &= q \\ \rho_y(l+1) &= 1-p & \rho_z(l+1) &= 1-q\,, \end{aligned}$$

for $p \in [0,1]$ and $q \in [0,1]$. We would like to know if the matching probability increases if we change the probability mass functions ρ_y and ρ_z to ρ'_y and ρ'_z, via

$$\begin{aligned} \rho'_y(l) &= p+\varepsilon & \rho'_z(l) &= q-\varepsilon \\ \rho'_y(l+1) &= 1-p-\varepsilon & \rho'_z(l+1) &= 1-q+\varepsilon\,, \end{aligned}$$

for some $\varepsilon > 0$. We note the following.

- ▷ If $p \geqslant q$, i. e., $\Delta_y \leqslant \Delta_z$, this modification would move both means towards the boundary of the interval $[l, l+1]$. Moving a mean beyond the boundary cannot increase the matching probability since this would be a contradiction to Lemma 3.5.3.
- ▷ If $p < q$, i. e., $\Delta_y > \Delta_z$, this modification would move the means towards each other.

As in Lemma 3.5.3, it can be shown that the matching probability increases if and only if

$$\sum_{d_y, d_z \in \{l, l+1\}} \text{Fail}(d_y, d_z) \cdot \rho_y(d_y) \cdot \rho_z(d_z) > \sum_{d_y, d_z \in \{l, l+1\}} \text{Fail}(d_y, d_z) \cdot \rho'_y(d_y) \cdot \rho'_z(d_z)\,.$$

This inequality is equivalent to

$$[-\varepsilon^2 - \varepsilon \cdot (p-q)] \cdot \underbrace{[\text{Fail}(l,l) - 2 \cdot \text{Fail}(l,l+1) + \text{Fail}(l+1,l+1)]}_{=:K} < 0\,, \qquad (3.19)$$

utilizing the symmetry $\text{Fail}(l+1,l) = \text{Fail}(l,l+1)$. Whether there is an ε that makes (3.19) true depends on K, which is independent of y, z and p, q. There are three cases.

$K = 0$. The modifications to ρ_y and ρ_z do not change the failure probability. We will discuss this singularity later.

$K > 0$. Arrange that $p \geqslant q$ (by interchanging y and z if necessary). Then increasing p and decreasing q (moving the means away from each other) increases the success probability.

$K < 0$. Arrange that $p \leqslant q$. If $p = q$, we are at a local maximum of the matching probability, otherwise increasing p and decreasing q (moving the means closer together) increases the success probability.

Unfortunately, using the same method as in Lemmas 3.5.2 and 3.5.3 it is not possible to show that always $K < 0$ or always $K > 0$ happens. To see this, we try to show $K > 0$ which is equivalent to proving that

$$\mathrm{Fail}(l, l) + \mathrm{Fail}(l+1, l+1) > 2 \cdot \mathrm{Fail}(l, l+1)\,. \tag{3.20}$$

As before we only consider the sufficient condition

$$\mathrm{fail}(l, l) + \mathrm{fail}(l+1, l+1) > 2 \cdot \mathrm{fail}(l, l+1)\,. \tag{3.21}$$

This inequality is equivalent to

$$\begin{aligned} &2 \cdot \beta^l - \beta^{2l} + \sum_{j=1}^{r} [(\iota_j + \beta)^l - \beta^l]^2 \\ &+2 \cdot \beta^{l+1} - \beta^{2l+2} + \sum_{j=1}^{r} [(\iota_j + \beta)^{l+1} - \beta^{l+1}]^2 \\ &\quad > 2 \cdot \beta^l + 2 \cdot \beta^{l+1} - 2 \cdot \beta^{2l+1} + 2 \cdot \sum_{j=1}^{r} [(\iota_j + \beta)^l - \beta^l] \cdot [(\iota_j + \beta)^{l+1} - \beta^{l+1}]\,, \end{aligned}$$

where we use the substitutions $\iota_j = i_j/m$ and $\beta = b/m$. Moving the $\sum_j$-terms to the left and the remaining β-terms to the right gives

$$\begin{aligned} \sum_{j=1}^{r} \Big\{ &(\iota_j + \beta)^{2l} - 2 \cdot (\iota_j + \beta)^l \cdot \beta^l + \beta^{2l} + (\iota_j + \beta)^{2l+2} - 2 \cdot (\iota_j + \beta)^{l+1} \cdot \beta^{l+1} \\ &+ \beta^{2l+2} - 2 \cdot [(\iota_j + \beta)^l - \beta^l] \cdot [(\iota_j + \beta)^{l+1} - \beta^{l+1}] \Big\} > \beta^{2l} \cdot (1-\beta)^2\,, \end{aligned}$$

which is, by a straightforward calculation, equivalent to

$$\sum_{j=1}^{r} \left[(\iota_j + \beta)^l \cdot (1 - \iota_j - \beta) - \beta^l \cdot (1-\beta) \right]^2 > \beta^{2l} \cdot (1-\beta)^2\,. \tag{3.22}$$

Inequality (3.22) may hold or may not hold depending on ι_j and β. For example, consider the following events:

- ▷ $\{\beta = 0\}$. — Then (3.22) is true for all $r \geqslant 2$.

▷ $\{r = 2,\ \iota_1, \iota_2 = 1/(2 \cdot l),\ \beta = 1 - 1/l\}$. — Then (3.22) is false.

Note that both events have positive probability.

It follows that there exists graphs $G_{S-\{y,z\}}$ in which (3.21) is true as well as graphs in which (3.21) is false. For each pair or nodes y, z let $K(y, z)$ be the factor K from (3.19) with respect to the graph $G_{S-\{y,z\}}$. It could be possible that there are nodes y_1, z_1 with $K(y_1, z_1) < 0$ (their means should be made equal), and nodes y_2, z_2 with $K(y_2, z_2) > 0$ (their means should be moved away from each other). So hypothetically, it could be optimal when S is subdivided into three disjoint sets S_l, S_{l+1}, and $S_{l,l+1}$ where each node from S_l has fixed degree l, each node from S_{l+1} has fixed degree $l + 1$, and each node from $S_{l,l+1}$ has a degree that is concentrated on l and $l + 1$ with the same mean $\Delta \in (l, l + 1)$. But this would mean if we assume such an "optimal situation" and we have three different nodes, say y_1, y_2 and z, where $y_1, y_2 \in S_{l,l+1}$ and $z \in S_l$, then $K(y_1, z) > 0$ and $K(y_1, y_2) < 0$, which seems unlikely since $S - \{y_1, z\}$ and $S - \{y_1, y_2\}$ differ in only one node. (Case $K = 0$ does not seem plausible, either.) Therefore we conjecture that it is optimal if one of the following two cases holds.

(F) We have $S = S_l \cup S_{l+1}$, that is, for each x from S the mean Δ_x is fixed to one of the interval borders l and $l + 1$, and therefore a fixed fraction of $\lceil \bar{d} \rceil - \bar{d}$ of the nodes have degree l (assuming that $\bar{d} \cdot n$ is an integer).

(B) We have $S = S_{l,l+1}$, that is, $\bar{d} = \Delta_x$ for each x from S, and therefore the number of nodes of degree l follow a binomial distribution with parameters n and $\lceil \bar{d} \rceil - \bar{d}$.

For the rest of the discussion we only focus on these two degree distributions (fixed and binomial) and we try to argue under which conditions (F) is optimal and when (B) is optimal.

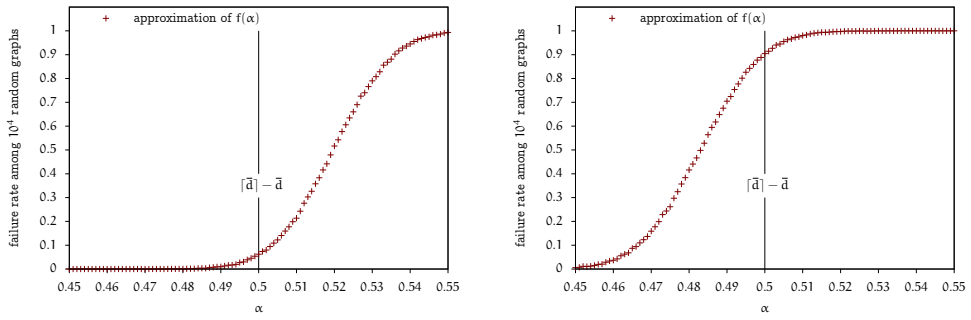
Again our starting point is (3.20). Now fix the degree of all left nodes from G_S and let α be the fraction of nodes from S with degree l and let α' be the fraction of nodes from $S - \{y, z\}$ with degree l. Then there are three situations to distinguish according to the degrees of y and z.

(i) $\alpha = \alpha' + 2/n$, that is y and z have degree l,

(ii) $\alpha = \alpha' + 1/n$, that is one node has degree l the other node has degree $l + 1$,

(iii) $\alpha = \alpha'$, that is both nodes have degree $l + 1$.

Inequality (3.20) states that the increase of the failure probability from (ii) to (i) is larger than the increase of the failure probability from (iii) to (ii) for all α' from $[0, 1]$, that is, the failure probability as a function of α should be convex (while strictly monotonically increasing). Experimental results as shown in Figure 3.5.6 suggest that this is not the case in general. In fact there are three notable situations for fixed $\bar{d}$, two of them shown in Figures 3.5.6 (a) and 3.5.6 (b). Let $f(\alpha)$ denote the failure probability as a function of α.



(a) $c < \hat{c}(\bar{d})$, $c = 0.956$ (b) $c > \hat{c}(\bar{d})$, $c = 0.958$

Figure 3.5.6.: Rate of pseudorandom bipartite graphs G_S with $m = 10^5$ right nodes, $c \cdot m = n$ left nodes, $\alpha \cdot n$ left nodes of degree 3, and $(1-\alpha) \cdot n$ left nodes of degree 4 that have no left-perfect matching, as a function of α. The reference average mean degree is $\bar{d} = 3.5$. The plots show that the failure function $f(\alpha)$ has probably a transition from convex to concave. The theoretical threshold $\hat{c}(3.5)$ is approximately 0.957.

- ▷ If $c < \hat{c}(\bar{d})$, then f is convex in a neighborhood of $\lceil \bar{d} \rceil - \bar{d}$. By Jensen's inequality it follows that the failure probability for fixed degree distribution $f(\lceil \bar{d} \rceil - \bar{d})$ is smaller than the failure probability under the binomial distribution

$$\sum_{i=0}^{n} f(i/n) \cdot \binom{n}{i} \cdot (\lceil \bar{d} \rceil - \bar{d})^i \cdot (1 - \lceil \bar{d} \rceil + \bar{d})^{n-i},$$

  ignoring the right tail of the binomial distribution that reaches the concave part of $f(\alpha)$, since the tail covers only an exponentially small probability mass.
- ▷ If $c > \hat{c}(\bar{d})$, then f is concave in a neighborhood of $\lceil \bar{d} \rceil - \bar{d}$ and the binomial degree distribution leads to a smaller failure probability than the fixed degree distribution.
- ▷ If $c = \hat{c}(\bar{d})$ and $\lceil \bar{d} \rceil - \bar{d} = 0.5$, then, assuming that f is symmetric relative to its point of inflection $\left(\lceil \bar{d} \rceil - \bar{d}, f(\lceil \bar{d} \rceil - \bar{d})\right)$, we have that the binomial distribution and the fixed degree distribution lead to the same failure probability. Moreover, under these constraints each mixed distribution where for some nodes the degree is concentrated on l and $l+1$ and for the rest of the nodes the degree is fixed to l or $l+1$ leads to $K = 0$, the singularity mentioned above.

In order to back up this observation, an additional experiment was done which directly compares the failure rates for degree distributions around the threshold. The results are shown in Figure 3.5.7. Although the differences are small they confirm the conjecture that if $c < \hat{c}(\bar{d})$, then the fixed degree distribution is optimal (F), and if $c > \hat{c}(\bar{d})$, then the binomial degree distribution is optimal (B). The difference is most pronounced for

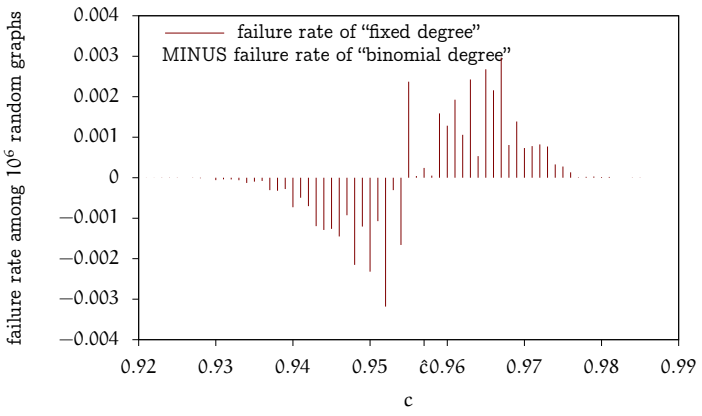


Figure 3.5.7.: Difference of the failure rates of pseudorandom graphs $G^{\bar{d}}_{n,m}((\rho_x)_{x\in S})$ with $D_x \in \{3,4\}$, $\bar{d} = 3.5$, $m = 10^3$, and different $(\rho_x)_{x\in S}$, as a function of the load factor $c = n/m$. Minuend is the failure rate using fixed degree, i. e., $\rho_x(3) \in \{0,1\}$ for each $x \in S$. Subtrahend is the failure rate using binomial degree distribution, i. e., $\rho_x(3) = 0.5$ for each $x \in S$. The theoretical threshold $\hat{c}(3.5)$ is approximately 0.957.

edge densities close to the threshold, where the failure probabilities are not too close to 0 or 1.

3.5.5. Asymptotic Behavior and Thresholds

In this section we prove Proposition 3.1. Let $(\rho_x)_{x\in S}$ be an arbitrary but fixed *near optimal sequence* of degree distributions with $\alpha = (1/n) \cdot \sum_{x\in S} \rho_x(\lfloor \bar{d} \rfloor) \in (0,1]$, for

$$\bar{d} = \alpha \cdot \lfloor \bar{d} \rfloor + (1-\alpha) \cdot \lceil \bar{d} \rceil > 2\,.$$

A concrete *near optimal sequence* that fulfills these constraints is $(\rho'_x)_{x\in S}$ with $\rho'_x(\lfloor \bar{d} \rfloor) = \alpha$ for all $x \in S$. To shorten notation, we let $l = \lfloor \bar{d} \rfloor$. Note that $l+1 = \bar{d} + \alpha$.

We consider the following random bipartite graphs with m right nodes and $n = c \cdot m$ left nodes:

$$G := G^{\bar{d}}_{n,m}((\rho_x)_{x\in S}) \text{ of type A and } G' := G^{\bar{d}}_{n,m}((\rho'_x)_{x\in S}) \text{ of type B.}$$

In $G = G(\bar{d})$ each left node x has $D_x \in \{l, l+1\}$ random right neighbors, chosen with replacement, where D_x is distributed according to ρ_x. The expected average degree of a left node in G is $\bar{d}$. In graph $G' = G'(\bar{d})$ the degrees of the left nodes are chosen independently as l and $l+1$ with probabilities α and $1-\alpha$, respectively, and neighbors

are chosen at random without replacement. About G' Theorem 3.3 tells us that there is a constant threshold $\hat{c}(\bar{d}) < 1$, such that for any constant $\varepsilon > 0$ we have the following: If $c \leqslant \hat{c}(\bar{d}) - \varepsilon$, then asymptotically almost surely G' has a matching, and if $c \geqslant \hat{c}(\bar{d}) + \varepsilon$, then asymptotically almost surely G' has no matching. We will show that the same holds for graph G.

Our first step is to define auxiliary random bipartite graphs $\tilde{G} = \tilde{G}(n_1, \ldots, n_l, n_{l+1})$, where $n = n_1 + \ldots + n_{l+1}$. Such a graph has n left nodes and m right nodes, with n_j nodes of degree j, $j = 1, \ldots, l+1$. The neighbors of each left node are chosen uniformly at random without replacement, independently of the other nodes. We show that asymptotically almost surely G equals some $\tilde{G}(n_1, \ldots, n_l, n_{l+1})$ if we ignore multiple edges, where n_l and n_{l+1} are close to $\alpha \cdot n$ and $(1-\alpha) \cdot n$, respectively, and $n_1 + \ldots + n_{l-1}$ is very small.

> Lemma 3.5.4 (Concentration Bound). For $d \in \{l, l+1\}$ let Y^d be the number of left nodes of $G(\bar{d})$ with exactly d pairwise distinct neighbors. Then for any constant $\delta \in (1/2, 1)$ we have
>
> $$\Pr\left(|n \cdot \alpha - Y^l| \leqslant n^\delta \ \wedge\ |n \cdot (1-\alpha) - Y^{l+1}| \leqslant n^\delta\right) = 1 - O\left(e^{-n^{2\cdot\delta-1}}\right),$$
>
> and the number of nodes with fewer than l neighbors is $\mathrm{Bin}\left(n, \Theta(1/m)\right)$-distributed.

We will prove the lemma in Section 3.5.5.1 below, employing a standard version of the Azuma–Hoeffding tail bound.

For technical reasons we also need to consider a variant of model G', denoted by G'': In graph $G'' = G''(\bar{d})$ for $O(\log n)$ left nodes matching edges have been fixed; the neighborhood of the other nodes is determined as in G'. Let $\varepsilon > 0$ be an arbitrary constant. By a straightforward modification of the arguments in [DGM+10, Sections 2 and 4] one sees that still for $c \leqslant \hat{c}(\bar{d}) - \varepsilon$ asymptotically almost surely G'' has a matching.

Now we are ready to prove Proposition 3.1 under the assumption of Lemma 3.5.4.

Case $c \leqslant \hat{c}(\bar{d}) - \varepsilon$. Consider graph G. By Lemma 3.5.4 we see that with probability $1 - O(1/m)$ ignoring multiple edges will give a graph $\tilde{G}(n_1, \ldots, n_l, n_{l+1})$ with $n_l = \alpha \cdot n \cdot \left(1 \pm o(1)\right)$ and $n_{l+1} = (1-\alpha) \cdot n \cdot \left(1 \pm o(1)\right)$, and the number of left nodes with degree smaller than l is $O(\log n)$. Moreover, with probability $1 - O\left((\log n)^2/n\right)$, we will find a matching for these nodes. (This is true even if $n_1 > 0$, which might well be the case especially if $l = 2$.) This leads to a graph of type $G''(\bar{d})$, which has a matching asymptotically almost surely, as noted above.

Case $c \geqslant \hat{c}(\bar{d}) + \varepsilon$. Consider the subgraph $G_{l,l+1}$ of G that only contains those left nodes that have l or $l+1$ pairwise distinct neighbors and ignores multiple edges. By Lemma 3.5.4, with high probability this subgraph will be a graph $\tilde{G}(0, \ldots, 0, n_l, n_{l+1})$ with $n_l + n_{l+1} = \left(c - o(1)\right) \cdot m$ and average degree $\bar{d} \pm o(1)$. This means that $G_{l,l+1}$ is a random graph of type $G'\left(\bar{d} \pm o(1)\right)$. Applying [DGM+10, Theorem 3] we get that

asymptotically almost surely $G_{l,l+1}$ does not have a matching, so G cannot have a matching either.

In order to complete the proof of Proposition 3.1 it only remains to show Lemma 3.5.4, which is done in the following section.

3.5.5.1. Concentration Bound

In this section we prove Lemma 3.5.4. First, using a standard concentration bound we will show that with high probability the number of left nodes of G that have l and $l+1$ pairwise distinct neighbors is concentrated around $\alpha \cdot n$ and $(1-\alpha) \cdot n$, respectively. Recall that $l = \lfloor \bar{d} \rfloor$.

For each d and each x from S let Y_x^d be a binary random variable with $Y_x^d = 1$ if the neighborhood set N_x of x has size d, and $Y_x^d = 0$ if N_x has size strictly smaller than d. Furthermore, let $Y^d = \sum_{x \in S} Y_x^d$. Then we have

$$\begin{aligned} \mathrm{Exp}(Y_x^{l+1}) &= \big(1 - \rho_x(l)\big) \cdot \frac{\binom{m}{l+1} \cdot (l+1)!}{m^{l+1}}, \text{ and} \\ \mathrm{Exp}(Y_x^{l}) &= \rho_x(l) \cdot \frac{\binom{m}{l} \cdot l!}{m^{l}} + \big(1 - \rho_x(l)\big) \cdot \frac{\binom{m}{l} \cdot l! \cdot \binom{l+1}{2}}{m^{l+1}}. \end{aligned} \tag{3.23}$$

Consider the events

$$\begin{aligned} \mathcal{A} &:= \{n \cdot \alpha - n^\delta \leqslant Y^l \leqslant n \cdot \alpha + n^\delta\} \text{ and} \\ \mathcal{B} &:= \{n \cdot (1-\alpha) - n^\delta \leqslant Y^{l+1} \leqslant n \cdot (1-\alpha) + n^\delta\}, \end{aligned}$$

stating that the number of left nodes of G with neighborhood size l and $l+1$ is near $n \cdot \alpha$ and $n \cdot (1-\alpha)$, respectively.

We want to bound the probability of the event $\mathcal{A} \cap \mathcal{B}$ from below using the complementary event $\overline{\mathcal{A}} \cup \overline{\mathcal{B}}$, via

$$\Pr(\overline{\mathcal{A}} \cup \overline{\mathcal{B}}) \leqslant \Pr(\overline{\mathcal{A}}) + \Pr(\overline{\mathcal{B}}).$$

First consider the event $\overline{\mathcal{A}}$. Let $Y_x = Y_x^l$ and $Y = Y^l$ as well as $p_x = \Pr(Y_x = 1)$. According to (3.23) it holds that

$$p_x = \mathrm{Exp}(Y_x) = \rho_x(l) \cdot \big(1 - \Theta(1/m)\big) + \Theta(1/m),$$

since we have $1 - l^2/(2 \cdot m) < \binom{m}{l} \cdot l!/m^l < 1 - 1/m$.

For each $x \in S$ let $Z_x = Y_x - p_x$. Now fix an arbitrary order of the left nodes, i. e., let $S = \{x_1, x_2, \ldots, x_n\}$. It holds that $X_0, X_1, \ldots, X_n$ with $X_0 = 0$ and $X_i = X_{i-1} + Z_{x_i}$ is a martingale with bounded differences, since

$$\mathrm{Exp}(X_{i+1} \mid X_0, \ldots, X_i) = \mathrm{Exp}(X_i + Z_{x_{i+1}} \mid X_0, \ldots, X_i) = X_i$$

and $-p_{x_{i+1}} \leqslant X_{i+1} - X_i \leqslant 1 - p_{x_{i+1}}$. Applying a standard Azuma–Hoeffding inequality [DP09, Theorem 5.1], we get

$$\Pr\left(|X_n - X_0| \geqslant n^\delta/\sqrt{2}\right) \leqslant 2 \cdot e^{-2\cdot(n^\delta/\sqrt{2})^2/n} = 2 \cdot e^{-n^{2\cdot\delta-1}} .$$

Since $|Y - \mathrm{Exp}(Y)| = |X_n - X_0|$, this means that the number of left nodes with a neighborhood set of size l differs more than $n^\delta/\sqrt{2}$ from its expected value only with exponentially small probability. By linearity of expectation, we have

$$\mathrm{Exp}(Y) = \sum_{x\in S} p_x = \big(1 - \Theta(1/m)\big) \cdot \sum_{x\in S} \rho_x(l) + \Theta(1) .$$

Since $\alpha = (1/n) \cdot \sum_{x\in S} \rho_x(l)$, it follows that $\mathrm{Exp}(Y) = n \cdot \alpha \pm \Theta(1)$. This means that the inequality $|\mathrm{Exp}(Y) - n \cdot \alpha| \leqslant n^\delta \cdot (1 - 1/\sqrt{2})$ holds for all but a constant number of n. Thus, the probability of event $\overline{\mathcal{A}}$ is exponentially small in n. Essentially the same proof shows an exponentially small bound for $\overline{\mathcal{B}}$. Hence, the event $\Pr(\mathcal{A} \cap \mathcal{B})$ occurs with high probability. This finishes the proof of the first claim in Lemma 3.5.4.

The proof of the second claim is much simpler. We have seen in (3.23) that the probability that a node x with degree $d = l + 1$ has l neighbors is $O(1/m)$. Similarly, one sees that the probability that a node with degree $d \in \{l, l+1\}$ has fewer than l distinct neighbors is $O(1/m)$. Assume this probability is p. By independence, the number of nodes with fewer than l distinct neighbors follows the binomial distribution $\mathrm{Bin}(n, p)$. This concludes the proof of Lemma 3.5.4 and also completes the proof of Proposition 3.1. ■

3.6. Minimize the Number of Page Accesses for Cuckoo Hashing with Pages

In this section we discuss our results on d-ary cuckoo hashing in a setting where the table is subdivided into pages of equal size — they appeared in [DMR11a, DMR11b]. We give experimental evidence for Conjecture 3.1.3, which states that even with small pages it is sufficient if each key is associated with only two pages, a *primary* and a *backup page*, in order to obtain the same load thresholds as for normal d-ary cuckoo hashing. Furthermore, our experiments show that one can efficiently force keys to their primary page in order to reduce the number of page accesses for lookup to 1 for most of them, without reducing the maximum load.

3.6.1. Graph Model

We study graphs $G^{d_p,d_b}_{n,(m,\check{m})}$ of type B, which are random bipartite graphs with left node set $L = S$ and right node set $R = [m]$; the left nodes correspond to keys, the right nodes correspond to table cells. The set R is subdivided into p segments $R_0, R_1, \ldots, R_{p-1}$, each

of size $\check{m} = \lfloor m/p \rfloor$, which correspond to the separate pages (sub-tables). We assume that m is divisible by p in the following. Each left node x is incident to $d = d_p + d_b$ edges where its neighborhood $N(\{x\})$ consists of two disjoint sets $N_p(\{x\})$ and $N_b(\{x\})$ determined according to the following scheme (all choices are fully random):

1. choose p from $[p]$, the index of the primary page;
2. choose d_p different right nodes from R_p to build the set $N_p(\{x\})$;
3. choose b from $[p] - \{p\}$, the index of the backup page;
4. choose d_b different right nodes from R_b to build the set $N_b(\{x\})$.

Let $e = \{x, y\}$ be an edge where $x \in L$ and $y \in R$. We call e a *primary edge* if $y \in N_p(\{x\})$ and call e a *backup edge* if $y \in N_b(\{x\})$.

3.6.2. Problem Description

First, we want to find parameters $\check{m}$, d_p, and d_b such that even with high load factor $c = n/m$ the graph $G^{d_p,d_b}_{n,(m,\check{m})}$ is likely to have a left-perfect matching.

Remark. As mentioned in Section 3.1.3, if $d_p = d$, $d_b = 0$ and page sizes are $\check{m} = m^\delta$ for constant $\delta > 0$, then the asymptotic load factor threshold is the same as in the setting without pages, see Table 3.1.1. This is easily proven using tight concentration bounds on the number of keys per page. Our interest, however, is in ranges for m that are realistic and not too large page sizes $\check{m}$, so that this asymptotic behavior is not an adequate description of performance.

For a given placement let n_p be the *number of keys that are assigned to their primary page*, and let n_b be the *number of keys that are assigned to their backup page*. Second, we are interested in the potential for saving page accesses on lookup due to finding left-perfect matchings such that the fraction n_p/n is maximized. Appropriate algorithms are presented in the next section.

3.6.3. Algorithms

Each matching M with respect to a bipartite graph $G = (L \cup R, E)$ implies an *orientation* of the edges according to:

- ▷ Matching edges are directed from right to left.
- ▷ Non-matching edges are directed from left to right.

We call an orientation of the edge set (k, ℓ)-orientation, if the indegree of all left nodes is exactly k, the outdegree of all right nodes is at most ℓ, and for each left node we have that the incoming edges originate from pairwise distinct right nodes. This is exactly the same as the (k, ℓ)-orientation with respect to hypergraphs as discussed

in Section 3.1.1.5. Hence, the problem of finding a left-perfect matching in G, and especially $G^{d_p,d_b}_{n,(m,\check{m})}$, can be seen as problem of finding a $(1,1)$-*orientation* of the graph. A left node whose primary edge is a matching edge is called *primary key*. A left node whose backup edge is a matching edge is called *backup key*. Each $(1,1)$-orientation or left-perfect matching that has a *maximum number of primary keys* is called *optimal*.

3.6.3.1. Static Case

The problem of finding an optimal left-perfect matching can be easily reformulated as *minimum weight matching* problem or *minimum cost flow* problem, see, e. g., [AMO93, Section 12.4 and Chapters 9–11]. Hence, in the static case, i. e., if a graph G is given in advance, there are well-known efficient (but not linear time) algorithms for finding such a matching. We propose Algorithm 5, a *modified Hopcroft-Karp algorithm* for determining optimal left-perfect matchings in random graphs $G^{d_p,d_b}_{n,(m,\check{m})}$ — a detailed description follows below.

Minimum Cost Flow To obtain a corresponding minimum cost flow problem direct all edges of $G = (L \cup R, E)$ from left to right. Add a source node s and a target node t to G. Furthermore for each node x from L add an edge (s,x) and for each node y from R add an edge (y,t) to the graph. Let E_{cf} be the modified edge set. A cost function

$$\omega_{cf} \colon E_{cf} \to \{0,1\}$$

assigns each primary edge cost 0, each backup edge cost 1, and edges incident to the source or target node cost 0. All edges get capacity 1. The supply of s and the demand of t is n, where the supply and demand of all other nodes is 0. Now the problem is to find an *integral minimum cost flow of value* n, i. e., a flow $f \colon E_{cf} \to \{0,1\}$ which assigns each edge $(u,v) \in E_{cf}$ a value $f(u,v)$ such that the total cost

$$\sum_{(u,v)\in E_{cf}} \omega_{cf}(u,v) \cdot f(u,v)$$

is minimized under the constraints that the flow of each edge does not exceed its capacity, the outgoing flow of the source node $\sum_{x\in L} f(s,x)$ and the incoming flow of the target node $\sum_{y\in R} f(y,t)$ are both n, for all other nodes v the incoming flow $\sum_{u\in N(\{v\})} f(u,v)$ equals the outgoing flow $\sum_{w\in N(\{v\})} f(v,w)$.

Minimum Weight Matching Another possibility is to consider a corresponding *minimum weight matching problem*, see Section 3.3. Analogously as above, we use a weight (cost) function

$$\omega \colon E \to \{0,1\}$$

in order to assign each primary edge the weight 0 and each backup edge the weight 1. Now the problem of finding an optimal left-perfect matching translates into finding a left-perfect matching M with minimum weight

$$\omega^+(M) = \sum_{e \in M} \omega(e)\,.$$

This can be done via successively finding augmenting paths that have smallest incremental weight, see Section 3.3. The incremental weight of an augmenting path P with respect to some matching M, is

$$\omega^{\pm}_M(P) = \sum_{e \in P - M} \omega(e) - \sum_{e \in M \cap P} \omega(e)\,.$$

Our Algorithm 5 is a variant of the *successive shortest path algorithm*, see, e. g., [AMO93, Section 9.7 and page 471], but in order to determine augmenting paths with smallest incremental weight, it relies on a modification of the *Hopcroft-Karp algorithm*, i. e., a combination of breadth and depth first search, instead of the *Moore-Bellman-Ford algorithm* or *Dijkstra's algorithm*.

Remark. Results on the length of augmenting paths in sparse random bipartite graphs [BMST04, BMST06] indicate that on input $G^{d_p,d_b}_{n,(m,\check{m})}$, with load factor and left degree upper bounded by constants, w. h. p. the running time of the Hopcroft-Karp algorithm is $O(n \cdot \log n)$.

For the following detailed description of our algorithm, we use the view that the edges of G are directed, initially with respect to the empty matching, i. e., all edges are non-matching edges and therefore go from left to right. The algorithm works in rounds. In each round, w_{lo} is a lower bound for the smallest incremental weight of any augmenting path. According to our weight function, w_{lo} is initially zero. Consider some fixed round. With the combination of a modified *breadth first search* (BFS) and *depth first search* (DFS) a maximal set of augmenting paths, i. e., directed paths that start with an unmatched node from L and end with an unmatched node from R, that have the following properties is found:

- ▷ The paths have incremental weight exactly w_{lo}.
- ▷ They are vertex disjoint.
- ▷ All have the same number of edges and the number of edges is minimal.

The BFS starts from all left nodes with indegree zero, i. e., unmatched nodes. The search partitions the nodes into layers. For each explored node the layer and the weight of the path to this node are stored. A node can be re-explored if it is reached by a path of smaller weight. The BFS stops at the first level where one or more unmatched nodes of R are reached by a path of weight exactly w_{lo}. Let R_w be the set of these right nodes. If R_w is not empty, then DFS is applied to each node $y \in R_w$ to find node

ALGORITHM 5: `min_weight_Hopcroft-Karp`

Input : Bipartite Graph $G = (L \cup R, E)$ and weight function $\omega\colon E \to \mathbb{Z}$.
Output : Matching with maximum number of left nodes that has minimum weight.
Require : Modified breadth-first search and depth-first search algorithms as described below. We interpret G as directed graph according to Section 3.6.3, starting with all edges directed from left to right.

`target_weight_BFS`(G, w_{lo}):

- ▷ Partition the nodes of G into layers $0, 1, 2, \ldots$ starting from the unmatched left nodes.
- ▷ Stop at the smallest layer l such that there is a path of weight w_{lo} through consecutive layers $0, 1, \ldots, l$ to an unmatched right node.
  - ▷ If such a layer exists, then all unmatched right nodes at this layer that have such a path are collected in R_w, and (true, R_w) is returned.
  - ▷ Otherwise, if there is only a path of higher weight, then $(\text{true}, \emptyset)$ is returned; else $(\text{false}, \emptyset)$ is returned.

`target_weight_DFS`(G, y, w, w_{lo}):

- ▷ Do a recursive descent, starting with y, through the layers of G given by `target_weight_BFS`(G, w_{lo}).
- ▷ The weight of the current path is given via w.
- ▷ If an unmatched left node is reached and $w = w_{lo}$, then the recursive ascent flips the edge orientations along the path.

foreach $e \in E$ do direct e from left to right;
$w_{lo} \leftarrow \min_{e \in E} \omega(e)$;
while true do
  $(\mathit{somePathIsFound}, R_w) \leftarrow$ `target_weight_BFS`(G, w_{lo});
  if $R_w \neq \emptyset$ then
    foreach $y \in R_w$ do `target_weight_DFS`$(G, y, 0, w_{lo})$;
  else
    if *somePathIsFound* then $w_{lo} \leftarrow w_{lo} + 1$;
    else break;
  end
end
return $\{e \in E \mid e \text{ is directed from right to left}\}$;

disjoints paths between R_w and the unmatched left nodes, where the search can only follow edges between two successive layers. During the recursive descent the weight of a path is accumulated. If the DFS reaches a unmatched node and the weight of the path is exactly w_{lo}, the recursive descent stops and the recursive ascent flips the orientations along the path. After all nodes from R_w are covered by the DFS, the next round starts. If no augmenting path with incremental weight w_{lo} is found but there is an augmenting path of larger incremental weight, then w_{lo} is increased by 1. If there is no augmenting path at all, then the algorithm stops and returns all matching edges, i. e., edges that are directed from right to left. The algorithm is optimal in the sense that it successively finds augmenting paths with smallest incremental weight and therefore returns a maximum cardinality matching of minimum weight. The crucial insight is that, according to Lemma 3.3.4, the smallest incremental weight can only increase and since all edge weights are integers, w_{lo} remains a lower bound for this value in each round.

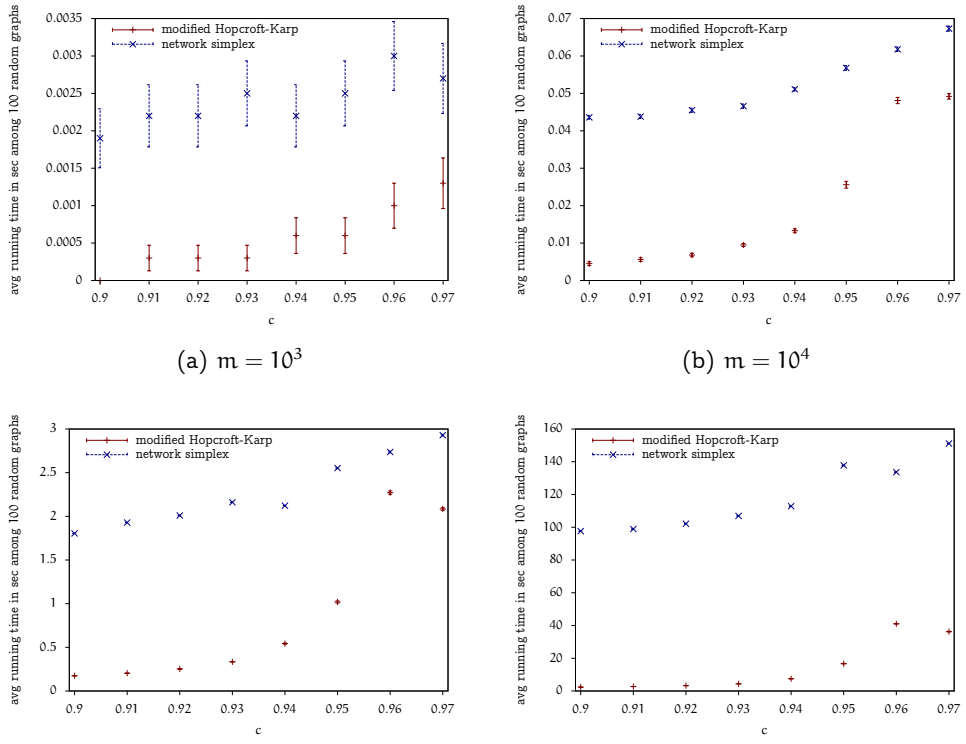
Experimental Comparison In experiments we compared the running time of our algorithm with the running time of several standard algorithms for solving the corresponding minimum cost flow problem made available via the *LEMON Graph Library* [LEM11] as part of a recent comparative study [KK10]. We found that for our pseudorandom graphs $G^{d_p,d_b}_{n,(m,\check{m})}$ (problem instances) *network simplex* was the fastest algorithm among the algorithms provided by the library (network simplex, cost scaling, capacity scaling, cycle-canceling), which is consistent with the experimental results in [KK10, Section 3]. However, this algorithm was clearly outperformed by our modified Hopcroft-Karp algorithm, which became our algorithm of choice for investigating the potential of cuckoo hashing with paging. Figure 3.6.8 gives an exemplary comparison of the running times of both algorithms.

3.6.3.2. Dynamic Case

In the online scenario the graph G initially consists only of the right nodes. To begin, let us consider the case of *insertions* only. The keys arrive and are inserted one by one, and with each new key the graph grows by one left node and d edges. To find an appropriate orientation of the edges in each insertion step, we apply Algorithm 6, which is a modification of the common random walk for d-ary cuckoo hashing [FPSS05] but with two additional constraints:

1. avoid creating backup keys at the beginning of the insertion process, and
2. keep the number of backup keys below a small fixed fraction.

For the description of the algorithm we use a dual approach. Algorithm 6 refers to orienting the graph G and the following explanation uses the view of placing keys into table cells.



(a) $m = 10^3$ (b) $m = 10^4$

(c) $m = 10^5$ (d) $m = 10^6$

Figure 3.6.8.: Comparison of the average running times of the modified Hopcroft-Karp algorithm and the network simplex algorithm for graphs $G^{d_p,d_b}_{n,(m,\check{m})}$ with $\check{m} = 100$, $d_p = 3$, $d_b = 1$, and different n and m. The values $\overline{\mathcal{T}}$ are averages over 100 runs each. The error bars depict the *estimated standard error of the mean* $\sqrt{s^2[\mathcal{T}]/100}$.

ALGORITHM 6: `random_walk_basic_step`

Input : Directed bipartite graph $G = (L \cup R, E)$ and node $x \in L$. All edges incident with x are directed from left to right. The graph induced by $(L - \{x\}) \cup R$ has a $(1,1)$-orientation.

Purpose : Direct all edges of G such that G gets a $(1,1)$-orientation

Output : (G, true) if a $(1,1)$-orientation is found within the maximum number of total steps left given by *globalCounter*; otherwise (G, false).

Require : Random number generator `randomNumber()` that gives realizations of independent uniformly distributed random variables with support $[0,1]$.

```
success ← false;
while globalCounter > 0 and not success do
    if ∃y ∈ N_p({x}) with out-deg(y) < 1 then
        flip edge (x, y); success ← true;
    end
    if not success then
        if randomNumber() < β then
            choose random y ∈ N_p({x}); choose random x' ∈ N({y});
            flip edge (x, y); flip edge (y, x'); x ← x';
        else
            if ∃y ∈ N_b({x}) with out-deg(y) < 1 then
                flip edge (x, y); success ← true;
            else
                choose random y ∈ N_b({x}); choose random x' ∈ N({y});
                flip edge (x, y); flip edge (y, x'); x ← x';
            end
        end
    end
    globalCounter ← globalCounter − 1;
end
return (G, success);
```

(*The modification to avoid unnecessary back steps is not shown for the sake of clarity.*)

We refer to the d_p cells of a key that are on its primary page as *primary positions*, and we call the d_b cells on its backup page *backup positions*. The insertion of an arbitrary key x takes one or more *basic steps* of the random walk, which can be separated into the following sub-steps.

Let x be the key that is currently "nestless", i. e., x is not stored in the memory. First check if one of its primary positions is free. If this is the case store x in such a free cell and stop successfully. Otherwise toss a biased coin to decide whether the insertion of x should be proceed on its primary page or on its backup page.

- ▷ If the insertion of x is restricted to the primary page, randomly choose one of its primary positions y. Let x' be the key which is stored in cell t_y. Store x in t_y, replace x with x', and start the next step of the random walk.
- ▷ If x is to be stored on its backup page, first check if one of the backup positions of x is free. If this is the case store x in such a free cell and stop successfully. Otherwise randomly choose one of the backup positions y on this page and proceed as in the previous case.

The matching procedure is slightly modified to avoid unnecessary back steps. That is, if a key x displaces a key x' and in the next step x' displaces x'' then $x'' = x$ is forbidden as long as x' has another option on this page.

Our random walk algorithm uses two parameters:

- β - the bias of the virtual coin. This influences the fraction of backup keys.
- θ - controls the terminating condition. A global counter is initialized with value $\theta \cdot n$, which is the maximum number of total steps of the random walk summed over all keys. For each basic step the global counter is decremented by one. If the limit is exceeded the algorithm stops with "failure".

Deletions are carried out in a straightforward fashion. To remove a key x, first the primary page is checked for x in its possible cells, and if needed the backup page is checked as well. The cell containing x is marked as empty, which can be interpreted as removing the left node x and its d incident edges from G. The global counter is ignored in this setting ($\theta = \infty$).

3.6.4. Experiments

For each of the following experiments we used *pseudorandom* graphs $G^{d_p,d_b}_{n,(m,\check{m})}$ of type B randomly generated using the Mersenne Twister as in Section 3.4.4. The graphs are created according to some configuration $(c, m, \check{m}, d_p, d_b)$, where c is the quotient of left nodes (keys) and right nodes (table cells), m is the total number of right nodes, $\check{m}$ is the page size, and d_p, d_b are the number of primary and backup edges of each left node. In the implementation the left and right nodes were simply the number sets

$[n]$ and $[m]$. If not stated otherwise the total number of cells is $m = 10^6$ and pages are of size $\breve{m} = 10^i$ for $1 \leqslant i \leqslant 6$.

We restrict ourselves to the cases $d_p = 3, d_b = 1$ and (just for comparison) $d_p = 4$ and $d_b = 0$. While we have done experiments with other parameter values, we believe these settings are sufficient to illustrate the main points. Also, while we have computed sample variances, in many cases they are small; this should be assumed when they are not discussed.

3.6.4.1. Static Case

Experimental results for the static case determine the limits of our approach and serve as a basis of comparison for the dynamic case.

Setup and Measurements We focus on the following points.

I. Our aim is to get approximations for *possible threshold densities*, where first of all we want to see the limits of cuckoo hashing with pages if there are no backup options at all.

Remark. Note that for constant page size $\breve{m}$ and larger and larger table size m the fraction of keys that can be placed decreases. If $c = n/m$ is constant too, the load of each page is approximately Poisson distributed with parameter $c \cdot \breve{m}$; asymptotically the success probability can be estimated as

$$O\Big(\Big(\Pr\big(\mathrm{Po}[c \cdot \breve{m}] \leqslant \breve{m}\big)\Big)^p\Big) = O\Big(\Big(\sum_{i=0}^{\breve{m}} \frac{(c \cdot \breve{m})^i}{i!} \cdot e^{-c \cdot \breve{m}}\Big)^p\Big),$$

for $p = m/\breve{m}$, which approaches 0 for $m \to \infty$.

In the following, we *assume the existence* of load thresholds $\hat{c}_{\breve{m}}^{d_b}(d_p + d_b)$ that identify a transition from the situation that $G_{n,(m,\breve{m})}^{d_p,d_b}$ is likely to admit a left-perfect matching to the situation that such a matching is unlikely. Accordingly, we denote the *hypothetical threshold* for the case with three primary and no backup option with $\hat{c}_{\breve{m}}^0(4)$, and we denote the *hypothetical threshold* for the case with three primary and one backup option with $\hat{c}_{\breve{m}}^1(4)$. To get approximations for $\hat{c}_{\breve{m}}^0(4)$ and $\hat{c}_{\breve{m}}^1(4)$, we study different ranges of load factors $[c_{\mathrm{start}}, c_{\mathrm{end}}]$. Specifically, analogous to Section 3.4.4, for all c_i where $c_i = c_{\mathrm{start}} + i \cdot 10^{-4} \leqslant c_{\mathrm{end}}$, and $i = 0, 1, 2, \ldots, \lfloor (c_{\mathrm{end}} - c_{\mathrm{start}}) \cdot 10^4 \rfloor$, we construct a random graphs and measure the failure rate ξ_i at c_i. We fit the sigmoid function $\sigma(c; \gamma, \delta)$ (3.3) to the data points (c_i, ξ_i) using the method of least squares. The parameter γ (inflection point) is used as an approximation of $\hat{c}_{\breve{m}}^0(4)$ and $\hat{c}_{\breve{m}}^1(4)$ respectively. With Σ_{sre} we denote the sum of squares of the residuals (3.4).

II. Furthermore, for different c and page sizes $\breve{m}$, we are interested in the maximum ratio $r_p = n_p/n$ or load factor $c_p = n_p/m$ of primary keys, respectively.

III. For an arbitrary but fixed page let $n_{b|\star}$ be the number of keys that have this page as primary page but are inserted on their backup page. Since the number of potential primary keys for a page follows a binomial distribution, some pages will be lightly loaded and therefore have a small value of $n_{b|\star}$ or even $n_{b|\star} = 0$. Some pages will be overloaded and hence have to shed load, yielding a large value of $n_{b|\star}$. We want to estimate the probability $\Pr(n_{b|\star} = j)$ for each $j \in [n]$. Therefore, we generate a many pseudorandom graphs $G^{d_p,d_b}_{n,(m,\breve{m})}$ and number all pages consecutively. Then for each page $i \in [a \cdot p]$ we determine the value $n_{b|i}$ in order to obtain the corresponding relative frequencies

$$\frac{1}{a \cdot p} \cdot \sum_{\substack{i \in [a \cdot p] \\ i\colon n_{b|i} = j}} 1 \quad \text{for all } j \in [n],$$

as well as sample mean

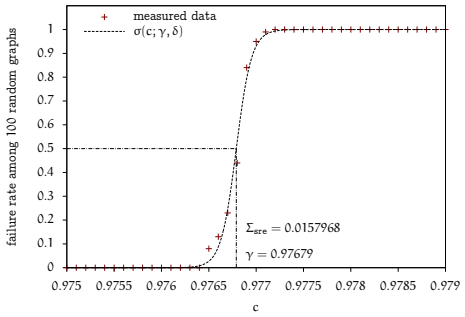
$$\overline{n_{b|\star}} = \frac{1}{a \cdot p} \cdot \sum_{i \in [a \cdot p]} n_{b|i}\,,$$

and sample variance

$$s^2[n_{b|\star}] = \frac{1}{a \cdot p - 1} \sum_{i \in [a \cdot p]} (n_{b|i} - \overline{n_{b|\star}})^2\,.$$

Results Here we consider results from an optimal placement algorithm.

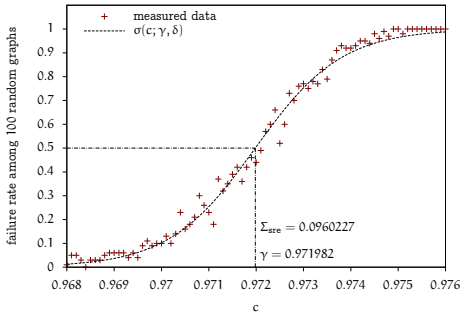
I. Table 3.6.4 gives *approximations of the transition points* where cuckoo hashing with paging and $d = 4$ hash functions has failure rate $\xi = 0.5$ in the case of 1 or 0 backup pages. With no backup pages the number of keys that can be stored decreases with decreasing page size and the success probability around $\hat{c}^0_{\breve{m}}(4)$ converges less rapidly, as demonstrated clearly in Figures 3.6.9 and 3.6.10. This effect becomes stronger as the pages get smaller. For this reason the range of load factors $[c_{start}, c_{end}]$ of Table 3.6.4 (a) grows with decreasing page size. Using only one backup edge per key almost eliminates this effect. In this case the values $\hat{c}^1_{\breve{m}}(4)$ seem to be stable for varying $\breve{m}$ and are very near to the theoretical threshold of standard 4-ary cuckoo hashing, which is $\hat{c}(4) \approx 0.97677$, see Table 3.1.1; only in the case of very small pages $\breve{m} = 10$ can a minor shift of $\hat{c}^1_{\breve{m}}(4)$ be observed. The position of $\hat{c}^1_{\breve{m}}(4)$ as well as the slope of the fitting function appear to be quite stable for all considered page sizes.



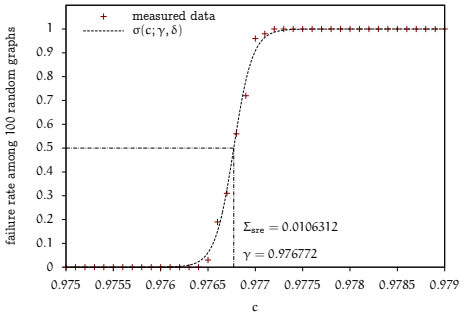
(a) $\check{m} = 10^6$, 41 data points, $d_p = 4$, $d_b = 0$

identical to Figure 3.6.9 (a)

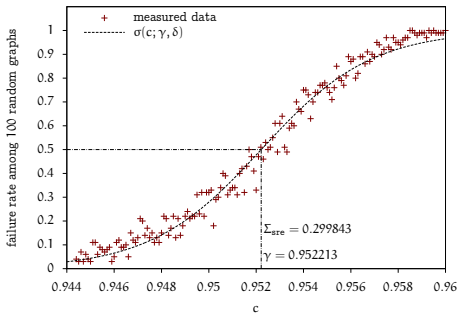
(b) $\check{m} = 10^6$, 41 data points, $d_p = 3$, $d_b = 1$



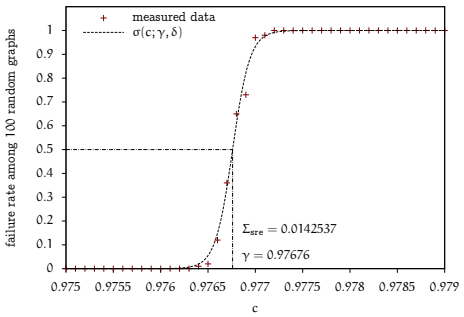
(c) $\check{m} = 10^5$, 81 data points, $d_p = 4$, $d_b = 0$



(d) $\check{m} = 10^5$, 41 data points, $d_p = 3$, $d_b = 1$

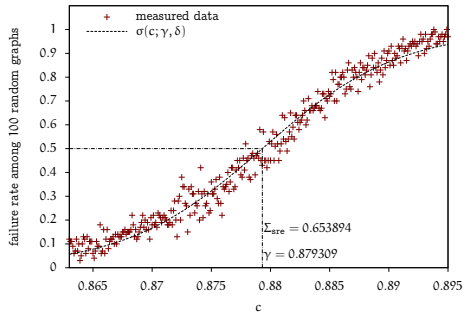


(e) $\check{m} = 10^4$, 161 data points, $d_p = 4$, $d_b = 0$

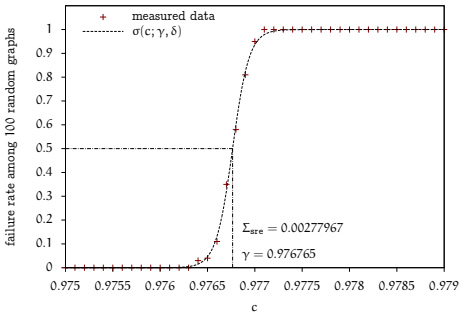


(f) $\check{m} = 10^4$, 41 data points, $d_p = 3$, $d_b = 1$

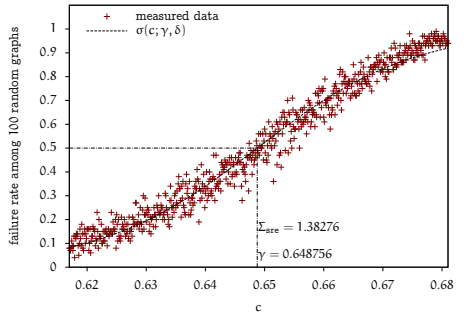
Figure 3.6.9.: Identification of transition points $\hat{c}^0_{\check{m}}(4)$ (left column) and $\hat{c}^1_{\check{m}}(4)$ (right column) for $m = 10^6$ and page sizes $\check{m} = 10^4, 10^5, 10^6$; see Table 3.6.4.



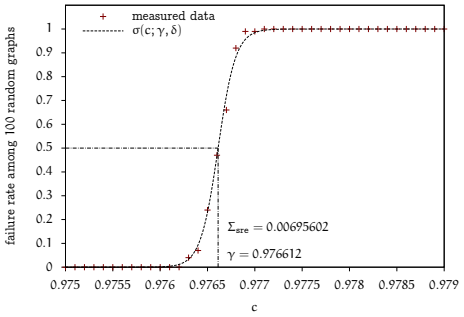
(a) $\check{m} = 10^3$, 321 data points, $d_p = 4$, $d_b = 0$



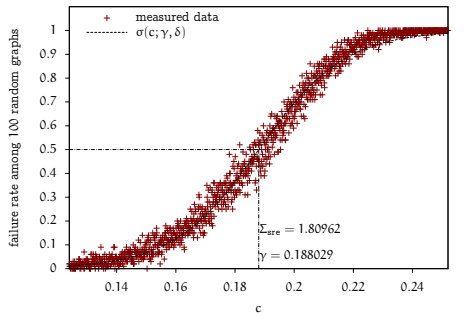
(b) $\check{m} = 10^3$, 41 data points, $d_p = 3$, $d_b = 1$



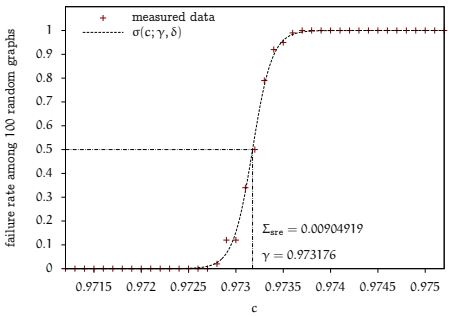
(c) $\check{m} = 10^2$, 641 data points, $d_p = 4$, $d_b = 0$



(d) $\check{m} = 10^2$, 41 data points, $d_p = 3$, $d_b = 1$



(e) $\check{m} = 10^1$, 1281 data points, $d_p = 4$, $d_b = 0$



(f) $\check{m} = 10^1$, 41 data points, $d_p = 3$, $d_b = 1$

Figure 3.6.10.: Identification of transition points $\hat{c}^0_{\check{m}}(4)$ (left column) and $\hat{c}^1_{\check{m}}(4)$ (right column) for $m = 10^6$ and page sizes $\check{m} = 10^1, 10^2, 10^3$; see Table 3.6.4.

$\breve{m}$	$[c_{start}, c_{end}]$	γ	Σ_{sre}
10^6	$[0.975, 0.979]$	0.97679	0.0157968
10^5	$[0.968, 0.976]$	0.971982	0.0960227
10^4	$[0.944, 0.960]$	0.952213	0.299843
10^3	$[0.863, 0.895]$	0.879309	0.653894
10^2	$[0.617, 0.681]$	0.648756	1.38276
10^1	$[0.124, 0.252]$	0.188029	1.80962

(a) $d_p = 4$, $d_b = 0$, $40 \cdot 2^{6-\log_{10}(\breve{m})} + 1$ data points per fit

$\breve{m}$	$[c_{start}, c_{end}]$	γ	Σ_{sre}
10^5	$[0.975, 0.979]$	0.976772	0.0106312
10^4	"	0.97676	0.0142537
10^3	"	0.976765	0.00277967
10^2	"	0.976612	0.00695602
10^1	$[0.9712, 0.9752]$	0.973176	0.00904919

(b) $d_p = 3$, $d_b = 1$, 41 data points per fit

Table 3.6.4.: List of approximations γ of load factors $\hat{c}^0_{\breve{m}}(4)$ (left) and $\hat{c}^1_{\breve{m}}(4)$ (right) that are the midpoints of the transition from failure rate 0 to failure rate 1 (without and with backup option). The values were determined via fitting the function (3.3) to a series of data points (c, ξ), using equidistant c from $[c_{start}, c_{end}]$. The plots are shown in Figures 3.6.9 and 3.6.10.

II. The average of the *maximum fraction of primary keys*, allowing one backup option, is shown in Table 3.6.5. The fraction decreases with increasing load factor c and decreases with decreasing page size $\breve{m}$ as well. The sample variance (not shown in the table) is very small and increases slightly for decreasing page size; the maximum value is about $4 \cdot 10^{-7}$ for $\breve{m} = 10$ and $c = 0.97$. Interestingly, for several parameters, we found that an optimal algorithm finds placements with more than $\hat{c}(3) \cdot m$ keys sitting in one of their 3 primary positions, where $\hat{c}(3) \approx 0.91794$ is the threshold for standard 3-ary cuckoo hashing, see Table 3.1.1. That is, more keys obtain one of their primary three choices with three primary and one backup choice than what could be reached using just three primary choices even without paging.

III. Figure 3.6.11 depicts the *relative frequency* of the values $n_{b|\star}$ for selected parameters $(c, \breve{m}) = (0.95, 10^3)$, among 10^5 pages arising from $a = 10^2$ pseudorandom graphs with $p = 10^3$ pages each. In this case about 17 percent of all pages do not need backup pages, i. e., $n_{b|\star} = 0$. This is consistent with the idea that pages with a load below $\hat{c}(3) \cdot \breve{m}$ will generally not need backup pages. The mean $\overline{n_{b|\star}}$ is about 2.5 percent of the page size $\breve{m}$ and for about 87.6 percent of the pages the value $n_{b|\star}$ is at most 5 percent of the page size. The relative frequency of $n_{b|\star}$ being greater than $0.1 \cdot \breve{m}$ (very right tail) is very small, about $1.1 \cdot 10^{-3}$.

Number of Page Accesses We observed that using pages with $(d_p, d_b) = (3, 1)$ we achieve load factors very close to the $\hat{c}(4)$ threshold, i. e., $\hat{c}^1_{\breve{m}}(4) \approx \hat{c}(4)$. Moreover the load factor c_p concerning the keys placed on their primary page is quite large, near or even above $\hat{c}(3)$.

Let X be the average (over all keys that have been inserted) number of page requests

	$\breve{m} = 10^6$		$\breve{m} = 10^5$		$\breve{m} = 10^4$	
c	$\overline{r_p}$	$\overline{c_p}$	$\overline{r_p}$	$\overline{c_p}$	$\overline{r_p}$	$\overline{c_p}$
0.90	1.000000	0.900000	1.000000	0.900000	0.999881	0.899893
0.91	1.000000	0.910000	0.999997	0.909997	0.999093	0.909175
0.92	0.998412	0.918539	0.998136	0.918286	0.996111	0.916422
0.93	0.990918	0.921554	0.990957	0.921510	0.990467	0.921134
0.94	0.983438	0.924431	0.983443	0.924436	0.983422	0.924416
0.95	0.975982	0.927183	0.975952	0.927154	0.975961	0.927163
0.96	0.968528	0.929787	0.968578	0.929835	0.968524	0.929783
0.97	0.961163	0.932328	0.961112	0.932279	0.961157	0.932323
	$\breve{m} = 10^3$		$\breve{m} = 10^2$		$\breve{m} = 10^1$	
c	$\overline{r_p}$	$\overline{c_p}$	$\overline{r_p}$	$\overline{c_p}$	$\overline{r_p}$	$\overline{c_p}$
0.90	0.995650	0.896085	0.975070	0.877563	0.902733	0.812460
0.91	0.993008	0.903638	0.971556	0.884116	0.898281	0.817436
0.92	0.989452	0.910296	0.967781	0.890358	0.893546	0.822062
0.93	0.985015	0.916064	0.963723	0.896263	0.888041	0.825878
0.94	0.979730	0.920946	0.959429	0.901863	0.880848	0.827997
0.95	0.973744	0.925057	0.954876	0.907132	0.872427	0.828805
0.96	0.967224	0.928535	0.947650	0.909744	0.862883	0.828367
0.97	0.956892	0.928185	0.935928	0.907850	0.850154	0.824650

Table 3.6.5.: Average maximum fraction of primary keys among 100 random graphs for different page sizes $\breve{m}$ and $d_p = 3$, $d_b = 1$, $m = 10^6$. The failure rate is 0. For $c \geqslant 0.98$ the pseudorandom graphs did not admit a left-perfect matching anymore. The entries of the gray cells are larger than $\hat{c}(3)$.

needed in a search for a key $x \in S$, where naturally we first check the primary page. If $(d_p, d_b) = (3, 1)$ and a key is equally likely to be in any of its four locations, the *expected number of page requests* $\mathrm{Exp}(X)$ would satisfy $\mathrm{Exp}(X) = 1.25$. Actually, for $(d_p, d_b) = (3, 1)$ and c near $\hat{c}(4)$ we have roughly

$$\mathrm{Exp}(X) \approx \hat{c}(3)/c \cdot 1 + (1 - \hat{c}(3)/c) \cdot 2\,.$$

For example, for $(c, \breve{m}) = (0.95, 10^3)$, using the values of Table 3.6.4 we find

$$\mathrm{Exp}(X) \approx 0.974 \cdot 1 + 0.026 \cdot 2 < 1.03\,.$$

Now assume we perform a lookup for a key $x \in U - S$, i. e., a key not in the table. The disadvantage of using two pages per key is that now we always require two page requests, i. e., $\mathrm{Exp}(X) = 2$. This can be circumvented by storing an additional set membership data structure, such as a Bloom filter [Blo70], for each page i representing the $n_{b|i}$ many keys that have primary page i but are inserted on their backup page.

One can trade off space, computation time, and the false positive probability of the Bloom filter as desired. As an example, suppose the Bloom filters use 3 hash functions and their size corresponds to just one bit per page cell. In this case, we can in fact use

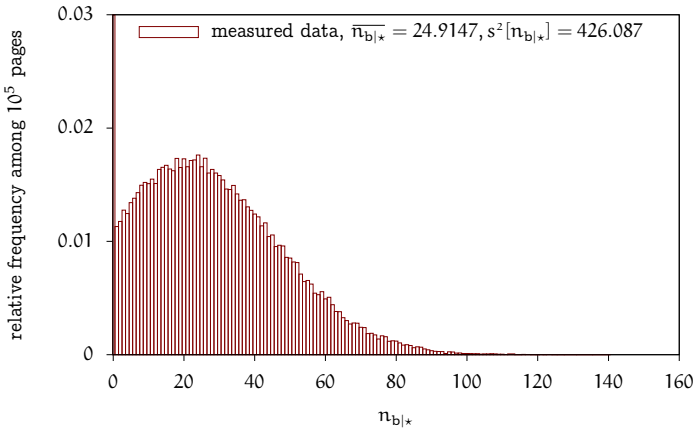


Figure 3.6.11.: Relative frequencies of $n_{b|\star}$ values, i. e., the number of backup keys that are associated with the same primary page, using $(c, m, \check{m}) = (0.95, 10^6, 10^3)$. The relative frequency of $n_{b|\star} = 0$ is 0.169, the failure rate is 0.

the same hash functions that map keys to table cell locations for our Bloom filters. Bounding the fraction of 1 bits of a Bloom Filter from above via $(d_p \cdot n_{b|\star})/\check{m}$, the distribution of $n_{b|\star}$ as in Figure 3.6.11 leads to an average false positive rate of less than 0.15 percent and therefore an expected number of page requests $\mathrm{Exp}(X)$ of less than 1.0015 for unsuccessful searches. One could reduce false positives even further using more hash functions, or use less space.

3.6.4.2. Dynamic Case

We have seen the effectiveness of optimal offline cuckoo hashing with paging. We now investigate whether similar placements can be found online, by considering the simple random walk algorithm from Section 3.6.3.2. We begin with the case of insertions only.

Setup and Measurements Along with the failure rate ξ, the fraction of primary keys r_p and corresponding load factor c_p, and the distribution of the number of keys $n_{b|\star}$ inserted on their backup page, we consider two more performance characteristics:

$\mathfrak{s}$ — the average number of steps of the random walk insertion procedure. A step is either storing a key x in a free cell y or replacing an already stored key with the current "nestless" key.

$\mathfrak{r}$ — the average number of page requests over all inserted items. Here each new key x requires at least one page request, and every time we move an item to its backup page requires another page request.

I. We focus on characteristics of the algorithm with load factors near the hypothetical thresholds $\hat{c}^1_{\check{m}}(4)$, for whose existence we found experimental evidence in the static case, see, e. g., Figures 3.6.9 and 3.6.10, considering page sizes $\check{m} = 10^1, 10^2, 10^3$ and a varying number of table cells $m = 10^5, 10^6, 10^7$. The performance of the algorithm heavily depends on the choice of parameters β and θ. Instead of covering the complete parameter space we first set θ to infinity and use the measurements to give insight into the performance of the algorithm for selected values of β.

II. In addition we want to explore whether we can expect a sufficiently low failure probability of the random walk algorithm, at least for some selected configurations $\kappa = (c, \check{m}, \beta, \theta)$ that include practical values of θ. For this we tested the following null hypothesis

$$\mathcal{H}_0(\kappa, p) \coloneqq \{\text{If one uses parameter set } \kappa\text{, Algorithm 6 fails with probability } \geqslant p.\}.$$

To test the null hypothesis for a specific κ we performed the random experiment

"insertion of $n = c \cdot m$ keys with Algorithm 6"

a times. Consider the event

$$\mathcal{A}(\kappa, a) \coloneqq \{\text{All of the } a \text{ many random experiments for given } \kappa \text{ ended successfully.}\}.$$

Then we have:

$$\Pr\left(\mathcal{A}(\kappa, a) \mid \mathcal{H}_0(\kappa, p)\right) \leqslant (1-p)^a \leqslant \exp(-p \cdot a)\,.$$

For example if $a = 10^6$ and $p = 10^{-5}$ we have

$$\Pr\left(\mathcal{A}(\kappa, a) \mid \mathcal{H}_0(\kappa, p)\right) \leqslant \exp(-10) \approx 4.54 \cdot 10^{-5}\,.$$

Hence, if we observe $\mathcal{A}(\kappa, a)$ we may reject the null hypothesis with high confidence.

III. We also study the influence of β for a fixed configuration. We vary β to see qualitatively how the number of primary keys as well as the number of steps and page requests depend on this parameter.

IV. It is well known that hashing schemes can perform differently in settings with insertions and deletions rather than insertions alone, so we investigate whether there are substantial differences in this setting. Specifically, we consider the table under a constant load by alternating insertion and deletion steps.

V. Finally, we are interested in the relative frequencies of the values $n_{b|\star}$ in order to minimize the number of page accesses for lookup.

$\check{m}$	m	a	p	$\overline{r_p}$	$\overline{c_p}$	$\bar{s}$	$s^2[s]$	$\bar{r}$	$s^2[r]$
10^1	10^5	10^4	10^4	0.860248	0.817236	158.707669	114.072760	10.327258	0.409334
"	10^6	10^3	10^5	0.860219	0.817208	158.618752	11.405092	10.321981	0.040869
"	10^7	10^2	10^6	0.860217	0.817206	158.645056	1.092781	10.323417	0.003914
10^2	10^5	10^4	10^3	0.938431	0.891509	22.807328	1.081478	2.248953	0.003760
"	10^6	10^3	10^4	0.938424	0.891503	22.813986	0.104012	2.249273	0.000366
"	10^7	10^2	10^5	0.938412	0.891491	22.813905	0.010862	2.249201	0.000038
10^3	10^5	10^4	10^2	0.955773	0.907985	16.580150	0.512018	1.892190	0.001779
"	10^6	10^3	10^3	0.955737	0.907950	16.603145	0.052386	1.893515	0.000182
"	10^7	10^2	10^4	0.955730	0.907943	16.598381	0.005534	1.893248	0.000019

Table 3.6.6.: Characteristics of the random walk algorithm concerning the ratio of primary keys as well as the number of steps and page requests, for parameters $(c, \beta, \theta) = (0.95, 0.97, \infty)$ with failure rate $\xi = 0$.

RESULTS Here we consider results from the random walk algorithm, see Algorithm 6.

I. Tables 3.6.6 and 3.6.7 show the behavior of the *random walk algorithm with load factors near* $\hat{c}^1_{\check{m}}(4)$ for $(c, \beta) = (0.95, 0.97)$ and $(c, \beta) = (0.97, 0.90)$. The number of allowed steps for the insertion of n keys is set to infinity via $\theta = \infty$. The number of trials a per configuration is chosen such that $a \cdot m = 10^9$ (keeping the running time for each configuration approximately constant). We first note that with these parameters the algorithm found a placement for the keys in all experiments; failure did not occur. For fixed page size the sample means are almost constant; for growing page size the average load factor with respect to primary keys $\overline{c_p}$ increases, while $\bar{s}$ and $\bar{r}$ decrease, with a significant drop from page size 10 to 100. For a fixed page size the sample variances of all measurements (including c_p) decrease for growing m with about the factor 10; for growing page sizes the sample variances decrease. The sample variances of c_p are not shown in the table; they were at most $3 \cdot 10^{-6}$. For our choices of β the random walk insertion procedure missed the maximum fraction of primary keys by up to 2 percent for $c = 0.95$ and by up to 6 percent for $c = 0.97$ and needs roughly the same average number of steps (for fixed page size).

II. To get more practical values for θ we scaled up the values $\bar{s}$ from Tables 3.6.6 and 3.6.7 and *estimated the failure probability* for each $\kappa = (c, \check{m}, \beta, \theta)$ from $\{(0.95, 10^2, 0.97, 30), (0.95, 10^3, 0.97, 25), (0.97, 10^2, 0.90, 30), (0.97, 10^3, 0.90, 25)\}$ using $m = 10^6$ cells. For all these configurations we observed a failure rate of zero among $a = 10^6$ attempts, i. e., event $\mathcal{A}(\kappa, a)$ took place. Hence, we can conclude at a level of significance of at least $1 - e^{-10}$ that for these configurations the failure probability of the random walk algorithm is at most 10^{-5}.

$\check{m}$	m	a	p	$\overline{r_p}$	$\overline{c_p}$	$\bar{s}$	$s^2[s]$	$\bar{t}$	$s^2[t]$
10^1	10^5	10^4	10^4	0.816795	0.792291	158.506335	1222.640379	32.336112	48.892079
"	10^6	10^3	10^5	0.816790	0.792286	153.645339	78.581917	31.363876	3.142566
"	10^7	10^2	10^6	0.816802	0.792298	152.873602	10.759338	31.209210	0.430827
10^2	10^5	10^4	10^3	0.886997	0.860387	23.320507	2.731285	5.361922	0.108700
"	10^6	10^3	10^4	0.886992	0.860382	23.289233	0.256942	5.355625	0.010218
"	10^7	10^2	10^5	0.886985	0.860375	23.268641	0.024796	5.351518	0.000986
10^3	10^5	10^4	10^2	0.898281	0.871332	19.497032	1.550490	4.607751	0.061739
"	10^6	10^3	10^3	0.898232	0.871285	19.486312	0.146267	4.605481	0.005816
"	10^7	10^2	10^4	0.898235	0.871288	19.493215	0.012744	4.606893	0.000507

Table 3.6.7.: Characteristics of the random walk algorithm concerning the ratio of primary keys as well as the number of steps and page requests, for parameters $(c, \beta, \theta) = (0.97, 0.90, \infty)$ with failure rate $\xi = 0$.

III. Figure 3.6.12 shows how *parameter* β *influences* the ratio of primary keys r_p, the number of insertion steps s and the number of page requests t for an insertion. The mean ratio of primary keys $\overline{r_p}$ grows linearly and $\bar{s}$ grows nonlinearly with growing β. For $\beta = 0.98$ the gap between the optimal fraction of primary keys and the fraction reached by the random walk algorithm is about 1 percent. The value of $\bar{t}$ also depends nonlinearly on β and reaches a local minimum at $\beta = 0.95$. The sample variances are quite small and stable except for $s^2[s]$ and large β (near 0.98).

IV. Figure 3.6.13 shows the results for *alternating insertions and deletions* using configuration $(c, m, \check{m}, \beta, \theta) = (0.95, 10^6, 10^3, 0.97, 30)$. We measured the current ratio of primary keys r_p and the number of insertion steps with respect to each key denoted by s_{key}; recall that s is the average number of insertion steps concerning all keys. In the first phase (insertions only) the average number of steps per key grows very slowly at the beginning and is below 10 when reaching a load where about 1 percent of current keys are backup keys. After that $\overline{s_{key}}$ grows very fast up to almost 10^3 (for the last few keys), which is the page size. The sample mean of the average number of steps $\bar{s}$ up to this point is about 16.6. Similarly the sample mean of the ratio of primary keys $\overline{r_p}$ decreases very slowly at the beginning and decreases faster at the end of the first phase. Up to load about 82.6 percent, the fraction of backup keys is below 1 percent. In the second phase (deletions and insertions alternate) $\overline{s_{key}}$ and $\bar{s}$ decrease and quickly reach a steady state. Since the decrease of $\overline{r_p}$ is marginal but the drop $\overline{s_{key}}$ is significant we may conclude that the overall behavior is better in steady state than at the end of the insertion only phase. Moreover, in an extended experiment with $n = c \cdot m$ insertions and $10 \cdot n$ delete-insert pairs the observed equilibrium remains the same and therefore underpins the conjecture that Figure 3.6.13 really shows a "convergence point" for alternating deletions and insertions.

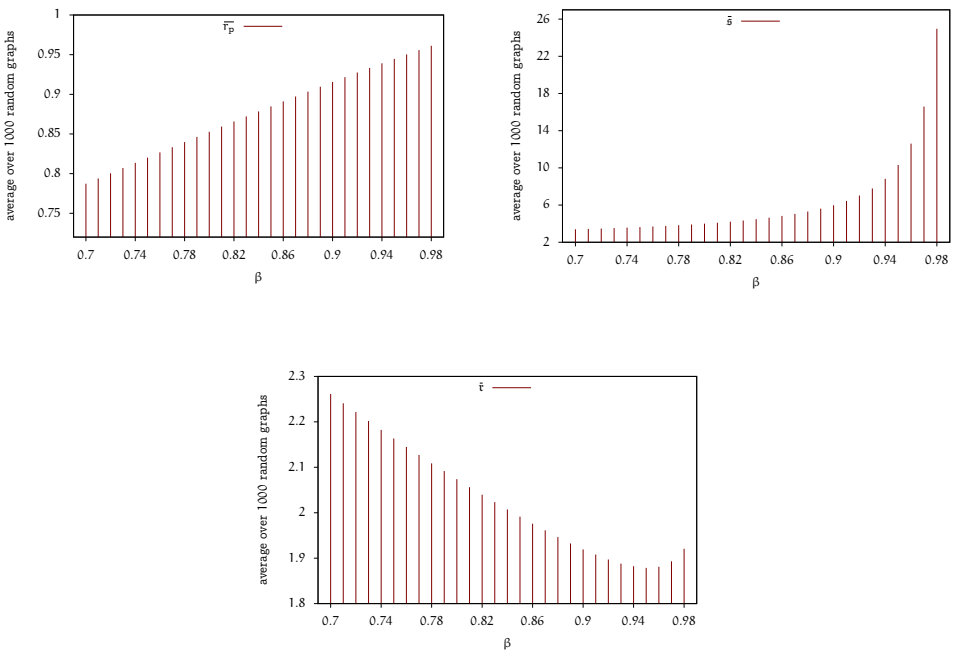


Figure 3.6.12.: Influence of parameter β on the ratio of primary keys as well as the number of steps and page requests for an insertion exemplarily for configuration $(c, m, \check{m}, \theta) = (0.95, 10^6, 10^3, 30)$. Average values are over $a = 10^3$ attempts with failure rate $\xi = 0$.

V. Figure 3.6.14 shows the *relative frequency of the values* $n_{b|\star}$ among 10^5 pages for parameters $(c, m, \check{m}, \beta, \theta) = (0.95, 10^6, 10^3, 0.97, 30)$ at the end of the insertion only phase, given by Figure 3.6.14 (a), and at the end of the alternation phase, given by Figure 3.6.14 (b). Note that Figure 3.6.14 (a) corresponds to Figure 3.6.11 with respect to the graph parameters. The shapes of the distributions differ only slightly, except that in the second phase the number of backup keys is larger. In comparison with the values given by the optimal algorithm in Figure 3.6.11 the distribution of the $n_{b|\star}$ values is more skewed and shifted to the right.

Number of Page Accesses Our simple online random-walk algorithm, with appropriately chosen parameters, performs quite close to the optimal offline algorithm for cuckoo hashing with pages, even in settings where deletions occur. With parameters $(c, \check{m}, \beta, \theta) = (0.95, 10^3, 0.97, 30)$ the *expected number of page requests* for a successful search is about 1.044, using the values from Table 3.6.6. With the Bloom filter approach described in Section 3.6.4.1, which can be done only after finishing the

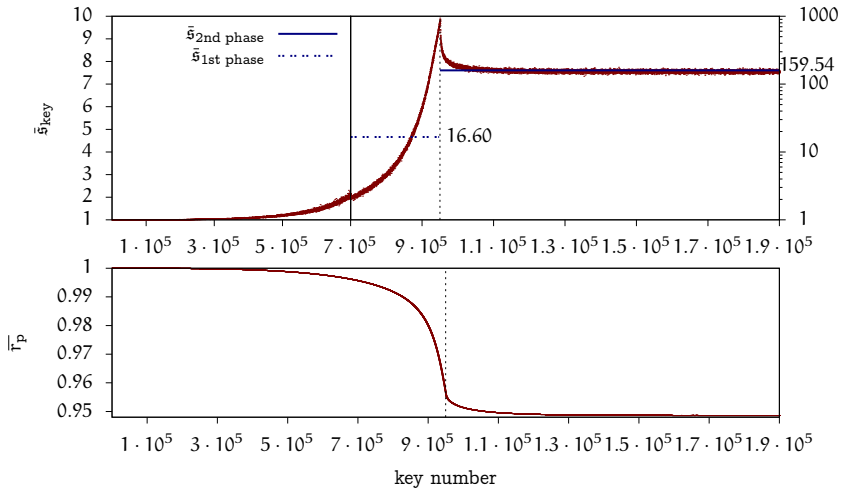
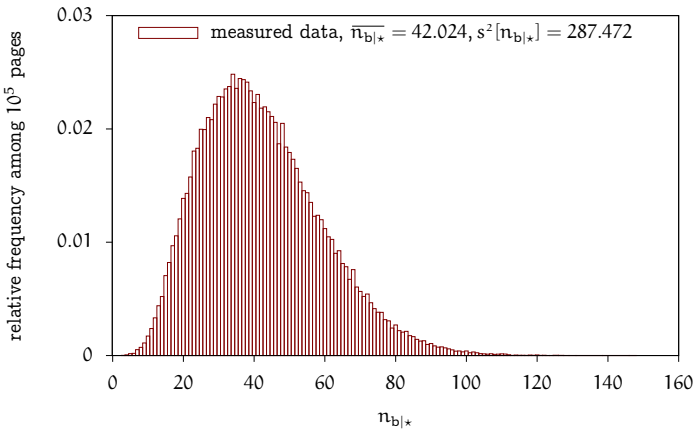


Figure 3.6.13.: Characteristics of the random walk algorithm concerning the ratio of primary keys and number of insertion steps per key in a scenario with alternating insertions and deletions using configuration $(c, m, \check{m}, \beta, \theta) = (0.95, 10^6, 10^3, 0.97, 30)$. Average values were determined over $a = 10^3$ attempts with failure rate $\xi = 0$. The upper plot is divided into two parts at $7 \cdot 10^5$. The ordinate of the right part is in logarithmic scale, whereas the ordinate of the left part is in linear scale.

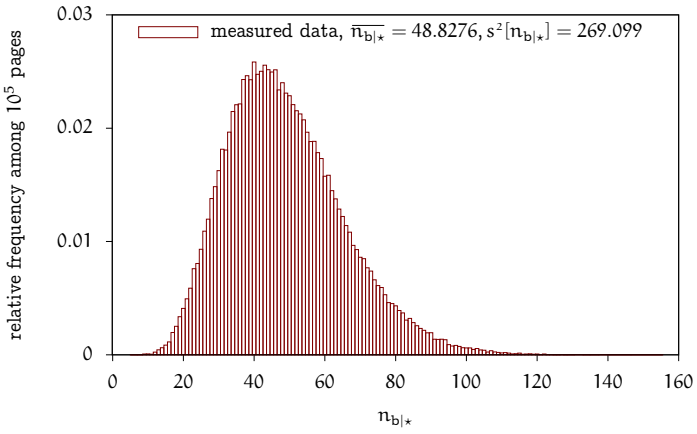
insertion of all keys, the distribution from Figure 3.6.14 (a) gives an expected number of page requests for an unsuccessful search of less than 1.0043. Both values are only slightly higher than those resulting from an optimal solution. In order to improve performance for unsuccessful searches with online insertions and deletions, one can use counting Bloom filters [FCAB98, FCAB00] at the cost of more space.

3.6.4.3. Alternative Approach for Small Pages

We have seen that if one uses one backup option, then the page size has only marginal influence on the existence of a left-perfect matching or $(1,1)$-orientation of $G^{d_p,d_b}_{n,(m,\check{m})}$, but heavily influences the maximum fraction of primary keys, in the dynamic case as well as in the static case. Tables 3.6.5 to 3.6.7 show that the smaller the page the smaller the fraction of primary keys, with a significant decrease from page size 100 to 10. In order to attenuate this downside one can use the following variant. We use the idea of (non-overlapping) *blocked cuckoo hashing* as described in Section 3.1.3, where the table consists of m blocks, each of capacity ℓ entries. Our graph model remains $G^{d_p,d_b}_{n,(m,\check{m})}$ but now each of the m right nodes has capacity ℓ, the page size is set to $\check{m} = 1$, i. e., we think of each table cell as separate page and we let $d_p = d_b = 1$ and $d = 2$. Analogously to the experiments before, we want to find a $(1,\ell)$-orientation



(a) insertion only phase



(b) insertion and deletion phase

Figure 3.6.14.: Relative frequency of $n_{b|\star}$ values among 10^5 pages, resulting from $a = 10^2$ graphs with $p = 10^3$ pages each, for $(c, m, \check{m}, \beta, \theta) = (0.95, 10^6, 10^3, 0.97, 30)$ with a failure rate of $\xi = 0$.

ℓ	2	3	4	5	8	10	16
$\hat{c}_{1,\ell}(2)/\ell$	0.89701	0.95915	0.98037	0.98955	0.99785	0.99914	0.99993

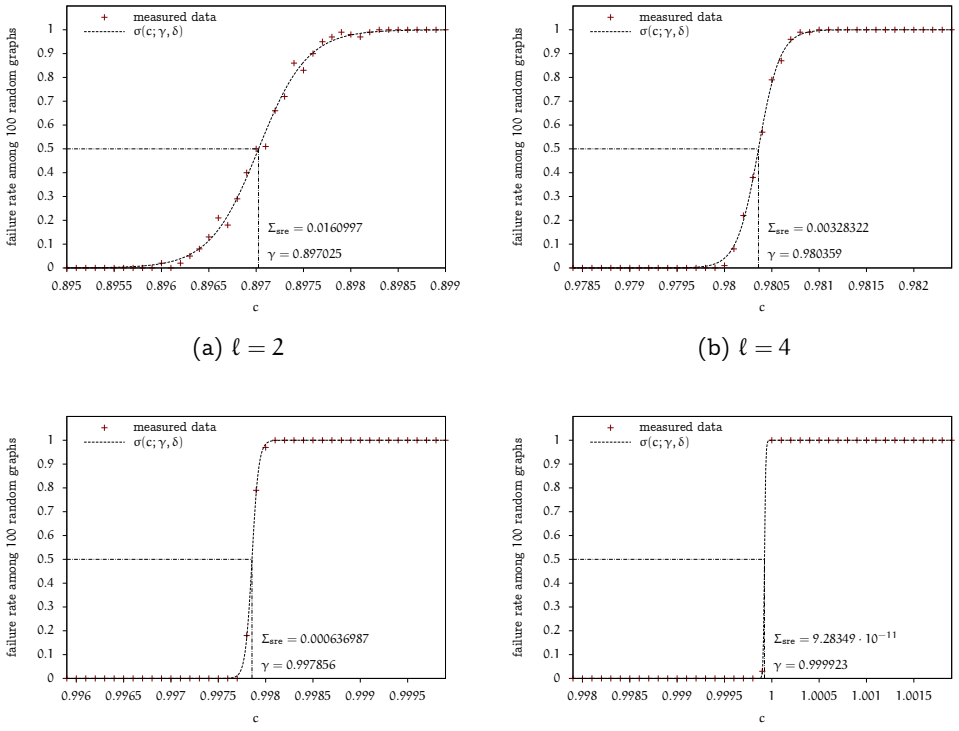
Table 3.6.8.: Normalized load thresholds $\hat{c}_{1,\ell}(2)/\ell$ for the existence of a $(1,\ell)$-orientation of $G^{d_p,d_b}_{n,(m,\check{m})}$ with $\check{m}=1$ for $d_p=d_b=1$. The values are rounded to the nearest multiple of 10^{-5}.

of $G^{d_p,d_b}_{n,(m,\check{m})}$, i. e., an orientation of the edges such that each left node has indegree exactly 1 and each right node has outdegree at most ℓ. A $(1,\ell)$-orientation that has a *maximum number of primary keys* is called *optimal*.

! For the rest of the section we will only consider the static case, since like before, in the dynamic case we come close to the solutions obtained in this ideal setting.

As already discussed in Section 3.1.1.5, the thresholds $\hat{c}_{1,\ell}(2)$ for the existence of a $(1,\ell)$-orientation of $G^{d_p,d_b}_{n,(m,\check{m})}$ and $G^{d}_{n,m}$, respectively, are known. Some reference *normalized load thresholds* are given in Table 3.6.8. For comparison, experimental threshold values are given in Figure 3.6.15. They were obtained using Algorithm 5 for determining a left-perfect matching or $(1,1)$-orientation, respectively, on modified pseudorandom graphs $G^{d_p,d_b}_{n,(m,\check{m})}$, where each right node y is replaced with ℓ copies and each edge to which y is incident is replace with ℓ copies as well. An advantage of the alternative approach for small pages is that for $\ell > 3$ the normalized load thresholds $\hat{c}_{1,\ell}(2)/\ell$ for a $(1,\ell)$-orientation are higher than the hypothetical thresholds $\hat{c}^{1}_{\check{m}}(4)$ for a $(1,1)$-orientation, which are near $\hat{c}(4) \approx 0.97677$. Our aim remains to *maximize the ratio of primary keys* while keeping the normalized load factor c/ℓ near the threshold $\hat{c}_{1,\ell}(2)/\ell$. Some experimental results are given in Table 3.6.9. The comparison with Table 3.6.5 shows that with respect to this ratio obtaining a $(1,\ell)$-orientation of $G^{d_p,d_b}_{n,(m,\check{m})}$ using parameters $(m, d_p, d_b, \check{m}, \ell) = (r, 1, 1, 1, 10)$ is slightly better than obtaining a $(1,1)$-orientation of $G^{d_p,d_b}_{n,(m,\check{m})}$ using parameters $(m, d_p, d_b, \check{m}, \ell) = (r \cdot 10, 3, 1, 10, 1)$, with the additional benefit of requiring only two hash functions instead of four.

Number of Page Accesses The higher ratio of primary keys r_p, achievable with the alternative approach for small pages, allows to further reduce the number of page requests, see Section 3.6.4.1. For example, with $(c, \check{m}, \ell) = (0.97, 10, 1)$ a fraction of $r_p = 0.850$ of the keys can be stored in their primary page, while for parameters $(c/\ell, \check{m}, \ell) = (0.97, 1, 10)$ we have $r_p = 0.884$, and for $(c/\ell, \check{m}, \ell) = (0.97, 1, 16)$ one gets even a ratio of $r_p = 0.913$. This also holds for load factors beyond $\hat{c}^{1}_{\check{m}}(4)$. For example, using parameters $(c/\ell, \check{m}, \ell) = (0.99, 1, 16)$, the ratio of primary keys is 0.904 which reduces the *expected number of page requests* for successful searches from 1.5, when each key is equally likely to be in either blocks, to about $0.904 \cdot 1 + 0.096 \cdot 2 < 1.1$.



(a) $\ell = 2$ (b) $\ell = 4$

(c) $\ell = 8$ (d) $\ell = 16$

Figure 3.6.15.: Approximation of normalized load thresholds $\hat{c}_{1,\ell}(2)/\ell$ for different capacities ℓ using the average failure rate over 100 pseudorandom graphs $G^{d_p,d_b}_{n,(m,\check{m})}$ with configuration $(c, m, d_p, d_b, \check{m}, \ell) = (c, 10^6, 1, 1, 1, \ell)$.

	$\ell = 4$		$\ell = 8$		$\ell = 10$		$\ell = 16$	
c/ℓ	$\overline{r_p}$	$\overline{c_p}/\ell$	$\overline{r_p}$	$\overline{c_p}/\ell$	$\overline{r_p}$	$\overline{c_p}/\ell$	$\overline{r_p}$	$\overline{c_p}/\ell$
0.90	0.822251	0.740026	0.898282	0.808454	0.913798	0.822418	0.940022	0.846020
0.91	0.815652	0.742244	0.894259	0.813776	0.910014	0.828113	0.936509	0.852223
0.92	0.808867	0.744157	0.890040	0.818837	0.906196	0.833700	0.932937	0.858302
0.93	0.801424	0.745325	0.885679	0.823682	0.902177	0.839024	0.929188	0.864145
0.94	0.793452	0.745845	0.881098	0.828232	0.898052	0.844169	0.925333	0.869813
0.95	0.784526	0.745300	0.876222	0.832411	0.893687	0.849003	0.921360	0.875292
0.96	0.774254	0.743283	0.870778	0.835947	0.889150	0.853584	0.917347	0.880653
0.97	0.761745	0.738893	0.864615	0.838676	0.884317	0.857787	0.913244	0.885846
0.98	0.743799	0.728923	0.857017	0.839876	0.878957	0.861378	0.908929	0.890751
0.99	no solution		0.847632	0.839156	0.870738	0.862030	0.904474	0.895429

Table 3.6.9.: Average maximum fraction of primary keys among 100 random graphs $G^{d_p,d_b}_{n,(m,\check{m})}$ with $\check{m} = 1$ and $m = 10^6$ right nodes, for different block sizes ℓ. For $\ell = 4$ and $c/\ell = 0.98$ the failure rate is $\xi = 0.01$, for $\ell = 4$ and $c/\ell = 0.99$ the failure rate is $\xi = 1$; otherwise we have $\xi = 0$.

3.7. Conclusion

We gave experimental evidence that the *generalized selfless algorithm* is likely to find (k, ℓ)-orientations of random hypergraphs $H^d_{m,n}$ as long as the edge density $c = n/m$ is just below the threshold $\hat{c}_{k,\ell}(d)$; a proof for arbitrary constant (k, ℓ) is still open. Furthermore, we showed how to implement this algorithm to run in linear time, which results in a practical construction algorithm for cuckoo hashing data structures.

We found (near) *optimal degree distributions* for matchings in random bipartite multigraphs $G^d_{n,m}$, where each left node chooses its right neighbors randomly with repetition according to its assigned degree. For the case that the number of left nodes is linear in the number of right nodes, we showed that all of these distributions give the matching thresholds $\hat{c}(\bar{d})$ and in the case of near optimal degree distributions we conjectured what the optimal distribution is above and below the threshold.

We studied *cuckoo hashing with pages*, i. e., essentially random bipartite graphs $G^{d_p,d_b}_{n,(m,\check{m})}$, and gave experimental evidence, that if a key is associated with d_p cells on its primary page and has only $d_b = 1$ cell on its backup page, then a left-perfect matching is likely to exist, even if the load factor is near the threshold $\hat{c}(d_p + d_b)$. Furthermore, we showed how to obtain a placement of the keys, such that most lookups require just one page access. Moreover, we proposed simple but efficient algorithms in order to find such placements offline and online.

4

Retrieval and Perfect Hashing

In this chapter we cover solutions of the perfect hashing and retrieval problem based on the Bloomier filter [CKRT04]. The two data structure problems can be briefly restated as follows.

Perfect Hashing — Given $S \subseteq U$ and an integer $\acute{m}$, build a data structure $\mathfrak{D}$ that on $\texttt{lookup}(\mathfrak{D}, x)$ returns some $a_x \in [\acute{m}]$, where $a_x \neq a_{x'}$ for all $\{x, x'\} \in \binom{S}{2}$.

Retrieval — Given $g = \{(x_0, v_0), (x_1, v_1), \ldots, (x_{n-1}, v_{n-1})\} \subset S \times V$, build a data structure $\mathfrak{D}$ that on $\texttt{lookup}(\mathfrak{D}, x)$ returns $g(x)$ if $x \in S$ and returns an arbitrary value otherwise.

Throughout the chapter we assume that $(V, \oplus)$ is an *abelian group* with neutral element 0_V.

4.1. Bloomier Filter Variants

We start with a description of the *immutable Bloomier filter* [CKRT04, Section 3.1], a data structure for solving retrieval without an explicit representation of the key set. Afterwards, we discuss a generalization, called *irregular immutable Bloomier filter* and state the main result of this chapter, which essentially says that an irregular left degree distribution of the underlying graph helps to increase the load factor of the Bloomier filter. Then we describe the *mutable Bloomier filter* [CKRT04, Section 3.3], a data structure for efficiently solving perfect hashing, using a slightly more universal approach than usually discussed in the literature. Finally, we consider *irregular mutable Bloomier filters* and analogously show that they can be made more space efficient than their regular counterparts.

4.1.1. Immutable Bloomier Filter

The immutable Bloomier filter is a hashing based data structure on top of the basic scheme considered in Section 2.2. It is not a "classical hash table", but nevertheless simple and elegant. Its construction principle, presented in [CHM92, MWHC96], was

independently rediscovered by Chazelle, Kilian, Rubinfeld, and Tal, who introduced a new clever modification, which we will discuss in Section 4.1.3.

4.1.1.1. Construction

To realize a retrieval data structure $\mathfrak{D}$ using the basic scheme we consider the matrix representation $\mathbf{M}^d_{n,m} = (\mathbf{b}^T_{x_i})_{i\in[n]} \in \{0,1\}^{n\times m}$. We use this binary matrix in order to define a group homomorphism $\phi\colon V^m \to V^n$ from $(V^m, \oplus)$ to $(V^n, \oplus)$ with componentwise $\oplus$, according to

$$\phi(\boldsymbol{w}) = \mathbf{M}^d_{n,m} \cdot \boldsymbol{w} := \left(\bigoplus_{j\in A_{x_i}} w_j \right)_{i\in[n]} \quad \text{for all } \boldsymbol{w} \in V^m.$$

Let $\boldsymbol{v} = (v_i)_{i\in[n]} \in V^n$ be the vector of the values to be stored. Now the construction boils down to setting the vector $\mathbf{t}$, which is the table of the basic scheme, to an arbitrary element of the preimage $\phi^{-1}(\boldsymbol{v})$, i. e.,

$$\mathbf{t} \in \{\boldsymbol{w} \in V^m \mid \mathbf{M}^d_{n,m} \cdot \boldsymbol{w} = \boldsymbol{v}\}.$$

So, in order that $\mathfrak{D}$ can be built successfully, the preimage of $\boldsymbol{v}$ must not be empty or with other words, there must be at least one solution $\mathbf{t}$ of the system of equations

$$\left\{ \bigoplus_{j\in A_{x_i}} t_j = v_i \right\}_{i\in[n]}. \tag{SoE}$$

To obtain simple *necessary and sufficient* conditions for this to happen, let us assume for a moment that $(V, \oplus)$ is an *elementary* abelian group, i. e., each element from V, without the neutral element, has order p for some fixed prime p. In this case the group is isomorphic to $(\mathbb{Z}_p)^r$ and can be seen as r-dimensional vector space over $\mathbb{F}_p$; a natural example is $(V, \oplus) = (\{0,1\}^r, \mathrm{xor})$. It follows that $(V^m, \oplus)$ and $(V^n, \oplus)$ can be considered as m and n dimensional vector spaces over $\mathbb{F}_{p^r}$. Now we have that ϕ is a linear map and the (linear) system (SoE) has a solution if and only if the rank of the augmented matrix $(\mathbf{M}^d_{n,m} \mid \boldsymbol{v})$ equals $\mathrm{rank}(\mathbf{M}^d_{n,m})$. This is the case if $\mathbf{M}^d_{n,m}$ has full row rank, i. e., there exists a $n \times n$ submatrix with non-zero *determinant*. A *sufficient condition* for full row rank is that the matrix $\mathbf{M}^d_{n,m}$ can be transformed into row echelon form by a finite sequence of *row and column permutations*. And this remains sufficient for the existence of a solution of (SoE) in the general case that $(V, \oplus)$ is *not elementary* abelian.

The matrix $\mathbf{M}^d_{n,m}$ can be transformed into row echelon form by permutations if and only if $G^d_{n,m}$ admits an *order generating matching* according to the following definition.

Definition 3: (Order Generating Matching, cf. [CKRT04, Definition 3.2])
Let $G = (L \cup R, E)$ be a bipartite graph, let M be a matching in G, and let L_M be the set of matched left nodes. We define a relation $>_M \subseteq L_M \times L_M$ according to

$$x >_M x' \Leftrightarrow \exists y \in R\colon \{x, y\} \in M \ \wedge \ y \notin N(\{x'\})\,.$$

The matching M is called *order generating matching* for G if there exists a permutation π such that the left node set $L = \{x_0, x_1, \ldots, x_{|L|-1}\}$ is completely ordered with respect to "$>_M$", i. e., $x_{\pi(0)} >_M x_{\pi(1)} >_M \ldots >_M x_{\pi(|L|-1)}$.

The graph $G^d_{n,m}$ has an order generating matching if and only if it fulfills a strong expansion property known as *singleton property*.

Definition 4: (Singleton Property, cf. [CKRT04, Definition 4.1])
A bipartite graph $G = (L \cup R, E)$ has the *singleton property* if each non-empty subset L' of L has an element x such that the neighborhood of L' is strictly larger than the neighborhood of L' without x,

$$\forall L' \subseteq L, L' \neq \emptyset, \exists x \in L'\colon |N(L' - \{x\})| < |N(L')|\,. \tag{SP}$$

Note that the singleton property implies the marriage condition (MC). An alternative view, which we will stress in the following, is that $G^d_{n,m}$ has the singleton property if and only if it is (completely) *peelable* according to [Sie04, Definition 2.5], which means that Algorithm 1 returns $\emptyset$ on input $H^d_{m,n}$, i. e., the 2-core of $H^d_{m,n}$ is *empty*. Figure 4.1.1 shows an example.

4.1.1.2. Lookup Operation

In order to retrieve the value that is associated with a key $x \in U$ the entries of all cells addressed by A_x are "added together" in arbitrary order, i. e., for all $x \in U$ we define

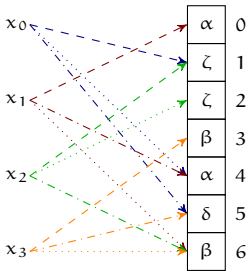
$$\texttt{lookup}(\mathfrak{D}, x) \coloneqq \bigoplus_{a \in A_x} t_a\,.$$

Hence, if $x \in S$, then by (SoE) its associated value v is returned, and if $x \in U - S$, the return value is unknown.

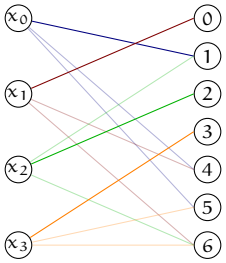
4.1.1.3. Success Probability

The probability of a successful construction is the probability that (SoE) has a solution. In the following we will restrict ourselves to *two events that are sufficient* conditions for solvability.

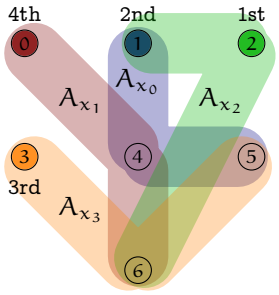
1. $\{G^d_{n,m}$ admits an order generating matching.$\}$
2. $\{\boldsymbol{M}^d_{n,m}$ has full row rank over $\mathbb{F}_2$.$\}$ — assuming implicitly that $(V, \oplus)$ is elementary



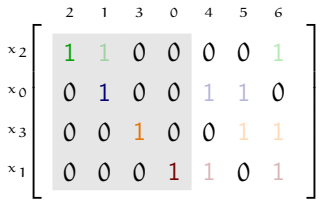
(a) retrieval data structure built upon the basic scheme as given in Figure 2.2.1 (b)



(b) bipartite graph representation, an order generating matching is indicated via dark edges



(c) corresponding hypergraph; removing the dark nodes with their incident edge, according to the given order, results in an empty set, that is the 2-core is empty



(d) binary matrix representation in row echelon form, transformed via row and column permutations, the gray square matrix has non-zero determinant over $\mathbb{F}_2$

Figure 4.1.1.: A retrieval data structure for $g = \{(x_0, \gamma), (x_1, \beta), (x_2, \zeta), (x_3, \zeta)\}$, realized with the basic hashing scheme (type B), as well as its corresponding graph, hypergraph, and matrix representations. The group $(\{\alpha, \beta, \gamma, \delta, \varepsilon, \zeta\}, \oplus)$ is isomorphic to $(\mathbb{Z}_6, +)$; $\alpha \hat{=} 0$, $\beta \hat{=} 1$, $\gamma \hat{=} 2$, $\delta \hat{=} 3$, $\varepsilon \hat{=} 4$, $\zeta \hat{=} 5$.

In contrast to the probability of a left-perfect matching, which was central in Chapter 3, it is not immediately clear how the probabilities of the two events depend qualitatively on d. As before we consider the situation that d is given and ask for the success probability as a function of the load factor $c = n/m$ for $n \to \infty$.

CASE $d = 1$ If $G^d_{n,m}$ is 1-left-regular, then each left-perfect matching is also an order generating matching. Furthermore, $G^d_{n,m}$ admits a left-perfect matching if and only if $\mathbf{M}^d_{n,m}$ has full row rank. As we have discussed in Section 3.3.2, the probability that $G^d_{n,m}$ admits a left-perfect matching is zero w. h. p. if $n(m) = \omega(\sqrt{m})$. Hence, this case is not attractive if we aim for small space usage of the data structure and therefore is omitted for the rest of the chapter. As before, the situation dramatically changes for $d \geqslant 2$.

CASE $d = 2$ If each left node of $G^d_{n,m}$ has degree two, then one observes a phase transition concerning the occurrence of both events, which, however, is not "sharp", as stated in the following theorem.

THEOREM 4.1 (BAD CYCLE(S), SEE [ER60, CHM92] AND APPENDIX A.1)
Let $d = 2$ and let $c = n/m$ be constant. Then the following holds:

(i) If $c < 0.5$, then $G^d_{n,m}$ admits an order generating matching as well as $\mathbf{M}^d_{n,m}$ has full row rank with probability $P(c, n)$, where

$$\lim_{n\to\infty} P(c,n) = \begin{cases} \exp(c) \cdot \sqrt{1-2\cdot c}, & \text{for type A} \\ \exp(c) \cdot \sqrt{1-2\cdot c}, & \text{for type B} \\ \sqrt{1-4\cdot c^2}, & \text{for type C} \end{cases}.$$

(ii) If $c > 0.5$, then $G^d_{n,m}$ admits no order generating matching as well as $\mathbf{M}^d_{n,m}$ has not full row rank w. h. p..

The theorem is based on the observation that if the (normal) graph $H^d_{m,n}$ has a cycle of length at least two (type A, B and C), or a simple path that starts and ends at two nodes that have a cycle of length one (only type A), then:

1. The singleton property is violated.
2. The corresponding rows of $\mathbf{M}^d_{n,m}$ add up to the zero vector.

For type B and type C the theorem follows directly from [CHM92, Section 3], which extends results from [ER60, Theorem 5b]; see also [CHM97, Theorem 6.3]. The transfer from type B to type A is discussed in Appendix A.1.

REMARK. The threshold 0.5 is the same as the threshold for the appearance of the *giant component* in $H^d_{m,n}$.

d	3	4	5	6	7	8	9
$\check{c}(d)$	0.81847	0.77228	0.70178	0.63708	0.58178	0.53500	0.49526

Table 4.1.1.: Thresholds $\check{c}(d)$ for the existence of order generating matchings in $G^d_{n,m}$, rounded to the nearest multiple of 10^{-5}.

Case $d \geqslant 3$ If $G^d_{n,m}$ has a left degree of at least 3, then there is a sharp phase transition but now the probabilities of the sufficient conditions diverge. As we have mentioned before, $G^d_{n,m}$ has an order generating matching if and only if the 2-core of $H^d_{m,n}$ is empty, i. e., the points of transition are the thresholds $\check{c}(d)$, as defined in Section 2.3.1 and given in Table 4.1.1.

Theorem 4.2 (Order Generating Matching)
Let $d \geqslant 3$ be a constant integer and let $c = n/m$. Then for any constant $\varepsilon > 0$ a. a. s. the following holds:

(i) If $c \leqslant \check{c}(d) - \varepsilon$, then $G^d_{n,m}$ admits an order generating matching.

(ii) If $c \geqslant \check{c}(d) + \varepsilon$, then $G^d_{n,m}$ admits no order generating matching.

Since $\arg\min_{\lambda \in (0,\infty)} \mathrm{key}(\lambda, d) \approx \ln(d) + \ln(\ln(d))$, see [MWHC96, Section 4.3], the threshold for the appearance of the 2-core can be approximated via

$$\check{c}(d) \approx \frac{\ln(d) + \ln(\ln(d))}{d \cdot \left(1 - \frac{1}{d \cdot \ln(d)}\right)^{d-1}}.$$

The function $\check{c}(d)$ is monotonically decreasing. Hence the maximum threshold is $\check{c}(3)$, which is about 0.818.

It was shown by Dietzfelbinger et al. [DGM+09, DGM+10] that the *satisfiability threshold* for random d-XORSAT, obtained by Dubois and Mandler for $d = 3$ [DM02] and by Pittel and Sorkin for $d \geqslant 3$ [PS12], is the same as the threshold that $G^d_{n,m}$ has a left-perfect matching. Since a random instance of the d-XORSAT problem can be interpreted as linear system $\boldsymbol{M}^d_{n,m} \cdot \boldsymbol{w} = \boldsymbol{v}$ over $\mathbb{F}_2$, with random right-hand side $\boldsymbol{v} \in \{0,1\}^n$, this threshold is in essence the threshold that the random matrix $\boldsymbol{M}^d_{n,m}$ has full row rank.

Theorem 4.3 (Full Row Rank, [DGM+10, PS12])
Let $d \geqslant 3$ be a constant integer and let $c = n/m$. Then for any constant $\varepsilon > 0$ a. a. s. the following holds:

(i) If $c \leqslant \hat{c}(d) - \varepsilon$, then $\boldsymbol{M}^d_{n,m}$ has full row rank.

(ii) If $c \geqslant \hat{c}(d) + \varepsilon$, then $\boldsymbol{M}^d_{n,m}$ has not full row rank.

In contrast to the threshold $\check{c}(d)$ for an order generating matching, the thresholds for full row rank $\hat{c}(d)$ are monotonically increasing with increasing d, see Table 3.1.1.

4.1.1.4. Space and Time Consumption

Let $m = (1+\varepsilon) \cdot n$ and recall that we assume ideal hash functions as stated in Section 2.2.2. If we rely on the condition that $\mathbf{M}^d_{n,m}$ has full row rank, then the immutable Bloomier filter can be constructed a. a. s. with

space usage $(1+\varepsilon) \cdot |S| \cdot \lceil \log|V| \rceil$, and

lookup time $O\big(\log(1/\varepsilon)\big)$.

for arbitrary constant $\varepsilon \in (0,1)$. This is a factor of $1+\varepsilon$ away from the information theoretical minimum, see Section 4.3.1. However, if we rely on the condition that $G^d_{n,m}$ has an order generating matching, then ε is bounded from below by $1/\check{c}(3) - 1 \approx 0.22$, since $\check{c}(d)$ is monotonically decreasing.

4.1.1.5. Linear Time Algorithms

We can build a Bloomier filter retrieval data structure if and only if the system of equations (SoE) has a solution. If $\mathbf{M}^d_{n,m}$ has an order generating matching, then a solution can be found easily via Algorithm 7, cf., [CKRT04, page 35 *CREATE*], a combination of *peeling* and *back substitution*, as follows.

Consider the corresponding hypergraph $H^d_{m,n}$. Successively remove nodes x_i of degree 1 together with their incident edge A_{x_i} and put them on a stack. Each node-edge pair encodes a column and a row permutation of $\mathbf{M}^d_{n,m}$. If the procedure ends with an empty hypergraph or 2-core respectively, then the matrix can be transformed in row echelon form using only these permutations. In this case we apply back substitution via successively removing node-edge pairs (x_i, A_{x_i}) from the top of the stack and solving the equations $\bigoplus_{j \in A_{x_i}} t_j = v_i$; otherwise a failure is returned.

In the case that $\mathbf{M}^d_{n,m}$ has no order generating matching but has full row rank, we could try to solve (SoE) directly, assuming that $(V, \oplus)$ is elementary abelian and using an algorithm adapted for sparse linear systems over finite fields. However, these methods, like the *conjugate gradient method* and the *Lanczos algorithm*, see, e. g. [HE05, Introduction], as well as Wiedemann's algorithm [Wie86], have running time $\Omega(n^2)$ for constant $c = n/m$ and constant d. In order to achieve expected linear construction time, Dietzfelbinger and Pagh proposed an alternative that relies on randomly splitting the key set (matrix $\mathbf{M}^d_{n,m}$) into subsets (submatrices) with $n' = 1/2 \cdot \sqrt{\log n}$ keys (rows) in expectation [DP08a, DP08b]. For each submatrix that has not too many rows the solution of the corresponding linear system is obtained via precomputed pseudoinverses in time $O(n')$. The rest of the keys is handled using a backup data structure. Since the number of $n' \times m'$ binary matrices with d ones per row is $(m'^{\underline{d}}/d!)^{n'}$, it seems clear that this approach is worthwhile only for fairly small matrices, although only a fraction of them has full row rank. In fact, we confirm in an experimental study [ADR09] that the approach using precomputed pseudoinverses is of rather theoretical interest and cannot be used for realistic values of n to obtain very

ALGORITHM 7: `peeling_with_back_substitution`

Input : Matrix $\mathbf{M} \in \{0,1\}^{n \times m}$ and vector $\boldsymbol{v} \in V^n$.
Output : Vector $\mathbf{t} \in V^m$ with $\mathbf{M} \cdot \mathbf{t} = \boldsymbol{v}$, if 2-core of $\mathbf{M}$ is empty; otherwise $\emptyset$.
Require : Stack $\mathfrak{S}$ and abelian group $(V, \oplus)$.
$H \leftarrow$ hypergraph with node set $[m]$ and incidence matrix $\mathbf{M}$;
$\mathfrak{S} \leftarrow$`empty()`;
while H *has a node* i *incident to only one edge* e_j do
  $\mathfrak{S} \leftarrow \texttt{push}(\mathfrak{S}, (i, e_j))$;
  remove node i and edge e_j from H;
end
if $H \neq \emptyset$ then return $\emptyset$;
while not `empty(`$\mathfrak{S}$`)` do
  $(i, e_j) \leftarrow \texttt{top}(\mathfrak{S})$; $\mathfrak{S} \leftarrow \texttt{pop}(\mathfrak{S})$;
  $t_i \leftarrow v_j \oplus \Big(\bigoplus_{k \in e_j \,:\, k \neq i} t_k \Big)^{-1}$;
end
return $(t_i)_{i \in [m]}$

space-efficient Bloomier filters, e. g., with $m = 1.1 \cdot n$. However, using the same splitting approach and solving linear systems directly, i. e., without using precomputed tables, a reasonable tradeoff between space consumption (during and after the construction) and the construction time can be obtained; note that the construction time is expected linear for constant n'. For example, with $n = 10^6$, $n' = 500$, and $m = 1.1 \cdot n$ a Bloomier filter can be obtained within a few minutes, see [ADR09, Figure 1].

4.1.2. Irregular Immutable Bloomier Filter

Among the methods for solving (SoE) that we have discussed, the fastest and simplest approach is to apply Algorithm 7, i. e., peeling with back substitution. Unfortunately, with this algorithm the space overhead of the solution vector $\mathbf{t}$ cannot be made arbitrarily small, since $c = n/m$ is limited by the threshold of the appearance of the 2-core in $H^d_{m,n}$.

However, it is known for related random bipartite graphs that emerge in the context of erasure correcting codes that *irregular* node degrees that follow carefully chosen distributions increase the threshold for the appearance of a 2-core in comparison to the situation where the right node degree is *regular*, see, e. g., [LMSS01][1]. We show that this also holds for our random graphs, which can be used to derive space efficient irregular immutable Bloomier Filters that can be constructed via Algorithm 7 — more details will be provided soon.

[1] Here two degree distributions are selected, one for the right nodes and one for the left nodes.

We modify the basic scheme such that for each $x \in U$ a randomized algorithm determines some number of hash functions d_x, and change the sequence of addresses to

$$h(x) = \big(h_0(x), h_1(x), \ldots, h_{d_x-1}(x)\big)\,.$$

More precisely, the number of hash functions becomes a random variable D_x that follows a probability mass function ρ (on the natural numbers), given via two vectors $\mathbf{d} = (d_0, d_1, \ldots, d_{s-1})^\intercal$ and $\boldsymbol{\alpha} = (\alpha_0, \alpha_1, \ldots, \alpha_{s-1})^\intercal \in [0,1]^s$, with $\sum_{i\in[s]} \alpha_i = 1$ and $s \geqslant 1$, via $\Pr(D_x = d_i) = \rho(d_i) = \alpha_i$ for all $x \in U$. We define $G^{\mathbf{d}}_{n,m,\boldsymbol{\alpha}}$ to be the bipartite graph representation of this new scheme denoted by

$$G^{d}_{n,m} \rightsquigarrow G^{\mathbf{d}}_{n,m,\boldsymbol{\alpha}}\,,$$

where the edge size d is substituted by the vector $\mathbf{d}$, and the expected degree of each left node is $\boldsymbol{\alpha}^\intercal \cdot \mathbf{d}$. Analogously, we obtain the generalization for the hypergraph and matrix representations which we denote by

$$H^{d}_{m,n} \rightsquigarrow H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}} \text{ and } \boldsymbol{M}^{d}_{n,m} \rightsquigarrow \boldsymbol{M}^{\mathbf{d}}_{n,m,\boldsymbol{\alpha}}\,.$$

While studying hypergraphs in the context of d-ary cuckoo hashing, the authors of [DGM+10] described how to extend the analysis of 2-cores from $H^{d}_{m,n}$ to $H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ which directly leads to the following theorem. Consider the generalized *key function*, cf. Section 2.3.1,

$$\operatorname{key}(\lambda, \mathbf{d}, \boldsymbol{\alpha}) = \frac{\lambda}{\sum\limits_{i\in[s]} \alpha_i \cdot d_i \cdot \big(\Pr\left(\operatorname{Po}[\lambda] \geqslant 1\right)\big)^{d_i - 1}}\,.$$

For each fixed pair $\mathbf{d}, \boldsymbol{\alpha}$ we define $\check{c}(\mathbf{d}, \boldsymbol{\alpha})$ to be the global minimum of $\operatorname{key}(\lambda)$, i. e.,

$$\check{c}(\mathbf{d}, \boldsymbol{\alpha}) \coloneqq \min_{\lambda>0} \operatorname{key}(\lambda, \mathbf{d}, \boldsymbol{\alpha})\,.$$

Theorem 4.4 (2-Core Appearance, general. of, e. g., [Mol04, Thm. 1.2])
Let $s \geqslant 1$, $\mathbf{d} = (d_i)_{i\in[s]}$, with $d_i \geqslant 3$ for all $i \in [s]$, and $\boldsymbol{\alpha} = (\alpha_i)_{i\in[s]} \in [0,1]^s$, with $\sum_{i\in[s]} \alpha_i = 1$, be constants. Then for any constant $\varepsilon > 0$ a. a. s. the following holds:

(i) If $c \leqslant \check{c}(\mathbf{d}, \boldsymbol{\alpha}) - \varepsilon$, then $G^{\mathbf{d}}_{n,m,\boldsymbol{\alpha}}$ (type B) admits an order generating matching.

(ii) If $c \geqslant \check{c}(\mathbf{d}, \boldsymbol{\alpha}) + \varepsilon$, then $G^{\mathbf{d}}_{n,m,\boldsymbol{\alpha}}$ (type B) admits no order generating matching.

Or equivalently:

(i) If $c \leqslant \check{c}(\mathbf{d}, \boldsymbol{\alpha}) - \varepsilon$, then $H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ (type B) has an empty 2-core.

(ii) If $c \geqslant \check{c}(\mathbf{d}, \boldsymbol{\alpha}) + \varepsilon$, then $H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ (type B) has a non-empty 2-core.

Using the outline from [DGM+10, Section 4], we prove part (i) of Theorem 4.4 along the lines of [Mol04, Theorem 1.2] in Section 4.4; furthermore, we sketch the proof of part (ii) along the lines of [Kim06, Theorem 1.7]. While Theorem 4.4 does also hold for type A and type C, we restrict ourselves to type B for simplicity.

Remark. Analogous to [Mol04, Theorem 1.2] and [Kim06, Theorem 1.7], the theorem can be generalized such that it covers ℓ_+-cores for all $\ell_+ \geqslant 2$, see Section 4.4. Furthermore, the 2-core threshold is often given for random hypergraphs slightly different from $H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$. A justification that some "common" hypergraph models are equivalent in terms of this threshold is given in Section 4.3.2.

Related Optimization Problems As mentioned before, our graph model is motivated by irregular bipartite graphs or hypergraphs that underlie certain erasure correcting codes such as Tornado codes [LMS+97, LMSS01], LT codes [Lub02], Online codes [May02], and Raptor codes [Sho06]. Each of these codes heavily rely on one or more random bipartite graphs or hypergraphs where the left nodes or hyperedges correspond to variables, often denoted as input/message symbols, and the right nodes or hypernodes correspond to constraints on these variables, often denoted as encoding/check symbols. An essential part of the decoding process of an encoded message, more precisely the recovery process, is the application of a procedure that can be interpreted as peeling, where it is required that the result is an empty 2-core, see Section 2.3.1. Given n message symbols, carefully designed right-irregular bipartite graphs or irregular hypergraphs allow, in contrast to regular ones, to gain codes where in the example of Tornado, Online, and Raptor codes a random set of $(1+\varepsilon)\cdot n$ encoding symbols are necessary to decode the whole message in linear time w. h. p., and in the case of LT codes a random set of $n + o(n)$ encoding symbols are necessary to decode the whole message in time proportional to $n \cdot \ln(n)$ w. h. p.. As mentioned in [GM11, footnote page 796 (5)], one could expect similar improvements concerning the 2-core threshold when switching from left-regular graphs $G^{d}_{n,m}$ or uniform hypergraphs $H^{d}_{m,n}$ to left-irregular graphs $G^{\mathbf{d}}_{n,m,\boldsymbol{\alpha}}$ or non-uniform hypergraphs $H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$.

Results

! We consider hypergraphs $H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ of type B as in Theorem 4.4.

Define $\check{c}(\mathbf{d})$ to be a maximum threshold $\check{c}(\mathbf{d},\boldsymbol{\alpha})$ among all valid $\boldsymbol{\alpha}$, i. e.,

$$\check{c}(\mathbf{d}) \coloneqq \max_{\boldsymbol{\alpha}} \check{c}(\mathbf{d},\boldsymbol{\alpha})\,, \tag{OPT}$$

and define $\boldsymbol{\alpha}^*$ to be an optimal vector according to

$$\boldsymbol{\alpha}^* \in \{\boldsymbol{\alpha} \mid \check{c}(\mathbf{d},\boldsymbol{\alpha}) = \check{c}(\mathbf{d})\}\,.$$

Note that there must be some maximum threshold $\check{c}(\mathbf{d})$ and corresponding $\boldsymbol{\alpha}^*$ for compactness reasons. We prove the following.

$\mathbf{d}$	$(3,3)$	$(3,4)$	$(3,6)$	$(3,8)$	$(3,10)$	$(3,12)$	$(3,14)$	$(3,16)$	$(3,21)$
$\check{c}(\mathbf{d})$	0.81847	0.82151	0.83520	0.85138	0.86752	0.88298	0.89761	0.91089	0.92004
α^*	1	0.83596	0.85419	0.86512	0.87315	0.87946	0.88464	0.88684	0.88743
$\bar{d}$	3	3.16404	3.43744	3.67439	3.88795	4.08482	4.26898	4.47102	5.02626

Table 4.1.2.: Optimal 2-core thresholds $\check{c}(\mathbf{d}) = \check{c}(\mathbf{d}, \boldsymbol{\alpha}^*)$ for selected $\mathbf{d} = (d_0, d_1)$, where $\boldsymbol{\alpha}^* = (\alpha^*, 1-\alpha^*)$ and $\bar{d} = \alpha^* \cdot d_0 + (1-\alpha^*) \cdot d_1$. The values are rounded to the nearest multiple of 10^{-5}.

THEOREM 4.5 (MAXIMUM 2-CORE THRESHOLD (INFORMAL))
There is an algorithm that on input $\mathbf{d} = (d_0, d_1)$ either gives the threshold $\check{c}(\mathbf{d})$ in analytical form or uses binary search in order to obtain an optimal vector $\boldsymbol{\alpha}^* = (\alpha^*, 1-\alpha^*)$, within a subset of the interval $(0,1]$, which allows to determine $\check{c}(\mathbf{d})$ numerically with arbitrary precision.

Interestingly, it turns out that for adequate edge sizes d_0 and d_1 the maximum 2-core threshold $\check{c}(\mathbf{d})$ exceeds the maximum 2-core threshold for the left-regular case, i. e., $\check{c}(d)$ for $d = 3$. Table 4.1.2 lists some values. This allows us to build *irregular immutable Bloomier filters* that rely on Algorithm 7 with improved space consumption compared to the standard construction. More comprehensive tables for parameters $3 \leqslant d_0 \leqslant 6$ and $d_0 \leqslant d_1 \leqslant 50$ are given in Appendix A.2. The *maximum threshold found* is about 0.92 for $\mathbf{d} = (3, 21)$.

Numerical experiments indicate that with larger vectors $\mathbf{d}$ the value of 0.92 can be surpassed. For example using the approach

$$\alpha_i = \frac{\left(\prod_{j \in [s]\colon j \neq i} d_j\right)^{1+\theta}}{\sum_{i \in [s]} \left(\prod_{j \in [s]\colon j \neq i} d_j\right)^{1+\theta}} \text{ for all } i \in [s],$$

we get a threshold of $\check{c}(\mathbf{d}, \boldsymbol{\alpha}) \approx 0.93220$ with expected degree $\bar{d} = \boldsymbol{\alpha}^\intercal \cdot \mathbf{d} \approx 7$, using parameters $d_0 = 3$, $d_1 = 21$, $d_2 = 151$, and $\theta = 0.06355$. It is conjectured that the thresholds can be made arbitrarily close to 1.

CONJECTURE 4.1.1 (ARBITRARILY SMALL GAP [PAN12]). For each $\varepsilon > 0$ there exists vectors $\mathbf{d}$ and $\boldsymbol{\alpha}$ such that $\check{c}(\mathbf{d}, \boldsymbol{\alpha}) \geqslant 1 - \varepsilon$.

EXEMPLARY PARAMETER CHOICE FOR AN IRREGULAR BLOOMIER FILTER Substituting the underlying graph $G^d_{n,m}$ of the immutable Bloomier filter by $G^{\mathbf{d}}_{n,m,\boldsymbol{\alpha}}$ allows to increase the space consumption with increasing expected left degree, while simultaneously relying on Algorithm 7 for the construction. For example, using $\mathbf{d} = (3, 16)$ allows to set $m = 1.1 \cdot n$ in the irregular case compared to $m = 1.22 \cdot n$ in the regular case, at the price that the number of cell probes grows from 3 to about 4.5 in expectation.

4.1.3. Mutable Bloomier Filter

The mutable Bloomier filter for solving perfect hashing in essence applies retrieval as described above. The difference is that instead of using pairwise distinct addresses directly as values $\boldsymbol{\nu} \in V^m$, like suggested by Majewski, Wormald, Havas, and Czech, see [MWHC96], Chazelle et al. applied retrieval to a vector $\boldsymbol{\nu}$ of indices of hash functions whose hash values are pairwise distinct. This reduces the size of V and therefore the overall space usage dramatically. The approach was later independently rediscovered by Botelho, Pagh, and Ziviani [BPZ07, BPZ13].

Remark. Another kind of mutable Bloomier filter for retrieval, without solving perfect hashing in the first place, can be obtained via determining the table $\mathbf{t}$ such that for all keys $x \in S$ the function value $g(x) = \nu_x$ appears at least $k = \lfloor d/2 \rfloor + 1$ times in the table cells address by A_x; on lookup the most frequent entry is returned. Here, the maximum load is bounded by the threshold for a $(k, 1)$-orientation which has a (local) maximum of about 0.212 for $d = 5$ and $k = 3$.

4.1.3.1. Construction

! In the following we describe a slightly more general approach than discussed in [CKRT04] and [BPZ13].

In order to build a realization $\mathfrak{D}$ of a perfect hash function, we use *two basic schemes* (not necessarily different) and apply a two-phase construction.

First Phase Using the bipartite graph representation $G^{\acute{d}}_{n,\acute{m}} = (S \cup [\acute{m}], E)$ of the first basic scheme we obtain a left-perfect matching

$$M \subseteq \left\{ \{x_i, \acute{h}_j(x)\} \mid i \in [n], j \in [\acute{d}] \right\}.$$

Then, we generate a vector $\boldsymbol{\nu} = (\nu_i)_{i \in [n]} \in V^n$ with $(\mathbb{Z}_{\acute{d}}, +)$ and $\nu_i = j$ if $\{x_i, \acute{h}_j(x)\}$ is a matching edge. That is, for each key $x \in S$ we store the index of the hash function that belongs to its matching edge. By definition of left-perfect matching, each value ν_i is well-defined and we have $\acute{h}_{\nu_i}(x_i) \neq \acute{h}_{\nu_j}(x_j)$ for all $\{x_i, x_j\} \in \binom{S}{2}$. The number of right nodes $\acute{m}$ will become the range of the perfect hash function.

Second Phase Using the vector $\boldsymbol{\nu} \in V^n$ and the matrix $\mathbf{M}^d_{n,m}$ of the second basic scheme, we build a retrieval data structure as in Section 4.1.1 and obtain a vector $\mathbf{t} \in V^m$ as solution of the system of equations (SoE).

Classical Variant Usually, the construction is described in a simplified way where one uses one and the same graph in both phases [CKRT04, BPZ13]. In the first phase an order generating matching is obtained and the same matching is used to solve (SoE) in the second phase.

4.1.3.2. Lookup Operation

For a given key the return value of the perfect hash function is determined as follows. First we retrieve an index of a hash function used in the first phase and then obtain its hash value, i. e., for all $x \in U$ we define

$$\texttt{lookup}(\mathfrak{D}, x) \coloneqq \acute{h}_j(x) \text{ with } j \coloneqq \left(\sum_{a \in A_x} t_a \right) \bmod \acute{d}.$$

4.1.3.3. Success Probability

Clearly, the whole construction is successful if and only if the first and second phase are successful. Given $\acute{d}$ and d, the upper bounds for the load factors $\acute{c} = n/\acute{m}$ and $c = n/m$ up to which a. a. s. a construction is possible have been discussed in detail in Sections 3.1.1.3 and 4.1.1.3.

4.1.3.4. Space and Time Consumption and Range

With respect to perfect hashing we now have a third competing performance metric, the range $\acute{m}$ of the perfect hash function. Let $\acute{\varepsilon} \in (0, 1)$ and $\acute{m} = (1 + \acute{\varepsilon}) \cdot n$, further let $\varepsilon \in (0, 1)$ and $m = (1 + \varepsilon) \cdot n$. Assuming ideal hash functions, see Section 2.2.2, a mutable Bloomier filter can be constructed a. a. s. with

range $(1 + \acute{\varepsilon}) \cdot |S|$,

space usage $(1 + \varepsilon) \cdot |S| \cdot \lceil \log \log(1/\acute{\varepsilon}^{O(1)}) \rceil$, and

lookup time $O\big(\log(1/\varepsilon)\big)$,

utilizing that $\acute{d} = O\big(\log(1/\acute{\varepsilon})\big)$ and $d = O\big(\log(1/\varepsilon)\big)$ for $\acute{d} \geqslant 3$ and $d \geqslant 3$. For more insight concerning the space consumption, we consider the space usage per key and omit asymptotic notation. From [FPSS05], we obtain $\acute{d} = \big(1 + \acute{\delta}(\acute{d})\big) \cdot \ln\big(1 + 1/\acute{\varepsilon}(\acute{d})\big)$, where $\acute{\delta} \in (0, 0.2)$ and $\acute{\varepsilon} \leqslant 0.09$. So, if we ignore $\lceil\rceil$, i. e., we assume some ideal compression scheme, the space usage per key is

$$\big(1 + \varepsilon(d)\big) \cdot \log(\acute{d}) = \big(1 + \varepsilon(d)\big) \cdot \frac{\ln\Big(\big(1 + \acute{\delta}(\acute{d})\big) \cdot \ln\Big(1 + \frac{1}{\acute{\varepsilon}(\acute{d})}\Big)\Big)}{\ln 2}.$$

For $\acute{m} = (1 + \acute{\varepsilon}) \cdot n$ and assuming that the universe U has size polynomial in n, the worst-case asymptotic space consumption per key is bounded from below by

$$\frac{1 + \acute{\varepsilon} \cdot \ln\left(\frac{\acute{\varepsilon}}{1+\acute{\varepsilon}}\right)}{\ln 2}, \text{ for } \acute{\varepsilon} > 0,$$

see Section 4.3.1. With respect to this bound the overhead is about $0.465 + \varepsilon \cdot 1.585$ bits per key for $\acute{d} = 3$ and $\acute{\varepsilon} \approx 0.0894$. This difference increases monotonically with increasing $\acute{d} \geqslant 3$ and goes to ∞ for $\acute{\varepsilon} \to 0$.

4.1.3.5. Liner Time Algorithms

We have discussed linear time algorithms for each of the two construction phases in Sections 3.1.1.5 and 4.1.1.5. Combining them to an algorithm for the mutable Bloomier filter is straightforward. Note that the generalized selfless algorithm for finding a left-perfect matching in the first phase is only *conjectured* to work up to the threshold $\hat{c}(\acute{d})$ a. a. s., see Section 3.4.

4.1.4. Irregular Mutable Bloomier Filter

As in the case of the immutable Bloomier filter, practical problems arise if we want to solve the system of equations (SoE) in the second phase and $c = n/m$ is slightly below $\hat{c}(d)$. Since then, up to our knowledge, the best we can do is to split the matrix $\boldsymbol{M}^d_{n,m}$ into sub-matrices of almost equal but constant size and solving each of the corresponding small equation systems via relatively slow system solvers, see Section 4.1.1.5. However, as discussed in Section 4.1.2, an alternative is to replace the left-regular bipartite graph $G^d_{n,m}$ by a left-irregular bipartite graph $G^{\mathbf{d}}_{n,m,\boldsymbol{\alpha}}$, which leads to an irregular mutable Bloomier filter. Using $G^{\mathbf{d}}_{n,m,\boldsymbol{\alpha}}$ with suitable parameters $\mathbf{d}$ and $\boldsymbol{\alpha}$, the simple and fast peeling approach (Algorithm 7) a. a. s. obtains a solution of the corresponding system of equations at least up to $c = 0.92$.

Results

We demonstrate the practicability of this approach for constructing a *space and time efficient perfect hash function with small range*. More precisely, for a key set S of strings, we show in experiments that with common (non-ideal) underlying hash functions of small description size, one fastly obtains an irregular mutable Bloomier filter with range $1.1 \cdot n$ and space usage $1.1 \cdot \lceil \log 3 \rceil = 2.2$ bits per key, for $n \geqslant 10^5$. Applying a simple compression technique that stores every 5 consecutive elements from $\mathbf{t}$ in one byte, we then obtain a space usage of $1.1 \cdot 8/5 = 1.76$ bits per key for $n \geqslant 10^7$.

4.1.5. Overview of the Chapter

The next section presents some background information on perfect hashing and retrieval with focus on perfect hashing. A discussion of standard space bounds followed by some asymptotically nearly equivalent hypergraph models is covered in Section 4.3. Using these models, we give a proof of the first part of Theorem 4.4 in the subsequent Section 4.4. Thereafter, we focus on the main results of this chapter, where in Section 4.5 we determine thresholds $\check{c}(\mathbf{d})$ for random bipartite graphs $G^{\mathbf{d}}_{n,m,\boldsymbol{\alpha}}$ with two different left degrees according to Theorem 4.4; and in Section 4.6 we show experimental measurements on the linear time construction of a space efficient perfect hash function based on results of the previous section. We close with a short summary.

4.2. Further Background and Related Work

In this section, we discuss some background on perfect hashing and retrieval. This is only for informational purpose and not required for the rest of the chapter. A reader familiar with these topics can skip this section.

4.2.1. Perfect Hashing

Let $|U| = u$, $|S| = n$, and $|R| = \acute{m}$. A function

$$h\colon U \to R$$

is called perfect hash function for a key set $S \subseteq U$ if h is injective with respect to S. Given such a function, solving dictionary, membership, and retrieval is trivial. The only other ingredient that is needed is a table $\acute{t} = (\acute{t}_i)_{i\in[\acute{m}]}$ constructed according to

$$\acute{t}_{h(x)} \coloneqq \begin{cases} (x, v_x), & \text{dictionary} \\ x, & \text{membership tester} \\ v_x, & \text{retrieval data structure} \end{cases},$$

for all $x \in S$, with some associated value v_x. Moreover, suppose a perfect hash function h with range of size $\acute{m} = n$ can be provided for free, then this simple hash table matches the space lower bounds of dictionary and retrieval and almost matches the space lower bound for membership, see Sections 3.3.1 and 4.3.1.

4.2.1.1. Space Bounds

According to a lower bound by Mehlhorn, see [Meh82, Theorem A], any data structure that realizes a *perfect hash function* with range $\acute{m}$ needs at least $\Omega(n^2/\acute{m} + \log\log u)$ bits in the worst-case, cf., Lemma 4.3.2. That is, given n and u, the space consumption of a perfect hash function can be made negligible only for large $\acute{m}$. However, such a large range is not very attractive, since the perfect hash function is typically used with some table $\acute{t} = (\acute{t}_i)_{i\in[\acute{m}]}$ as described above, which leads to a very high space consumption for the overall data structure.

! Hence, for the rest of the section, we will solely concentrate on the case $\acute{m} = \Theta(n)$.

Often a perfect hash function h is preferred, or required, to be surjective as well, i. e., has range $\acute{m} = n$. Such a *minimal perfect* hash function needs at least $n \cdot \log e + \log\log u - O(\log n)$ bits in the worst-case, as proven by Fredman and Komlós [FK84], assuming that $u > n^{2+\varepsilon}$, for constant $\varepsilon > 0$.

A further common constraint, which also strongly influences the space consumption of h, is that any given total order "$\leqslant$" of the keys must be also respected by the hash values, i. e., $x \leqslant y \Rightarrow h(x) \leqslant h(y)$ for all x, y from S. The additional space

consumption of such an *order preserving* (minimal) perfect hash function is about $n \cdot \log n$, which can be directly derived from the lower bound for perfect hashing by Mehlhorn [Meh84, Theorem 6 a] with an additional factor of $n!$ for all possible total orderings (permutations) of the key set, see, e. g., [FCDH90, Section 3.1].

A special case arises if the construction algorithm is restricted to one fixed total order, usually lexicographical. For such a *monotone* (minimal) perfect hash function a space consumption of $O(n \cdot \log\log\log u)$, with $n \geqslant \log\log u$, is sufficient [BBPV09, Theorem 6.1].

Remark. There exists generalizations of perfect hashing like k-*probe* hashing [SS88, SS90], which however, we will not cover in the following.

4.2.1.2. Classical Constructions

For a range of size superlinear in n, space and time efficient perfect hash functions have been known for a long time. A classical construction, described for example in [JvEB86, Fact 1 and 2] and [SS88, page 10], allows to solve perfect hashing with $\acute{m} = n^2$, constant evaluation time, and space consumption $O(\log n + \log\log u)$. It can be used as building block for the construction of space and time efficient perfect hash functions with smaller range, essentially reducing the size of the domain to be handled, which is known as *collapsing the universe*, see Section 5.4.1.

A seminal result by Fredman, Komlós, and Szemerédi [FKS84] is the construction of a perfect hash function with range $\Theta(n)$, usually referred as *FKS scheme*, that has constant evaluation time and, used in combination with the mentioned perfect hash function[2] of range $\acute{m} = n^2$, has a space consumption of $O(n \cdot \log n + \log\log u)$.

Refinements of their approach allowed to further reduce the space usage while maintaining constant evaluation time, as, e. g., the construction by Jacobs and van Emde Boas [JvEB86, Theorem 2], which requires $O(n \cdot \log\log n + \log\log u)$ bits. This development culminated in a well-known construction by Schmidt and Siegel [SS88, Theorem 7] [3] that needs space $O(n + \log\log u)$, i. e., is space optimal up to a constant factor.

Remark. Other improvements of the FKS scheme provide dynamic insertion and deletion in expected constant amortized time, see [DKM+88, DKM+94]. However, even more than in Section 3.2 concerning hash tables, in this chapter we concentrate on the static case, i. e., we assume that the key set S is fixed and given in advance.

With respect to minimal perfect hashing, i. e., the case $n = \acute{m}$, Mehlhorn gave a construction with an almost matching upper space bound of $n \cdot \log e + \log\log u + O(\log n)$ [Meh82, Proof of Lemma 3], [Meh84, Section 3.2.3, Proof of Theorem 8]. However, evaluation time as well as construction time of the data structure are exponential in n.

[2] Collapsing the universe is already contained in [FKS84, Corollary 2 and Lemma 2].
[3] A variation of FKS scheme with new internal hash functions and sophisticated compression.

The best theoretical construction of a minimal perfect hash function, introduced by Hagerup and Tholey [HT01, Theorem 1], achieves a near optimal space consumption of $n \cdot \log e + \log\log u + O\big(n \cdot (\log\log n)^2 / \log n + \log\log\log u\big)$. They use a randomized variant of Mehlhorn's approach, which is applied on small disjoint subsets of the key set S, thereby achieving constant evaluation time and $O(n + \log\log u)$ expected construction time. The practicability of this construction is discussed, e. g., in [Bot08, Section 1.6.1].

4.2.1.3. Practical Constructions

Perfect hashing has been studied since decades theoretically as well as experimentally. For a comprehensive, but somewhat outdated, survey that covers a variety of techniques see [CHM97]. In the following, leaning on [Die07], we discuss two of these techniques and corresponding results, which allow very efficient and practical *randomized* constructions of (minimal) perfect hash functions and order preserving (minimal) perfect hash functions.

The functions that we cover are realized via data structures[4] that mainly consist of helper hash functions and a table $\mathbf{t} = (t_i)_{i \in [m]}$ of m cells that usually dominates the overall space consumption.

Displacement Based on an idea used before, among others, by Tarjan and Yao [TY79, Section 3 row displacement], Pagh introduced the following construction principle for perfect hash functions, see [Pag99]. Two helper hash functions, $h^{\text{row}} \colon U \to [m]$ and $h^{\text{col}} \colon U \to [\acute{m}]$, define a random matrix

$$\boldsymbol{A} = (a_{i,j})_{i \in [m], j \in [\acute{m}]}, \text{with } a_{i,j} \coloneqq |\{x \in S \mid h^{\text{row}}(x) = i \ \wedge \ h^{\text{col}}(x) = j\}|\,,$$

that is required to be binary. For each subset of keys $x \in S$ that have the same index $h^{\text{row}}(x) = i$, i. e., correspond to the same row of $\boldsymbol{A}$, a displacement value t_i is determined, such that shifting each row i of $\boldsymbol{A}$ circularly by t_i results in a matrix with column weight at most 1. That implies that for all $x \in S$ the values $\big(h^{\text{col}}(x) + t_{h^{\text{row}}(x)}\big) \bmod \acute{m}$ are pairwise distinct. Hence, this scheme defines a perfect hash function $h \colon U \to [\acute{m}]$ according to

$$\begin{aligned} h(x) \coloneqq &\big(h^{\text{col}}(x) + t_{h^{\text{row}}(x)}\big) \bmod \acute{m}, \text{ where} \\ &h^{\text{row}} \colon U \to [m], h^{\text{col}} \colon U \to [\acute{m}], \mathbf{t} \in [\acute{m}]^m\,. \end{aligned}$$

In order to find appropriate displacement values, Pagh suggested an algorithm called *random-fit-decreasing* that treats the rows of $\boldsymbol{A}$ in order of decreasing weight and chooses the correcting displacement values randomly. With this method (minimal) perfect hash functions with space consumption $(2 + \varepsilon) \cdot n \cdot \log \acute{m}$, for constant $\varepsilon > 0$,

[4] Implementations of some of them are available under `http://cmph.sourceforge.net/`.

and constant evaluation time can be obtained in $O(n)$ expected time [Pag99, Lemma 4 with Theorem 5].

REMARK. The group $(\mathbb{Z}_{\acute{m}}, +)$, which is chosen as range of h^{col} and h, can be replaced by any other group with $\acute{m}$ elements.

Using the same scheme but a different algorithm, called *undo-one*, Dietzfelbinger and Hagerup could reduce the table size from $m = (2+\varepsilon)\cdot n$ to $m = (1+\varepsilon)\cdot n$ for constant $\varepsilon > 0$, while maintaining the upper bounds for evaluation and construction time [DH01, Theorem 8]. The undo-one algorithm uses the random-fit-decreasing algorithm for rows down to weight 3, but now for each computation of the displacement of a weight-2 row, a displacement determined before can be changed. A further asymptotic space reduction to $O(n \cdot \log\log n)$, using Pagh's approach in combination with a splitting hash function as in [HT01], was proposed by Woelfel [Woe06, Theorem 3].

Another variation of Pagh's approach, introduced by Belazzougui, Botelho, Dietzfelbinger [BBD09], uses d displacement *functions* $h_0^{col}, h_1^{col}, \ldots, h_{d-1}^{col}$ instead of displacement *values* and defines a perfect hash function h according to

$$h(x) := h^{col}_{t_{h^{row}(x)}}(x), \text{ where}$$
$$h^{row}\colon U \to [m], h_i^{col}\colon U \to [\acute{m}] \text{ for all } i \in [d], \mathbf{t} \in [d]^m .$$

Assuming ideal hash functions h^{row}, $h_0^{col}, h_1^{col}, \ldots, h_{d-1}^{col}$, as defined in Section 2.2.2, they showed how this modification in combination with compression allows to build perfect hash functions with range[5] $\acute{m} = (1+\acute{\varepsilon})\cdot n$, expected space consumption $n\cdot\log e + n\cdot\zeta(\lambda) + O(\lambda^2)$, and expected construction time $O\big(n\cdot(2^\lambda + (1/\acute{\varepsilon})^\lambda)\big)$ [BBD09, Theorem 1 and 2]. Here, ζ denotes a scale factor, usually smaller than 1, given via some compression scheme. The value λ controls the tradeoff between space and time consumption, since ζ decreases with increasing λ, while the construction time increases. It turned out in experiments, that even for very small ranges a space consumption of less than 2 bits per key is achievable in a practicable amount of time, see [BBD09, Figure 1].

LINEAR SYSTEM In parallel, another efficient construction approach emerged, using perfect hash functions of the form

$$h(x) := (t_{h_0(x)} + t_{h_1(x)} + \ldots + t_{h_{d-1}(x)}) \bmod \acute{m}, \text{ where}$$
$$h_i\colon U \to [m] \text{ for all } i \in [d], \mathbf{t} \in [\acute{m}]^m ,$$

for $d \geqslant 2$, where each key x is considered as the label of an edge $A_x = \{h_i(x) \mid i \in [d]\}$ of a random (hyper-)graph $H^d_{m,n}$ that has binary incidence matrix

$$M^d_{n,m} = (m_{x,j})_{x\in S, j\in[m]}, \text{ where } m_{x,j} := \operatorname{sgn}\big(|\{i \in [d] \mid h_i(x) = j\}|\big) .$$

[5] A minimum perfect hash function can be derived using, e. g., further compression as discussed in [BBD09, Section 3].

A perfect hash function h is found via solving the system of equations $\mathbf{M}^d_{n,m} \cdot \mathbf{t} = \boldsymbol{\nu}$, for some arbitrary vector $\boldsymbol{\nu} = (\nu_x)_{x \in S} \in [\acute{m}]^n$ of pairwise distinct hash values. If we ignore the helper hash functions h_i, $i \in [d]$, then the space consumption of h is essentially the number of bits needed to store a solution $\mathbf{t}$, which is about $m \cdot \log \acute{m}$.

A sufficient condition for the existence of such a solution is that $H^d_{m,n}$ has an empty 2-core, which was first explored in this context in a series of works by Czech, Havas, and Majewski [CHM92] together with Wormald [HMWC93, MWHC96]. They argued that, assuming ideal hash functions, via peeling with back substitution an *order preserving* perfect hash function h can be constructed in expected linear time that has a space consumption of $(1/\check{c}(d) + \varepsilon) \cdot n \cdot \log \acute{m}$, for constant $\varepsilon > 0$, see Section 4.1.1.5.

Another approach, proposed in [SH94][6], originated from [Section 2][GS89][7], is to solve the system of equations over some field using standard methods, like, e. g., Gaussian elimination or Wiedemann's algorithm [Wie86] in time at least quadratic in n. Upper bounds for the minimum space consumption that is a. a. s. achievable with this approach were derived in a related context due to a series of independent work by Dietzfelbinger and Pagh [DP08a], Porat [Por08], and shortly after, Charles and Chellapilla [CC08a] — for the full versions see [DP08b, CC08b, Por09].

REMARK. In [DP08a, DP08b] the authors apply a result by Calkin [Cal97] stating that if c is bounded away by a constant from a threshold $\hat{c}'(d)$, with $\check{c}(d) < \hat{c}'(d) < \hat{c}(d)$, then w. h. p. $\mathbf{M}^d_{n,m}$ has full row rank. This gives a space bound of $(1/\hat{c}'(d) + \varepsilon) \cdot n \cdot \log \acute{m}$, for constant $\varepsilon > 0$. In Section 5.4.4 the success probability of this result is improved from $1 - O(n^{-1})$ to $1 - O(n^{-\Omega(d)})$ for sufficiently large d.

Finally, the lower space bound that is a. a. s. obtainable by such a direct solution of the system of equations was found to be $(1/\hat{c}(d) + \varepsilon) \cdot n \cdot \log \acute{m}$, for constant $\varepsilon > 0$, where $\hat{c}(d)$ is the threshold when the edge density in the 2-core of $H^d_{m,n}$ approaches 1, see [DM02, DGM+10, PS12] as well as Section 4.1.1.3.

A nifty improvement, introduced by Chazelle et al. [CKRT04], and later independently rediscovered by Botelho, Pagh, and Ziviani [BPZ07, BPZ13], replaces the hash *values* ν_i, $i \in [n]$, by the indices of the d hash *functions*[8] and defines a perfect hash function h according to

$$h(x) := h_{(t_{h_0(x)} + t_{h_1(x)} + \ldots + t_{h_{d-1}(x)}) \bmod d}(x), \text{ where}$$
$$h_i \colon U \to [m] \text{ for all } i \in [d], \mathbf{t} \in [d]^m .$$

Assuming ideal hash functions, and using the peeling approach, a space consumption of $(1/\check{c}(d) + \varepsilon) \cdot n \cdot \log d$, for constant $\varepsilon > 0$, can be obtained maintaining expected linear construction time. This allows to build perfect hash functions with 1.95 bits per

[6] Actually, they use the definition $m_{x,j} := |\{i \in [d] \mid h_i(x) = j\}|$, i. e., $\mathbf{M}^d_{n,m}$ is not necessarily binary anymore.

[7] This can be seen if one considers Equation 2.4 in combination with Example 1 from [GS89].

[8] In principle these indices could refer to other hash functions than $h_0, h_1, \ldots, h_{d-1}$, i. e., especially $m \neq \acute{m}$ can be useful, as we will discuss in Section 4.6.

key and minimal perfect hash functions with 2.62 bits per key in a practicable amount of time, see [BPZ13, Section 3.5 and 3.6, Table 7] and [BBD09, Figures 1 a and d].

HYBRID A third approach, which lies in between the first two, was taken by Sager [Sag85], who proposed perfect hash functions h according to the following scheme

$$\begin{aligned} h(x) \coloneqq &(h_2(x) + t_{h_0(x)} + t_{h_1(x)}) \bmod \acute{m}, \text{ where} \\ &h_0, h_1 \colon U \to [m], h_2 \colon U \to [\acute{m}], \mathbf{t} \in [\acute{m}]^m . \end{aligned}$$

His idea was further pursued by Fox, Chen, Heath, and Datta [FCHD89], as well as Fox, Chen, Heath, and Daoud [FCDH90, FCDH91, FHCD92], using various hash functions h_0, h_1, and h_2 and construction algorithms. Finally, this lead to an algorithm for finding minimal perfect hash functions, which worked in experiments with space consumption $0.6 \cdot n \cdot \log n$ in linear time [FHCD92, Algorithm 1].

For a perfect hash function h based on the related scheme

$$\begin{aligned} h(x) \coloneqq &(t_{h_0(x)} + t_{h_1(x)}) \bmod \acute{m} \text{ where} \\ &h_0, h_1 \colon U \to [m], \mathbf{t} \in [\acute{m}]^m , \end{aligned}$$

Weidling [Wei04] and independently Botelho, Kohayakawa, and Ziviani [BKZ05] presented similar algorithms with expected linear running time that can be seen as a combination of the methods from [CHM92] and [Pag99]. Assuming ideal hash functions h_0 and h_1, Weidling showed that his algorithm works for $m \geqslant 1.152 \cdot n$, i. e., with a space consumption of at least $1.152 \cdot n \cdot \log \acute{m}$ [Wei04, Satz 1], while experiments indicate that the actual lower bound is $m = 1/3 \cdot n$, see [Wei04, Abbildung 2.8]. Moreover, he proved a modified version of his algorithm, based on [DH01], to work for $m \geqslant 0.936 \cdot n$ [Wei04, Satz 3].

Botelho, Kohayakawa, and Ziviani conjectured a lower bound of $m \geqslant 1.152 \cdot n$ for their algorithm [BKZ05, page 492 and Conjecture 1], and they presented a modification that worked in experiments for $m \geqslant 0.93 \cdot n$.

REMARK. With the hybrid approach space factors $m/n = 1/c$ that are much lower than $1/\check{c}(2) = 1/\hat{c}(2) = 2$ are achievable, since the construction algorithms do not rely on the fact that each key x from S can be associated with a value $v = h(x)$ in advance. That is, in order to determine (some parts of) $\mathbf{t}$ no linear system is solved. In turn, the resulting hash function h is not order preserving.

4.2.2. RETRIEVAL

The retrieval problem is usually solved via perfect hashing, in the manner as sketched in Section 4.2.1. However, this intermediate step is not necessary for an efficient solution as pointed out by Chazelle et al. [CKRT04]. Their idea was to build a system of equations as discussed in Section 4.2.1.3 concerning the construction of order preserving hash

functions, but use as right-hand side a vector $\boldsymbol{v}$ of given function values $\boldsymbol{v} = (f(x))_{x \in S}$. Chazelle et al. used the same method as Majewski et al. [MWHC96] in order to solve the system of equations but they only derived a lower bound[9] on the exact load thresholds $\check{c}(d)$. This approach to build a retrieval data structure was further explored simultaneously and independently by Porat [Por09], Charles and Chellapilla [CC08b], and Dietzfelbinger and Pagh [DP08b]. Like Seiden and Hirschberg [SH94], who however gave only experimental results, they considered the system of equations as sparse linear system over some field $\mathbb{F}$, in order to find a solution for parameters $n = (1 + \varepsilon) \cdot m$, and arbitrary constant $\varepsilon(d) > 0$, in particular $n/m > \check{c}(d)$, using standard methods. Including the case that the 1's in the binary matrix $\boldsymbol{M}^d_{n,m}$ are replaced by random elements from $\mathbb{F}$ via further hash functions, they derived lower bounds on the maximum ratio n/m that guarantees a. a. s. that the linear equations are independent. Moreover, maintaining constant evaluation time, they showed how to obtain expected (almost) linear construction time [Por09, Section 6], [DP08b, Theorem 1 a], [CC08b, Theorem 6], via subdividing the large linear system into smaller ones, using a standard technique known as *bucketing*. The experimental evaluation of the bucketing approach by Pagh and Dietzfelbinger exposed the limitations of this method, which is practical only for very large n [ADR09].

4.2.2.1. Relationship to Bloom Filters

A main feature of the retrieval data structures just mentioned is that they can be easily extended in order to reduce the *false positive probability*, which is the probability that a valid value, i. e., a value from $\{g(x) \mid x \in S\}$, is returned on lookup for an $x \in U - S$. The idea is to *combine* $v_x = g(x)$ for each $x \in S$ with the value $h(x)$ of an independent and uniform hash function $h\colon U \to R$. This can be done, e. g., via embedding V in R and adding $g(x)$ and $h(x)$, for a false positive probability of $|V|/|R|$ and additional space usage $m \cdot (\log|R| - \log|V|)$, or via extending V to $V \times R$, and concatenating $g(x)$ and $h(x)$, for a false positive probability of $1/|R|$ and additional space usage $m \cdot \log|R|$, cf., [DP08b, Section 1.2].

For the special case that there is only one valid value, i. e., $g(x) = v$ for all $x \in S$, a false positive probability of $1/|R|$ can be also achieved via storing v and *replacing* $g(x)$ by $h(x)$ for all $x \in S$, see, e. g., [BM03, Section 2.2] and [DP08a, Section 4]. Now, solving retrieval with a certain false positive probability becomes equivalent to solving set membership with the same false positive probability, which for a membership tester is defined as the probability that "$x \in S$" is returned on lookup for an $x \in U - S$. Hence, these extended retrieval data structures can be seen as alternatives to the *Bloom filter* [Blo70], a classical membership tester with adjustable false positive probability.

[9] Actually, they did not emphasize the connection to perfect hashing, i. e., they seem not to have been aware of the work by, e. g., Seiden and Hirschberg [SH94] and Majewski et al. [MWHC96].

Remark. This is the origin of the name *Bloomier filter*, introduced by Chazelle et al., which we retain throughout this thesis although we do not care about false positives.

There are a multitude of further Bloom filter variants, rather loosely related to the work presented here. A detailed discussion of them is beyond the scope of this thesis. For a survey see, e. g., [BM03, Rin07].

4.3. Basics

The section starts with standard space lower bounds for retrieval and perfect hashing. Thereafter, we consider several related hypergraph models, which are used in Section 4.4 for a discussion of Theorem 4.4.

4.3.1. Worst-Case Space Lower Bounds

Deriving a suitable space lower bound for retrieval is straightforward and stated here only for completeness.

> Lemma 4.3.1 (Space Bound for Retrieval). A retrieval data structure for a key set S, where each key is associated with a value from V, requires at least $|S| \cdot \log|V|$ bits in the worst-case.

The argument is the same as used in the proof for the space lower bound of a dictionary in Section 3.3.1.

More challenging are good space bounds for perfect hashing. As we have discussed in Section 4.2.1.1, a *minimal* perfect hash function needs at least $\log e = 1/\ln 2$ bits per key in the worst-case, if we have $\log\log|U| = o(|S|)$ and $|S| \to \infty$, as proved by Fredman and Komlós [FK84], which is tight in our asymptotic view, see, e. g., [Meh82]. For the *more general* case of perfect hashing with range linear in the number of keys, we use the following standard bound, see, e. g., [Bot08, Theorems 1 and 2] and [BPZ13, Theorem 1.1] for alternative formulations, in order to value the space consumption of the mutable Bloomier filter.

> Lemma 4.3.2 (Asymptotic Space Bound for Perfect Hashing, follows from [Meh84, Chapter 3, Theorem 6 a)]). Let $h\colon U \to R$ be a hash function that is injective with respect to $S \subset U$, where $|U| = |S|^l$ and $(1+\varepsilon) \cdot |S| = |R|$ for constants $l > 1$ and $\varepsilon \geqslant 0$. Then for $n \to \infty$, any realization of h requires at least the following number of bits per element from S in the worst-case:
>
> $$\frac{1}{\ln 2}, \text{ if } \varepsilon = 0, \text{ and } \frac{1+\varepsilon \cdot \ln\left(\frac{\varepsilon}{1+\varepsilon}\right)}{\ln 2}, \text{ if } \varepsilon > 0.$$

Proof. Let $|S| = n$, $|R| = m$, and $|U| = u$. The lower bound is derived straightforwardly from [Meh84, Chapter 3, Theorem 6 a)], which states that the space usage for the

realization of a function that maps from U to R and is injective on S must be at least

$$\log\left(\frac{\binom{u}{n}}{\binom{m}{n}\cdot\left(\frac{u}{m}\right)^n}\right).$$

The proof of the lemma is divided into two parts. In the first part we restate the proof of [Meh84] for completeness, and in the second part we use integration to derive an asymptotic bound for $n \to \infty$.

Part (i): There are $T_0 = \binom{u}{n}$ pairwise distinct subsets of U that have size n. We will show that each of the m^u different functions from U to R are injective for at most $T_1 = \binom{m}{n}\cdot\left(\frac{u}{m}\right)^n$ of them. It follows that we need at least T_0/T_1 different functions such that for each set of size n there is one that is injective on this set. Hence, the space lower bound is $\log(T_0/T_1)$.

Let $h\colon U \to R$ be arbitrary but fixed and let $y_0, y_1, \ldots, y_{n-1} \in R$ be pairwise distinct, then the number of subsets from U with size n that are mapped injectively to $\{y_0, y_1, \ldots, y_{n-1}\}$ via h is

$$\left|h^{-1}(y_0)\right| \cdot \left|h^{-1}(y_1)\right| \cdot \ldots \cdot \left|h^{-1}(y_{n-1})\right|.$$

It follows that h is injective for at most $\sum_{R' \in \binom{R}{n}} \prod_{y \in R'} \left|h^{-1}(y)\right|$ subsets of size n. Let $z = (z_i)_{i\in[m]}$. We are looking for a global maximum of the function

$$f(z) \coloneqq \sum_{I \in \binom{[m]}{n}} \prod_{i \in I} z_i$$

under the condition that

$$g(z) \coloneqq \sum_{i=0}^{m-1} z_i = u,$$

which exists by compactness.

> Claim 4 (Maximum Number of Subsets). The point $(u/m, u/m, \ldots, u/m)$ is the global maximum point of $f(z)$ restricted to $g(z) = u$.

Proof of Claim. Let $z^* = (z_i^*)_{i\in[m]}$ be a global maximum point of $f(z)$ restricted to $g(z) = u$. Assume for a contradiction that there exists two elements z_a, z_b of z^* with $z_a < z_b$. For all $l \in \mathbb{N}$ let $J_l = \binom{[m]-\{a,b\}}{n-l}$. Using J_0, J_1, and J_2 we rewrite f as follows

$$f(z^*) = \sum_{I\in J_0} \prod_{i\in I} z_i + (z_a + z_b)\cdot \sum_{I\in J_1} \prod_{i\in I} z_i + z_a \cdot z_b \sum_{I\in J_2} \prod_{i\in I} z_i.$$

Define $\delta = \frac{z_b - z_a}{2}$ and let z' be the result of substituting z_a and z_b in z^* with $z'_a = z_a + \delta$ and $z'_b = z_b - \delta$. Since $z'_a + z'_b = z_a + z_b$ and $z'_a \cdot z'_b > z_a \cdot z_b$, we have that $f(z') > f(z^*)$ and $g(z') = g(z^*)$, a contradiction. □

Therefore, the number of n-element subsets of U for which h is injective is at most

$$\sum_{R' \in \binom{R}{n}} \prod_{y \in R'} \frac{u}{m} = \binom{m}{n} \cdot \left(\frac{u}{m}\right)^n = T_1 \,.$$

Part (ii): The term T_0/T_1 can be written as $T_0/T_1 = \prod_{i=0}^{n-1} \frac{u-i}{u} \cdot \frac{m}{m-i}$, which gives

$$\log(T_0/T_1) = \underbrace{\sum_{i=1}^{n-1} \log\left(1 - \frac{i}{u}\right)}_{K_0(n,u)} + \underbrace{\sum_{i=1}^{n-1} \log\left(1 + \frac{i}{m-i}\right)}_{K_1(n,m)} \,.$$

The sequence $\log\left(1 - \frac{i}{u}\right)$, $0 \leqslant i \leqslant u-1$, is monotonically decreasing and the sequence $\log\left(1 + \frac{i}{m-i}\right)$, $0 \leqslant i \leqslant m-1$, is monotonically increasing. Using that $n < u$ and $n \leqslant m$, we can bound K_0 and K_1 from below via

$$\begin{aligned} K_0(n,u) &\geqslant \int_1^n \log\left(1 - \frac{x}{u}\right) dx = \left[\frac{u-x}{\ln 2} \cdot \left(1 - \ln\left(1 - \frac{x}{u}\right)\right)\right]_1^n \\ &= \frac{-n + 1 + (u-1) \cdot \ln\left(1 - \frac{1}{u}\right) - (u-n) \cdot \ln\left(1 - \frac{n}{u}\right)}{\ln 2} =: L_0(n,u) \,. \end{aligned}$$

and

$$\begin{aligned} K_1(n,m) &\geqslant \int_0^{n-1} \log\left(1 + \frac{x}{m-x}\right) dx = \left[\frac{m-x}{\ln 2} \cdot \left(-1 + \ln\left(1 - \frac{x}{m}\right)\right)\right]_0^{n-1} \\ &= \frac{n - 1 + (m-n+1) \cdot \ln\left(1 - \frac{n-1}{m}\right)}{\ln 2} =: L_1(n,m) \,. \end{aligned}$$

Hence, we get $\log(T_0/T_1) \geqslant L_0(n,u) + L_1(n,m) =: L(n,m,u)$. We are interested in $\lim_{n\to\infty} L(n,m,u)/n$, i. e., the asymptotically minimum number of bits per key, for $m = (1+\varepsilon) \cdot n$ and $u = n^l$, with $\varepsilon \geqslant 0$ and $l > 1$ for constant ε and l.

For this, first consider the lower bound $L_0(n,u)$ for $u = n^l$.

$$\begin{aligned} \lim_{n\to\infty} \frac{L_0(n,n^l)}{n} &= \frac{1}{\ln 2} \cdot \lim_{n\to\infty} \left(\frac{-n+1}{n} + n^{l-1} \cdot \ln\left(1 - \frac{1}{n^l}\right) - \frac{\ln\left(1 - \frac{1}{n^l}\right)}{n} \right. \\ &\qquad \left. - n^{l-1} \cdot \ln\left(1 - \frac{n}{n^l}\right) + \ln\left(1 - \frac{n}{n^l}\right) \right) \\ &= \frac{1}{\ln 2} \cdot \lim_{n\to\infty} \left(-1 + \underbrace{n^{l-1} \cdot \ln\left(1 - \frac{1}{n^l}\right)}_{=:N_0(l,n)} - \underbrace{n^{l-1} \cdot \ln\left(1 - \frac{n}{n^l}\right)}_{=:N_1(l,n)} \right) \end{aligned}$$

Using that

$$\lim_{n\to\infty} N_0(l,n) = \lim_{n\to\infty} \frac{\frac{d}{dn} \ln\left(1 - \frac{1}{n^l}\right)}{\frac{d}{dn} \frac{1}{n^{l-1}}} = \lim_{n\to\infty} \frac{l}{n^{l+1} \cdot \left(1 - \frac{1}{n^l}\right)} \cdot \frac{n^l}{(-l+1)} = 0 \,,$$

as well as

$$\lim_{n\to\infty} N_1(l,n) = \lim_{n\to\infty} \frac{\frac{d}{dn}\ln\left(1-\frac{1}{n^{l-1}}\right)}{\frac{d}{dn}\frac{1}{n^{l-1}}} = \lim_{n\to\infty} \frac{l-1}{n^l\cdot\left(1-\frac{1}{n^{l-1}}\right)}\cdot\frac{n^l}{(-l+1)} = -1\,,$$

it follows that $\lim_{n\to\infty} L_0(n,n^l)/n = 0$.

Now consider the lower bound $L_1(n,m)$ for $m = n$, i. e., $\varepsilon = 0$.

$$\lim_{n\to\infty}\frac{L_1(n,n)}{n} = \frac{1}{\ln 2}\cdot\lim_{n\to\infty}\left(\frac{n-1}{n}+\frac{\ln\left(1-\frac{n-1}{n}\right)}{n}\right) = \frac{1}{\ln 2}\,.$$

Analogously, for $m = (1+\varepsilon)\cdot n$ and $\varepsilon > 0$, we get

$$\begin{aligned}\lim_{n\to\infty}\frac{L_1(n,(1+\varepsilon)\cdot n)}{n} &= \frac{1}{\ln 2}\cdot\lim_{n\to\infty}\left(\frac{n-1}{n}+\frac{(\varepsilon\cdot n+1)\cdot\ln\left(\frac{\varepsilon\cdot n+1}{(1+\varepsilon)\cdot n}\right)}{n}\right)\\ &= \frac{1+\varepsilon\cdot\ln\left(\frac{\varepsilon}{1+\varepsilon}\right)}{\ln 2}\,.\end{aligned}$$

This finishes the proof of the lemma. ■

4.3.2. Hypergraph Models

A random hypergraph model is a *probability space* that describes the random selection of a hypergraph according to a specified probability distribution. In this section, we discuss several such random experiments whose resulting hypergraphs a. a. s. share a large amount of properties. As a consequence, proving that a certain property a. a. s. holds in one model often implies that this property a. a. s. holds in a related model too. We will use these facts in Section 4.4 for determining the 2-core thresholds $\check{c}(\mathbf{d},\boldsymbol{\alpha})$ for non-uniform hypergraphs $H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$.

! Here and in the following section one should be aware that, with respect to hypergraphs, m stands for the number of nodes and n stands for the number of edges, which is exactly opposite to the common usage.

4.3.2.1. Probability Spaces

Before we take a look at the hypergraph models, we summarize all relevant variables and constants for reference.

Variables:

$$
\begin{aligned}
m &\in \mathbb{N} && \text{number of nodes}\\
n &\in \mathbb{N} && \text{number of hyperedges}\\
n_i &\in \mathbb{R} && \text{(expected) number of hyperedges of size } d_i,\\
& && \sum_{i\in[s]} n_i = n, \forall i \in [s]\colon n_i = \alpha_i \cdot n, \mathbf{n} = (n_0, n_1, \dots, n_{s-1}) \in \mathbb{R}^s\\
p_i &\in \mathbb{R} && \text{probability that edge of size } d_i \text{ exists, } \forall i \in [s]\colon p_i \in [0,1],\\
& && \forall i \in [s]\colon p_i = \frac{\alpha_i \cdot c \cdot m}{\binom{m}{d_i}} = \frac{n_i}{\binom{m}{d_i}}, \mathbf{p} = (p_0, p_1, \dots, p_{s-1}) \in [0,1]^s
\end{aligned}
\tag{PAR}
$$

Constants with respect to m, n, and $\mathbf{n}$:

$$
\begin{aligned}
c &\in \mathbb{R} && \text{load factor, } c = n/m\\
s &\in \mathbb{N} && \text{number of different edge sizes, } s \geqslant 1\\
d_i &\in \mathbb{N} && \text{edge size number } i, \forall i \in [s]\colon d_i \geqslant 3, \mathbf{d} = (d_0, d_1, \dots, d_{s-1}) \in \mathbb{N}^s\\
\alpha_i &\in \mathbb{R} && \text{(expected) fraction of nodes of size } d_i, \forall i \in [s]\colon \alpha_i \in [0,1],\\
& && \sum_{i\in[s]} \alpha_i = 1, \boldsymbol{\alpha} = (\alpha_0, \alpha_1, \dots, \alpha_{s-1}) \in [0,1]^s\\
\Lambda_i &\in \mathbb{R} && \text{(approximate) expected node degree with respect to edges of size } d_i,\\
& && \forall i \in [s]\colon \Lambda_i = p_i \cdot \tbinom{m-1}{d_i-1} = \alpha_i \cdot c \cdot d_i, \boldsymbol{\Lambda} = (\Lambda_0, \Lambda_1, \dots, \Lambda_{s-1}) \in \mathbb{R}^s
\end{aligned}
$$

If not stated otherwise, the parameters for all hypergraphs in this chapter are chosen according to these restrictions.

The *sample space* is defined to be the set of all hypergraphs with m nodes and at most $\binom{m}{d_i}$ edges of size d_i for all $i \in [s]$, where the edges are not necessarily pairwise distinct. We describe the hypergraph models or probability spaces, respectively, via their corresponding random experiments of choosing a hypergraph $H = (V, E)$. All of these experiments start with fixed given node set V of size m and an empty edge (multi-)set E and then successively add up edges to E according to certain random choices.

Type B Hypergraph Model We start with the random hypergraph $H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ that corresponds to the basic scheme type B, as used by the irregular Bloomier filters, see Sections 4.1.2 and 4.1.4. In order to determine a realization of this random variable, n edges are chosen one after the other as follows. First the edge size is set to a random element d from vector $\mathbf{d}$, where for all $i \in [s]$ the probability that $d = d_i$ is α_i. Then

an edge is chosen uniformly at random from the set of all possible edges of size d. Pseudocode is given as Algorithm 8.

ALGORITHM 8: `type_B_hypergraph`

Input : Node set V with $|V| = m$, number of edges n, edge sizes $\mathbf{d}$, edge probabilities $\boldsymbol{\alpha}$.
Output : Realization $H = (V, E)$ of random hypergraph $H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$.

$E_0 \leftarrow E_1 \leftarrow \ldots \leftarrow E_{s-1} \leftarrow \emptyset$; `// edge multisets`
for $j \leftarrow 0$ **to** $n-1$ **do**
  $i \leftarrow$ realization of Y_j with $\forall y \in [s]\colon \Pr(Y_j = y) = \alpha_y$;
  $e \leftarrow$ realization of Z_j with $\forall z \in \binom{V}{d_i}\colon \Pr(Z_j = z) = \binom{V}{d_i}^{-1}$;
  $E_i \leftarrow E_i \cup \{e\}$;
end
return $(V, \bigcup_{i\in[s]} E_i)$; `// expected size of` E_i `is` $\alpha_i \cdot n$

FIXED FRACTIONS MULTIPLE EDGES HYPERGRAPH MODEL In a common variation of the previous model the edge sizes are not chosen randomly but rather for each $i \in [s]$ the number of edges of size d_i is set to $\alpha_i \cdot n$, i. e., is given in advance. We refer to this hypergraph model[10] as ${}^{\text{mul}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$. Algorithm 9 gives the pseudocode for the corresponding random experiment.

ALGORITHM 9: `fixed_fractions_multiple_edges_hypergraph`

Input : Node set V with $|V| = m$, number of edges n, edge sizes $\mathbf{d}$, distribution of edge sizes $\boldsymbol{\alpha}$.
Output : Realization $H = (V, E)$ of random hypergraph ${}^{\text{mul}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$.

$E_0 \leftarrow E_1 \leftarrow \ldots \leftarrow E_{s-1} \leftarrow \emptyset$; `// edge multisets`
for $i \leftarrow 0$ **to** $s-1$ **do**
  for $j \leftarrow 0$ **to** $\alpha_i \cdot n - 1$ **do**
    $e \leftarrow$ realization of $Z_{i,j}$ with $\forall z \in \binom{V}{d_i}\colon \Pr(Z_{i,j} = z) = \binom{V}{d_i}^{-1}$;
    $E_i \leftarrow E_i \cup \{e\}$;
  end
end
return $(V, \bigcup_{i\in[s]} E_i)$; `// size of` E_i `is fixed to` $\alpha_i \cdot n$

[10] For $s = \alpha_0 = 1$ and $d_0 = 2$, i. e., normal graphs, this model corresponds to $\Gamma^*_{m,n}$ in notation of Erdős and Rényi [ER60, page 20], and was (probably) first studied in [AFPR59].

FIXED FRACTIONS SIMPLE EDGES HYPERGRAPH MODEL Prohibiting multiple edges straightforwardly leads from ${}^{\text{mul}}\mathrm{H}^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ to a model of simple hypergraphs which we refer[11] as ${}^{\text{sim}}\mathrm{H}^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$. Algorithm 10 gives the pseudocode for choosing such a hypergraph. For ease of subsequent discussions, we refer to the *support* of ${}^{\text{sim}}\mathrm{H}^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$, i. e., the set of all simple hypergraphs with m nodes and $\alpha_i \cdot n$ edges of size d_i, as ${}^{\text{sim}}\mathcal{H}^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$.

ALGORITHM 10: `fixed_fractions_simple_edges_hypergraph`

Input : Node set V with $|V| = m$, number of edges n, edge sizes $\mathbf{d}$, distribution of edge sizes $\boldsymbol{\alpha}$.
Output : Realization $H = (V, E)$ of random hypergraph ${}^{\text{sim}}\mathrm{H}^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$.

$E_0 \leftarrow E_1 \leftarrow \ldots \leftarrow E_{s-1} \leftarrow \emptyset$; `// edge sets`
for $i \leftarrow 0$ to $s-1$ do
  for $j \leftarrow 0$ to $\alpha_i \cdot n - 1$ do
    $e \leftarrow$ realization of $Z_{i,j}$ with $\forall z \in \binom{V}{d_i} - E_i$: $\Pr(Z_{i,j} = z) = \left(\binom{V}{d_i} - j\right)^{-1}$;
    $E_i \leftarrow E_i \cup \{e\}$;
  end
end
return $(V, \bigcup_{i \in [s]} E_i)$; `// size of` E_i `is fixed to` $\alpha_i \cdot n$

BINOMIAL HYPERGRAPH MODEL The next model[12], which we call binomial hypergraph model ${}^{\text{bin}}\mathrm{H}^{\mathbf{d}}_{m,\mathbf{p}}$, is conceptual slightly different from the previous ones. Now the total number of edges for a realization is not known in advance, instead, for all $i \in [s]$ each possible edge of size d_i from $\binom{V}{d_i}$ is present with probability p_i independent of all other edges. The edge selection procedure is shown in Algorithm 11.

POISSON CLONING HYPERGRAPH MODEL The last model that we consider is, unlike the previous ones, relatively new. It was introduced by Kim around 2004 and is called Poisson cloning model [Kim06, Kim08], or in our nomenclature Poisson cloning hypergraph model ${}^{\text{poi}}\mathrm{H}^{\mathbf{d}}_{m,\boldsymbol{\Lambda}}$. The selection of a hypergraph according to this model can be described as follows. Given V with $|V| = m$, for each $i \in [s]$ create a new set C_i of $\mathrm{Po}[m \cdot \Lambda_i]$ nodes, called clones, as well as an empty edge multiset E_i for edges of size d_i. If the number of clones $|C_i|$ is not divisible by d_i, then with respect to *our* applications the resulting hypergraph is not useable; for convenience let $E_j = \emptyset$ for all $j \in [s]$ in this case. Otherwise, choose an arbitrary but fixed partition of C_i, where each part has size d_i. Now, label each clone of C_i uniformly at random with a node from V and build a new edge for each part of the partition using the nodes that correspond to the labels

[11] This is a slight generalization of the classical graph model $\Gamma_{m,n}$ by Erdős and Rényi [ER60].
[12] This is a slight generalization of the classical graph model by Gilbert [Gil59], denoted $\Gamma^{**}_{m,n}$ by Erdős and Rényi [ER60, page 20].

ALGORITHM 11: `binomial_hypergraph`

Input : Node set V with $|V| = m$, edge sizes $\mathbf{d}$, edge probabilities $\mathbf{p}$.
Output : Realization $H = (V, E)$ of random hypergraph ${}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}}$.

```
E_0 ← E_1 ← ... ← E_{s-1} ← ∅;                                  // edge sets
for i ← 0 to s − 1 do
    foreach e ∈ (V choose d_i) do
        x ← realization of X_e with Pr(X_e = 1) = p_i;
        if x = 1 then E_i ← E_i ∪ {e};
    end
end
return (V, ⋃_{i∈[s]} E_i);          // size of E_i is Bin(|(V choose d_i)|, p_i)-distributed
```

of this part. If the labels for such an edge e are pairwise distinct, i. e., $|e| = d_i$, then increase E_i by e. Otherwise, for simplicity of further considerations, we choose not to use the resulting hypergraph, i. e., we define $E_j = \emptyset$ for all $j \in [s]$, although in principle e could have a valid size d_j for some $j \in [s]$. Pseudocode is given as Algorithm 12.

REMARK. If we don't care if $|C_i|$ is divisible by d_i or not, and consider edges as multisets, such that their nodes are allowed to have multiplicities larger than 1, then the degree D_x of an arbitrary but fixed node x with respect to edges of size d_i has Poisson distribution with parameter Λ_i, independent of the other nodes; here degree is understood as the sum over all occurrences of x in all edges of size d_i, i. e., the number of clones from C_i labeled with x. This can be shown as follows:

$$
\begin{aligned}
\Pr(D_x = d) &= \sum_{k \geqslant d} \Pr(D_x = d \mid \mathrm{Po}[\Lambda_i \cdot m] = k) \cdot \Pr(\mathrm{Po}[\Lambda_i \cdot m] = k) \\
&= \sum_{k \geqslant d} \Pr(\mathrm{Bin}[k, 1/m] = d) \cdot \Pr(\mathrm{Po}[\Lambda_i \cdot m] = k) \\
&= \sum_{k \geqslant d} \binom{k}{d} \cdot \left(\frac{1}{m}\right)^d \cdot \left(1 - \frac{1}{m}\right)^{k-d} \cdot \frac{(\Lambda_i \cdot m)^k}{k!} \cdot e^{-\Lambda_i \cdot m} \\
&= \sum_{k \geqslant 0} \frac{\Lambda_i^{k+d} \cdot e^{-\Lambda_i \cdot m} \cdot (m-1)^k}{d! \cdot k!} \cdot e^{-\Lambda_i} \cdot e^{\Lambda_i} \\
&= \frac{\Lambda_i^d}{d!} \cdot e^{-\Lambda_i} \cdot \sum_{k \geqslant 0} \frac{\big(\Lambda_i \cdot (m-1)\big)^k}{k!} \cdot e^{-\Lambda_i \cdot (m-1)} \\
&= \Pr(\mathrm{Po}[\Lambda_i] = d)\,.
\end{aligned}
$$

ALGORITHM 12: poisson_cloning_hypergraph

Input : Node set V with $|V| = m$, edge sizes $\mathbf{d}$, corresponding expected node degrees $\boldsymbol{\Lambda}$.
Output : Realization $H = (V, E)$ of random hypergraph ${}^{\text{poi}}H^{\mathbf{d}}_{m,\boldsymbol{\Lambda}}$.

```
E_0 ← E_1 ← ... ← E_{s-1} ← ∅;                                  // edge multisets
for i ← 0 to s − 1 do
    m_cl ← realization of Po[Λ_i · m];                          // number of clones
    if ∃n_i ∈ ℕ: m_cl = d_i · n_i then
        for j ← 0 to m_cl/d_i − 1 do
            for k ← 0 to d_i − 1 do                                  // node labels
                l_k ← realization of Z_{i,j,k} with ∀z ∈ V: Pr(Z_{i,j,k} = z) = m^{-1};
            end
            e ← {l_0, l_1, ..., l_{d_i−1}};
            if |e| < d_i then return (V, ∅);
            else E_i ← E_i ∪ {e};
        end
    else return (V, ∅);
end
return (V, ⋃_{i∈[s]} E_i);        // size of E_i is controlled by Poisson distr.
```

4.3.2.2. Asymptotic Equivalence with Respect to Certain Properties

A property $\mathcal{P}$ of hypergraphs is simply a set of hypergraphs, and a hypergraph H has property $\mathcal{P}$ if and only if $H \in \mathcal{P}$. Consequently, a property of a hypergraph model is a subset of its sample space, i. e., a hypergraph model has a property $\mathcal{P}$ with a specific probability.

In the following we give some standard proofs that certain properties which are likely for one of our hypergraph models are also likely for the other models too. Hereby, we consider the models in reversed order as discussed above.

> LEMMA 4.3.3 (POISSON CLONING AND BINOMIAL MODEL, DERIVED FROM [KIM06, KIM08, THEOREM 1.1]). Given (PAR). Let $\mathcal{P} \subseteq {}^{\text{sim}}\mathcal{H}^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ be a hypergraph property, then the following holds:
>
> (i) If $\Pr({}^{\text{poi}}H^{\mathbf{d}}_{m,\boldsymbol{\Lambda}} \in \mathcal{P}) = o(1)$, then $\Pr({}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}} \in \mathcal{P}) = o(1)$.
>
> (ii) If $\Pr({}^{\text{poi}}H^{\mathbf{d}}_{m,\boldsymbol{\Lambda}} \in \mathcal{P}) = 1 - o(1)$, then $\Pr({}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}} \in \mathcal{P}) = 1 - o(1)$.

PROOF. The proof follows [Kim08, proof of Theorem 1.1], with the difference that instead of considering d-uniform hypergraphs, i. e., one edge size class, we consider non-uniform (mixed) hypergraphs with s edge size classes. We will show that there exist non-zero constants $l = l(\mathbf{d})$ and $k = k(\mathbf{d})$, such that for an arbitrary but fixed

hypergraph $H \in \mathcal{P}$, with $\mathcal{P} \subseteq {}^{\text{sim}}\mathcal{H}^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$, we have

$$l \cdot \Pr({}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}} = H) \leqslant \Pr({}^{\text{poi}}H^{\mathbf{d}}_{m,\boldsymbol{\Lambda}} = H) \leqslant k \cdot \Pr({}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}} = H)\,.$$

Then by

$$\Pr({}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}} \in \mathcal{P}) = \sum_{H \in \mathcal{P}} \Pr({}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}} = H) \text{ and}$$
$$\Pr({}^{\text{poi}}H^{\mathbf{d}}_{m,\boldsymbol{\Lambda}} \in \mathcal{P}) = \sum_{H \in \mathcal{P}} \Pr({}^{\text{poi}}H^{\mathbf{d}}_{m,\boldsymbol{\Lambda}} = H)\,,$$

we get

$$l \cdot \Pr({}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}} \in \mathcal{P}) \leqslant \Pr({}^{\text{poi}}H^{\mathbf{d}}_{m,\boldsymbol{\Lambda}} \in \mathcal{P}) \leqslant k \cdot \Pr({}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}} \in \mathcal{P})\,,$$

which implies the claim.

So, let $H \in \mathcal{P}$ be arbitrary but fixed. Furthermore, let $n_i = \alpha_i \cdot n$ and observe that $\Lambda_i \cdot m = d_i \cdot n_i$. According to Section 4.3.2 and Algorithms 11 and 12, we have

$$\Pr({}^{\text{poi}}H^{\mathbf{d}}_{m,\boldsymbol{\Lambda}} = H) = \prod_{i \in [s]} \underbrace{\Pr(\text{Po}[d_i \cdot n_i] = d_i \cdot n_i) \cdot n_i! \cdot \left(\frac{d_i!}{m^{d_i}}\right)^{n_i}}_{=:F_i}$$
$$\Pr({}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}} = H) = \prod_{i \in [s]} \underbrace{p_i^{n_i} \cdot (1-p_i)^{\binom{m}{d_i} - n_i}}_{=:G_i}\,.$$

Now fix some $i \in [s]$ and let $F = F_i$ and $G = G_i$, as well as $p_i = p$, $n_i = n$, and $d = d_i$. We will now derive bounds for F and G. According to the definition of F, we have

$$F = \frac{(d \cdot n)^{d \cdot n}}{(d \cdot n)!} \cdot e^{-d \cdot n} \cdot n! \cdot \left(\frac{d!}{m^d}\right)^n\,.$$

By Stirling's formula, we get

$$\begin{aligned} F &= \frac{\big(1 + O(1/n)\big) \cdot \sqrt{2\pi \cdot n} \cdot \left(\frac{n}{e}\right)^n}{\big(1 + O(1/(d \cdot n))\big) \cdot \sqrt{2\pi \cdot d \cdot n} \cdot \left(\frac{d \cdot n}{e}\right)^{d \cdot n}} \cdot (d \cdot n)^{d \cdot n} \cdot e^{-d \cdot n} \cdot \left(\frac{d!}{m^d}\right)^n \\ &= \underbrace{\big(d^{-1/2} \pm o(1)\big)}_{=:f} \cdot \left(\frac{e}{n}\right)^{(d-1) \cdot n} n^{d \cdot n} \cdot e^{-d \cdot n} \cdot \left(\frac{d!}{m^d}\right)^n \\ &= f \cdot \frac{n^n \cdot \left(\frac{d!}{m^d}\right)^n}{e^n} = f \cdot \frac{p^n \cdot \binom{m}{d}^n \cdot \left(\frac{d!}{m^d}\right)^n}{e^n}\,. \end{aligned}$$

Using that $\left(\frac{m}{d}\right)^d \leqslant \binom{m}{d} \leqslant \frac{m^d}{d!}$, we get

$$f \cdot \frac{d!}{d^d} \cdot \frac{p^n}{e^n} \leqslant F \leqslant f \cdot \frac{p^n}{e^n}\,.$$

According to the definition of G, we have

$$G = p^n \cdot (1-p)^{\binom{m}{d}-n} = p^n \cdot e^{\ln(1-p)\cdot\left(\binom{m}{d}-n\right)}.$$

Using that

$$\begin{aligned}\ln(1-p)\cdot\left(\binom{m}{d}-n\right) &= -\sum_{j=1}^{\infty}\frac{p}{j}\cdot\left(\binom{m}{d}-n\right)\\ &= -p\binom{m}{d} + \sum_{j=1}^{\infty}\Bigg(\underbrace{\frac{p^j\cdot n}{j} - \frac{p^{j+1}}{j+1}\cdot\binom{m}{d}}_{=:g}\Bigg),\end{aligned}$$

where $g = \frac{n^{j+1}}{j\cdot\binom{m}{d}^j} - \frac{n^{j+1}}{(j+1)\cdot\binom{m}{d}^j} > 0$, we get the following bounds

$$p^n \cdot e^{-p\cdot\binom{m}{d}+p\cdot n-p^2/2\cdot\binom{m}{d}} \leqslant G \leqslant p^n \cdot e^{-p\cdot\binom{m}{d}+p\cdot n}.$$

This is equivalent to

$$\frac{p^n}{e^{(1-p/2)\cdot n}} \leqslant G \leqslant \frac{p^n}{e^{(1-p)\cdot n}}.$$

Now, since $p\cdot n = O(m^{2-d})$ and $d \geqslant 3$, we have $l < \frac{F_0\cdot F_1\cdot\ldots\cdot F_{s-1}}{G_0\cdot G_1\cdot\ldots\cdot G_{s-1}} < k$, for non-zero constants l and k, with

$$\begin{aligned} l &< \prod_{i\in[s]}\left((d_i^{-1/2}\pm o(1))\cdot\frac{d_i!}{d_i^{d_i}}\cdot\frac{p_i^{n_i}}{e^{n_i}}\right)\Big/\left(\frac{p_i^{n_i}}{e^{(1-p_i)\cdot n_i}}\right)\\ k &> \prod_{i\in[s]}\left((d_i^{-1/2}\pm o(1))\cdot\frac{p_i^{n_i}}{e^{n_i}}\right)\Big/\left(\frac{{p_i}^{n_i}}{e^{(1-p_i/2)\cdot n_i}}\right).\end{aligned}$$

This finishes the proof of the lemma. ■

The next "transfer" is from ${}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}}$ to ${}^{\text{sim}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$. For this purpose we make use of *convex properties*. A set $\mathcal{P}$ of hypergraphs is called convex property, if for each pair of hypergraphs H_A, H_C from $\mathcal{P}$ and each hypergraph H_B it holds that $H_A \subseteq H_B \subseteq H_C$ implies that the "middle" hypergraph H_B is also from $\mathcal{P}$.

> Lemma 4.3.4 (Binomial and Fixed Fractions Simple Edges Model, slight generalization of [JŁR00, Proposition 1.15]). Given (PAR). Let $\mathcal{P}$ be a convex property of all hypergraphs on m nodes whose edge sizes are restricted to $\mathbf{d}$. Then the following holds:
>
> $$\text{If } \Pr({}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}} \in \mathcal{P}) = 1-o(1), \text{ then } \Pr({}^{\text{sim}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}} \in \mathcal{P}) = 1-o(1).$$

PROOF. We follow the proof of [JŁR00, Proposition 1.15]. Let $n_i = \alpha_i \cdot n$ as well as let $\mathbf{n} = (n_i)_{i\in[s]}$. In the following, we will use ${}^{\text{sim}}H^{\mathbf{d}}_{m,\mathbf{n}}$ and ${}^{\text{sim}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ synonymously. Let $\mathbf{X} = (X_i)_{i\in[s]}$ be a vector of random variables, where each component $X_i \in \binom{m}{d_i}$ counts the number of edges of size d_i of ${}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}}$. Moreover, for any relation R from $\{=, \neq, <, >, \leqslant, \geqslant\}$ and two vectors $\mathbf{a}$ and $\mathbf{b}$ of the same dimension s, we define $\mathbf{a}\ R\ \mathbf{b} \Leftrightarrow \forall i \in [s]\colon a_i\ R\ b_i$. By the law of total probability we have

$$\begin{aligned}\Pr({}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}} \in \mathcal{P}) &= \sum_{\mathbf{x}} \Pr({}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}} \in \mathcal{P} \mid \mathbf{X} = \mathbf{x}) \cdot \Pr(\mathbf{X} = \mathbf{x}) \\ &= \sum_{\mathbf{x}} \Pr({}^{\text{sim}}H^{\mathbf{d}}_{m,\mathbf{x}} \in \mathcal{P}) \cdot \Pr(\mathbf{X} = \mathbf{x})\,.\end{aligned}$$

Now define

$$\begin{aligned}\mathbf{x}^- &\coloneqq \arg\max_{\mathbf{x}} \left\{\Pr({}^{\text{sim}}H^{\mathbf{d}}_{m,\mathbf{x}} \in \mathcal{P}) \mid \mathbf{x} \leqslant \mathbf{n}\right\}, \text{ and} \\ \mathbf{x}^+ &\coloneqq \arg\max_{\mathbf{x}} \left\{\Pr({}^{\text{sim}}H^{\mathbf{d}}_{m,\mathbf{x}} \in \mathcal{P}) \mid \mathbf{x} > \mathbf{n}\right\}\,.\end{aligned}$$

Applying these definitions gives the following two upper bounds

$$\Pr({}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}} \in \mathcal{P}) \leqslant \Pr({}^{\text{sim}}H^{\mathbf{d}}_{m,\mathbf{x}^-} \in \mathcal{P}) \cdot \Pr(\mathbf{X} \leqslant \mathbf{n}) + 1 \cdot \big(1 - \Pr(\mathbf{X} \leqslant \mathbf{n})\big) \qquad (\triangle)$$

$$\Pr({}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}} \in \mathcal{P}) \leqslant \Pr({}^{\text{sim}}H^{\mathbf{d}}_{m,\mathbf{x}^+} \in \mathcal{P}) \cdot \Pr(\mathbf{X} > \mathbf{n}) + 1 \cdot \big(1 - \Pr(\mathbf{X} > \mathbf{n})\big)\,. \qquad (\blacktriangle)$$

The vector $\mathbf{X}$ is a random vector where each component X_i is the sum of indicator variables Y_j, $j \in [\binom{m}{d_i}]$, which indicate if edge number j of size d_i is present. Using that the expected value for X_i is $\text{Exp}(X_i) = p_i \cdot \binom{m}{d_i} = n_i$, we get

$$\Pr(X_i \leqslant n_i) = \Pr(X_i - \text{Exp}(X_i) \leqslant 0) = \Pr\Big(\underbrace{\frac{X_i - \text{Exp}(X_i)}{\sqrt{\text{Var}(X_i)}}}_{=:Z_i} \leqslant 0\Big)\,.$$

The random variables Z_i are independent, because the X_i's are independent. Moreover, by the central limit theorem, they follow a standard normal distribution as m goes to ∞. Therefore, we have

$$\lim_{m\to\infty} \Pr(\mathbf{X} \leqslant \mathbf{n}) = \lim_{m\to\infty} \Pr(\mathbf{X} > \mathbf{n}) = 1/2^s\,.$$

Using the assumption $\Pr({}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}} \in \mathcal{P}) = 1 - o(1)$ along with the fact that $1/2^s$ is constant, it follows from $(\triangle)$ that $\Pr({}^{\text{sim}}H^{\mathbf{d}}_{m,\mathbf{x}^-} \in \mathcal{P}) = 1 - o(1)$, and it follows from $(\blacktriangle)$ that $\Pr({}^{\text{sim}}H^{\mathbf{d}}_{m,\mathbf{x}^+} \in \mathcal{P}) = 1 - o(1)$.

Analogously to ${}^{\text{sim}}\mathcal{H}^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$, let ${}^{\text{sim}}\mathcal{H}^{\mathbf{d}}_{m,\mathbf{x}^-,\boldsymbol{\alpha}}$ be the support of ${}^{\text{sim}}H^{\mathbf{d}}_{m,\mathbf{x}^-}$ and let ${}^{\text{sim}}\mathcal{H}^{\mathbf{d}}_{m,\mathbf{x}^+,\boldsymbol{\alpha}}$ be the support of ${}^{\text{sim}}H^{\mathbf{d}}_{m,\mathbf{x}^+}$. Furthermore, let

$$\begin{aligned}\mathcal{P}_A &\coloneqq {}^{\text{sim}}\mathcal{H}^{\mathbf{d}}_{m,\mathbf{x}^-,\boldsymbol{\alpha}} \cap\ \mathcal{P} \\ \mathcal{P}_C &\coloneqq {}^{\text{sim}}\mathcal{H}^{\mathbf{d}}_{m,\mathbf{x}^+,\boldsymbol{\alpha}} \cap\ \mathcal{P} \\ \mathcal{P}_B &\coloneqq \left\{H_B \in {}^{\text{sim}}\mathcal{H}^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}} \cap\ \mathcal{P} \mid \exists H_A \in \mathcal{P}_A\ \exists H_C \in \mathcal{P}_C\colon H_A \subseteq H_B \subseteq H_C\right\}.\end{aligned}$$

It holds that

$$\Pr({}^{\text{sim}}H^{\mathbf{d}}_{m,x^-} \in \mathcal{P}) = \frac{|\mathcal{P}_A|}{|{}^{\text{sim}}\mathcal{H}^{\mathbf{d}}_{m,x^-,\boldsymbol{\alpha}}|}$$
$$\Pr({}^{\text{sim}}H^{\mathbf{d}}_{m,x^+} \in \mathcal{P}) = \frac{|\mathcal{P}_C|}{|{}^{\text{sim}}\mathcal{H}^{\mathbf{d}}_{m,x^+,\boldsymbol{\alpha}}|}$$
$$\Pr({}^{\text{sim}}H^{\mathbf{d}}_{m,\mathbf{n}} \in \mathcal{P}) \geqslant \frac{|\mathcal{P}_B|}{|{}^{\text{sim}}\mathcal{H}^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}|}\,.$$

Since, we know that $\Pr({}^{\text{sim}}H^{\mathbf{d}}_{m,x^-} \in \mathcal{P})$ and $\Pr({}^{\text{sim}}H^{\mathbf{d}}_{m,x^+} \in \mathcal{P})$ are bounded by $1-o(1)$, it follows that $|\mathcal{P}_A| \to |{}^{\text{sim}}\mathcal{H}^{\mathbf{d}}_{m,x^-,\boldsymbol{\alpha}}|$ and $|\mathcal{P}_C| \to |{}^{\text{sim}}\mathcal{H}^{\mathbf{d}}_{m,x^+,\boldsymbol{\alpha}}|$ for $m \to \infty$. Since $\mathcal{P}$ is convex, we have that $|\mathcal{P}_B| \to |{}^{\text{sim}}\mathcal{H}^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}|$. Hence, we have

$$\Pr({}^{\text{sim}}H^{\mathbf{d}}_{m,\mathbf{n}} \in \mathcal{P}) = \Pr({}^{\text{sim}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}} \in \mathcal{P}) = 1 - o(1)\,,$$

which finishes the proof of the lemma. ■

Next, we show the asymptotic equivalence of the probability that ${}^{\text{sim}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ has a given property and the probability that ${}^{\text{mul}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ has the same property.

> Lemma 4.3.5 (Fixed Fractions Simple Edges and Multiple Edges Model, analogous to [FP10, Proposition 1]). Given (PAR). Let $\mathcal{P} \subseteq {}^{\text{sim}}\mathcal{H}^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ be a hypergraph property, then the following holds:
>
> $$\Pr({}^{\text{sim}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}} \in \mathcal{P}) = \Pr({}^{\text{mul}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}} \in \mathcal{P}) + o(1)\,.$$

Proof. The following is standard. Let $\mathcal{A}$ be the event that ${}^{\text{mul}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ has duplicate edges. We get

$$\begin{aligned}\Pr({}^{\text{mul}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}} \in \mathcal{P}) &= \Pr({}^{\text{mul}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}} \in \mathcal{P} \mid \overline{\mathcal{A}}) \cdot \Pr(\overline{\mathcal{A}}) \\ &= \Pr({}^{\text{sim}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}} \in \mathcal{P}) \cdot \big(1 - \Pr(\mathcal{A})\big)\,.\end{aligned}$$

For each $i \in [s]$ let E_i be the set of edges of size d_i of ${}^{\text{mul}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$. Furthermore, for each $\{e_i, e_i'\} \in \binom{E_i}{2}$ let $X_{\{e_i,e_i'\}}$ be a binary random variable with $X_{\{e_i,e_i'\}} = 1$ if $e_i = e_i'$ and $X_{\{e_i,e_i'\}} = 0$ otherwise. Now we have

$$\Pr(\mathcal{A}) = \Pr\left(\sum_{i\in[s]}\sum_{\{e_i,e_i'\}\in E_i} X_{\{e_i,e_i'\}} \geqslant 1\right) \leqslant \sum_{i\in[s]} \operatorname{Exp}\left(\sum_{\{e_i,e_i'\}\in E_i} X_{\{e_i,e_i'\}}\right).$$

For arbitrary but fixed i, it holds that $\operatorname{Exp}(\sum_{\{e_i,e_i'\}\in E_i} X_{\{e_i,e_i'\}}) = \binom{\alpha_i\cdot n}{2}/\binom{m}{d_i}$, which is bounded by $o(1)$ for $d_i \geqslant 3$. Since s is a constant, we have $\Pr(\mathcal{A}) = o(1)$, and therefore $\Pr({}^{\text{mul}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}} \in \mathcal{P}) = \Pr({}^{\text{sim}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}} \in \mathcal{P}) \cdot \big(1 - o(1)\big)$, which finishes the proof of the lemma. ■

The last lemma shows the close connection between ${}^{\text{mul}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ and $H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ (type B), cf., Section 3.5.5.1.

LEMMA 4.3.6 (FIXED FRACTIONS MULTIPLE EDGES AND TYPE B MODEL). Given (PAR). Let $\mathcal{P}$ be a convex property of all hypergraphs on m nodes whose edge sizes are restricted to $\mathbf{d}$. Furthermore, let $\delta > 1/2$ be an arbitrary constant. Then the following holds:

$$\text{If } \Pr({}^{\text{mul}}H^{\mathbf{d}}_{m,n-n^{\delta},\boldsymbol{\alpha}} \in \mathcal{P}) = 1 - o(1) \text{ and } \Pr({}^{\text{mul}}H^{\mathbf{d}}_{m,n+n^{\delta},\boldsymbol{\alpha}} \in \mathcal{P}) = 1 - o(1),$$
$$\text{then } \Pr(H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}} \in \mathcal{P}) = 1 - o(1)\,.$$

PROOF. For all $i \in [s]$ let X_i be a random variable that counts the number of edges of size d_i of $H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$; recall that $\text{Exp}(X_i) = \alpha_i \cdot n$. Arguing analogously to Section 3.5.5.1, as part of the proof of Proposition 3.1, we conclude that a. a. s. for all $i \in [s]$ it holds that $\alpha_i \cdot (n - n^{\delta}) \leqslant X_i \leqslant \alpha_i \cdot (n + n^{\delta})$. Since a realization of $H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}} \mid (X_i)_{i\in[s]} = (x_i)_{i\in[s]}$ is chosen fully random among all hypergraphs with m nodes and x_i edges of size d_i, the lemma follows by the convexity of $\mathcal{P}$. ■

4.4. Thresholds for the Appearance of Cores in Non-Uniform Hypergraphs

In this section we consider Theorem 4.4 in the subsequent, more general version concerning ℓ_+-cores instead of 2-cores, see Section 2.3.1. Let

$$\check{c}_{\ell_+}(\mathbf{d}, \boldsymbol{\alpha}) \coloneqq \min_{\lambda>0} \text{key}(\lambda, \mathbf{d}, \boldsymbol{\alpha}, \ell_+ - 1)\,,$$

for the *key function*

$$\text{key}(\lambda, \mathbf{d}, \boldsymbol{\alpha}, \ell) = \frac{\lambda}{\sum\limits_{i\in[s]} \alpha_i \cdot d_i \cdot \left(\Pr\left(\text{Po}\,[\lambda] \geqslant \ell\right)\right)^{d_i - 1}}\,,$$

with number of orientations $\ell = \ell_+ - 1$ and special case $\check{c}_2(\mathbf{d}, \boldsymbol{\alpha}) = \check{c}(\mathbf{d}, \boldsymbol{\alpha})$, see Sections 3.1.1.5 and 3.4.

THEOREM 4.6 (ℓ_+-CORE APPEARANCE, GENERALIZATION OF THEOREM 4.4) *Given* (PAR). *Let $c \leqslant \ell_+$ for constant $\ell_+ \in \mathbb{N}$ with $\ell_+ \geqslant 2$. Then for any constant $\varepsilon > 0$ a. a. s. the following holds:*

(i) If $c \leqslant \check{c}_{\ell_+}(\mathbf{d}, \boldsymbol{\alpha}) - \varepsilon$, then $H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ (type B) has an empty ℓ_+-core.

(ii) If $c \geqslant \check{c}_{\ell_+}(\mathbf{d}, \boldsymbol{\alpha}) + \varepsilon$, then $H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ (type B) has a non-empty ℓ_+-core.

Since an outline of a proof can be easily derived from [DGM+10, Section 4], Theorem 4.6 is not a fully adequate result of this thesis, and, accordingly, is not classified as such.

However, on the basis of [DGM+09, DGM+10] we elaborate the details missing for a full proof of the first part of the theorem in Section 4.4.1, i. e., showing that $\check{c}_{\ell_+}(\mathbf{d}, \boldsymbol{\alpha})$ is a lower bound of the ℓ_+-core threshold, as well as sketch the proof of the second part in Section 4.4.2, i. e., argue that $\check{c}_{\ell_+}(\mathbf{d}, \boldsymbol{\alpha})$ is also an upper bound of the ℓ_+-core threshold.

4.4.1. Lower Bound

The proof of the lower bound for the ℓ_+-core threshold is *almost identical* to the first part of the proof of [Mol04, Theorem 1.2] by Molloy, using exactly the same strategy and only slightly generalized lemmas regarding non-uniform or mixed hypergraphs, respectively.

Our starting point is a variant of Algorithm 1 that does peeling steps in parallel, formalized as Algorithm 13. In order to reference intermediate results, the rounds of the algorithm and the corresponding hypergraphs are numbered consecutively starting with 0.

Algorithm 13: `peeling_in_parallel`

```
Input  : Hypergraph H = (V, E), core parameter ℓ₊ ∈ ℕ with ℓ₊ ⩾ 1.
Output : The ℓ₊-core of H.
t ← 0; V_t = V; E_t = E;
while true do
    H_t ← (V_t, E_t);
    V' ← {v ∈ V_t | |{e ∈ E_t | v ∈ e}| ⩽ ℓ₊ − 1};
    if V' = ∅ then break;
    t ← t + 1; V_t ← V_{t−1} − V'; E_t ← E_{t−1} − {e ∈ E_{t−1} | e ∩ V' ≠ ∅};
end
return H_t;
```

If we apply `peeling_in_parallel` on a random hypergraph ${}^{\mathrm{mul}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$, then a. a. s. after a constant number of rounds we are left with a hypergraph that has a relatively small number of nodes.

> **Lemma 4.4.1** (Relatively Small Remaining Subgraph, slight generalization of [Mol04, Lemma 2.1]). Given (PAR) and let $\varepsilon > 0$ be arbitrary but fixed. If $c \leqslant \check{c}_{\ell_+}(\mathbf{d}, \boldsymbol{\alpha}) - \varepsilon$, then for every $\delta > 0$ there is a constant $t = t(\delta) \in \mathbb{N}$, such that a. a. s. after t many steps of Algorithm 13 on input ${}^{\mathrm{bin}}H^{\mathbf{d}}_{m,\mathbf{p}}$, the intermediate result H_t has less than $\delta \cdot m$ nodes.

Lemma 4.4.1 is proved in Section 4.4.1.1. Applying Lemmas 4.3.4 and 4.3.5, we conclude that this lemma also holds for ${}^{\mathrm{mul}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$.

Next, we consider the *average degree* of each subgraph of ${}^{\mathrm{mul}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ with less than $\delta \cdot m$ nodes, where the average degree of a hypergraph $H = (V, E)$ with node set V and edge set E is defined as $\sum_{e \in E} |e|/|V|$.

> Lemma 4.4.2 (Small Average Degree, slight generalization of [Mol04, Lemma 2.3]). Given (PAR). For any $c \leqslant \ell_+$ there is a constant $\delta = \delta(c)$ such that a. a. s. the graph ${}^{\mathrm{mul}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ has no subgraph with less than $\delta \cdot m$ nodes and average degree at least ℓ_+.

The proof of this lemma is given in Section 4.4.1.2.

By Lemmas 4.4.1 and 4.4.2, it follows that if we apply Algorithm 13 on ${}^{\mathrm{mul}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ with $c = n/m \leqslant \check{c}_{\ell_+}(\mathbf{d}, \boldsymbol{\alpha}) - \varepsilon$, then after $t(\delta(c))$ many steps a. a. s. either the output is empty, or the output is nonempty and all subgraphs of the resulting hypergraph have minimum node degree below ℓ_+. Hence, a. a. s. ${}^{\mathrm{mul}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ has no ℓ_+-core. This also holds if the number of edges of ${}^{\mathrm{mul}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ are increased or reduced by summands of size sublinear in n_i, e. g., for ${}^{\mathrm{mul}}H^{\mathbf{d}}_{m,n\pm n^{4/7},\boldsymbol{\alpha}}$. Hence, by Lemma 4.3.6, the lower bound holds for $H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ type B.

So, to complete the proof of the lower bound of the ℓ_+-core threshold, it remains to show the two lemmas above, which is done in the following two sections.

4.4.1.1. Obtaining a Relatively Small Subgraph After a Constant Number of Parallel Peeling Steps

In this Section we show Lemma 4.4.1, by adapting the proof of [Mol04, Lemma 2.1].

We start with some definitions to clarify the concepts of *path*, *simple path* and *simple tree* that we will use in the following, since, regarding hypergraphs, they are not necessarily unambiguous.

> Definition 5: (Path and Distance)
> A *path* $P = (v_0, e_1, v_1, e_2, \ldots, v_{l-1}, e_l, v_l)$ of length $l \in \mathbb{N}$ between two nodes v and v' of a hypergraph H is an alternating sequence of nodes and edges with the following properties:
>
> - ▷ The path begins with v and ends with v', i. e., $v = v_0$ and $v' = v_l$.
> - ▷ The edges of the path are pairwise distinct, i. e., $e_i \neq e_j$ for all $1 \leqslant i < j \leqslant l$.
> - ▷ Each node is element of the edge to the left and element of the edge to the right in the sequence P, i. e., $\{v_{i-1}, v_i\} \subseteq e_i$, $1 \leqslant i \leqslant l$.
> - ▷ If P has at least one edge, then the nodes of P are pairwise distinct, i. e., $v_i \neq v_j$ for all $0 \leqslant i < j \leqslant l$,
>
> If there is a path between v and v', then both nodes are *connected*. If the path has length l, then v is within *distance* l to v' and vice versa.

By definition, $P = (v)$ is a path of length 0 and hence v is within distance 0 to itself. As a special case, we consider paths where each pair of consecutive edges e and e' has exactly one common node, and non-consecutive edges share no node.

> Definition 6: (Simple Path)
> A path P of length l is called *simple*, if for each pair of edges e_i, e_j from P, with $1 \leqslant i < j \leqslant l$, we have that $|e_i \cap e_j| = 1$ if $j - i = 1$; otherwise $e_i \cap e_j = \emptyset$.

Using the concept of a simple path we define a special (hyper-)tree structure.

> Definition 7: (Simple Tree)
> A hypergraph H is called a *simple tree* if each pair of nodes is connected via exactly one path and all paths of H are simple.

Hence, in a simple tree no two different edges have more than a single common node.

With the above definitions, we can start with the actual proof. Consider an arbitrary but fixed node v from ${}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}}$ and its neighborhood within constant distance. According to the following lemma, a. a. s. these nodes induces a small simple tree with root v.

> Lemma 4.4.3 (Small Simple Tree, first part of the proof of [Mol04, Lemma 2.4]). Given (PAR). Let v be an arbitrary but fixed node from ${}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}}$. Let D_i be the set of nodes with distance i from v, where $0 \leqslant i \leqslant t$ for constant $t \in \mathbb{N}$. Then a. a. s. the node set $\bigcup_{i=0}^{t} D_i$ induces a simple tree with root v that has at most $\log m$ nodes.

Proof. Consider the following events

$$\mathcal{A} \coloneqq \{\text{At most } \log m \text{ nodes have distance} \leqslant t \text{ to } v.\},$$
$$\mathcal{B} \coloneqq \{\text{The nodes within distance } t \text{ to } v \text{ induce a simple tree.}\}.$$

We have to show that the probability of the event $\mathcal{A} \cap \mathcal{B}$ is $1 - o(1)$. We will do this via bounding the probability of the complementary event from above using the union bound, that is

$$\Pr(\overline{\mathcal{A}} \cup \overline{\mathcal{B}}) \leqslant \Pr(\overline{\mathcal{A}}) + \Pr(\overline{\mathcal{B}}) \stackrel{!}{=} o(1)\,.$$

Consider the event $\overline{\mathcal{A}}$. For each $0 \leqslant i \leqslant t$ let X_i be a random variable that counts the number of nodes in ${}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}}$ within distance exactly i to v. Using Markov's inequality, we get

$$\Pr(\overline{\mathcal{A}}) = \Pr\left(\sum_{i=0}^{t} X_i > \log m\right) < \frac{\sum_{i=0}^{t} \operatorname{Exp}(X_i)}{\log m}\,.$$

The expected values $\operatorname{Exp}(X_i)$, $i \in [t+1]$, can be bounded as follows.

$i = 0$. Since v is the only node with distance 0 to v, it holds $X_0 = \mathrm{Exp}(X_0) = 1$.

$i = 1$. To bound $\mathrm{Exp}(X_1)$ note that there are $\binom{m-1}{d_j-1}$ potential edges of size d_j that contain v. Each such an edge is present with probability $p_j = \alpha_j \cdot c \cdot m / \binom{m}{d_j}$ independently of the other edges. If such an edge is present, then $d_j - 1$ nodes of this edge are within distance 1 of v. Therefore, we get

$$\mathrm{Exp}(X_1) \leqslant \sum_{j\in[s]} (d_j - 1) \cdot \binom{m-1}{d_j-1} \cdot p_j \leqslant c \cdot m \cdot \sum_{j\in[s]} d_j \cdot \frac{\binom{m}{d_j-1}}{\binom{m}{d_j}} = O(1)\,,$$

using that $\binom{m}{d_j} = \Theta(m_j^d)$ for constant d_j.

$i \geqslant 2$. To bound $\mathrm{Exp}(X_i)$ for $2 \leqslant i \leqslant t$ consider all paths of length $i-1$ starting from $v = v_0$. Let $P = (v_0, e_1, v_1, e_2, \ldots, v_{i-2}, e_{i-1}, v_{i-1})$ be such a path. Each additional edge e that does not belong to P but has element v_{i-1}, the endpoint of P, contributes at most $d_j - 1$ nodes to X_i. The number of possibilities to choose the nodes $v_1, v_2, \ldots, v_{i-1}$ is bounded by m^{i-1}, the number of possibilities to choose the edges $e_1, e_2 \ldots, e_{i-1}$ of the path is bounded by $\left(\sum_{j\in[s]} \binom{m}{d_j-2}\right)^{i-1}$, and the number of possibilities to choose the last edge e is bounded by $\sum_{j\in[s]} \binom{m}{d_j-1}$. Hence, we have

$$\begin{aligned}\mathrm{Exp}(X_i) &\leqslant m^{i-1} \cdot \left(\sum_{j\in[s]} \binom{m}{d_j-2} \cdot p_j\right)^{i-1} \cdot \sum_{j\in[s]} (d_j - 1) \cdot \binom{m}{d_j-1} \cdot p_j \\ &\leqslant c^{i-1} \cdot m^{2i-2} \cdot \left(\sum_{j\in[s]} \frac{\binom{m}{d_j-2}}{\binom{m}{d_j}}\right)^{i-1} \cdot c \cdot m \cdot \sum_{j\in[s]} \frac{d_j \cdot \binom{m}{d_j-1}}{\binom{m}{d_j}} = O(1)\,.\end{aligned}$$

It follows that $\Pr(\overline{\mathcal{A}}) = O(1/\log m) = o(1)$.

Consider the event $\overline{\mathcal{B}}$. In order to bound the probability of $\overline{\mathcal{B}}$, we will make use of the event

$$\mathcal{C} \coloneqq \{\text{There is a sling starting at } v \text{ with length } 2 \leqslant l \leqslant 2 \cdot t.\}\,,$$

where *sling* is defined as follows.

DEFINITION 8: (SLING)
A *sling* $S = (P, e)$ of length l, $l \geqslant 2$, is a tuple of a path P and an edge e, where:

- ▷ The path $P = (v_0, e_1, v_1, e_2, \ldots, v_{l-2}, e_{l-1}, v_{l-1})$ has length $l - 1$.
- ▷ The last node v_{l-1} of P is element of e and it holds that $\left|e \cap \left(\bigcup_{i=1}^{l-1} e_i\right)\right| \geqslant 2$.

By the next claim, it follows that the probability of event $\mathcal{C}$ is at least as high as the probability of event $\overline{\mathcal{B}}$.

Claim 5 (No Simple Hypertree Implies a Sling). It holds $\overline{\mathcal{B}} \Rightarrow \mathcal{C}$.

Proof of Claim. Assume $\overline{\mathcal{B}}$. By definition, the connected hypergraph that is induced by the nodes within distance t to v, denoted as H_D, is not a simple tree if and only if we are in one of two cases:

1. There exists a path that is not simple.
2. All paths are simple and there exists a pair of nodes that is connected by two simple paths.

We will show that in the first as well as in the second case there must exist a sling.

ad (i): Let $P = (v, e_1, v_1, e_2, \ldots, v_{k-1}, e_k, v_k)$ be a path that is not simple. We use v as the first node of P, since there is a path from v to each other node in H_D. By definition. there is an index j, $2 \leqslant j \leqslant k$, for which it holds $|e_{j-1} \cap e_j| \geqslant 2$ or $|e_i \cap e_j| \geqslant 1$ for $i < j-1$. Hence there exists a sling $S = \big((v, e_1, v_1, e_2, \ldots, v_{j-2}, e_{j-1}, v_{j-1}), e_j\big)$ starting at v.

ad (ii): There must be two nodes u and w that are connected by two simple paths $P_0 = (u, e_1, v_2, e_2, \ldots, v_{k-1}, e_k, w)$ and $P_1 = (u, e'_1, v'_2, e'_2, \ldots, v'_{l-1}, e'_l, w)$ that have no common edge, i. e., $e_i \neq e'_j$ for all $1 \leqslant i \leqslant k$ and $1 \leqslant j \leqslant l$. Since all paths are simple, no two different edges have more than one common node. Hence the path $P = (u, e_1, v_2, e_2, \ldots, v_{k-1}, e_k, w, e'_l, v'_{l-1}, \ldots, e'_2, v'_2)$ is a simple path and $S = (P, e'_1)$ is a sling starting at u. If v is an element of an edge e_i or e'_j of P, then there exists also a sling starting at v. If v is not an element of an edge of P, then there is a simple path from v to a node v_i or v'_j of P that creates a sling starting at v.

This finishes the proof of the claim. □

For each l with $2 \leqslant l \leqslant 2 \cdot t$ let Y_l be a random variable that counts the number of slings of length l starting at v. By Claim 5 we have that $\Pr(\mathcal{C}) \geqslant \Pr(\overline{\mathcal{B}})$ and therefore

$$\Pr(\overline{\mathcal{B}}) \leqslant \Pr\left(\sum_{l=2}^{2t} Y_l \geqslant 1\right) \leqslant \sum_{l=2}^{2t} \operatorname{Exp}(Y_l) \ .$$

One can bound $\operatorname{Exp}(Y_l)$ analogous to $\operatorname{Exp}(X_i)$ via

$$\begin{aligned}\operatorname{Exp}(Y_l) &\leqslant m^{l-1} \cdot \left(\sum_{j\in[s]} \binom{m}{d_j - 2} \cdot p_j\right)^{l-1} \cdot \sum_{j\in[s]} \binom{m}{d_j - 2} \cdot p_j \\ &\leqslant c^{l-1} \cdot m^{2\cdot l-2} \cdot \left(\sum_{j\in[s]} \frac{\binom{m}{d_j-2}}{\binom{m}{d_j}}\right)^{l-1} \cdot c \cdot m \cdot \sum_{j\in[s]} \frac{\binom{m}{d_j-2}}{\binom{m}{d_j}} = O\left(m^{-1}\right) \ .\end{aligned}$$

It follows that $\Pr(\overline{\mathcal{B}}) = o(1)$ and therefore it holds $\Pr(\overline{\mathcal{A}} \cup \overline{\mathcal{B}}) = o(1)$, which finishes the proof of the lemma. ■

Now we turn our attention to `peeling_in_parallel` (Algorithm 13) and determine the probability that a node v from ${}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}}$ survives a constant number of rounds of this peeling algorithm.

LEMMA 4.4.4 (PROBABILITY OF SURVIVAL, SLIGHT GENERALIZATION OF [MOL04, LEMMA 2.4]). Given (PAR). Let v be an arbitrary but fixed node from ${}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}}$. For any constant $t \in \mathbb{N}$, let σ_t be the probability that after $t \geqslant 1$ rounds of Algorithm 13 on input $H = {}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}}$ the node v is still present in H_t. Then the following holds:

$$\sigma_t = \Pr\left(\mathrm{Po}\left[\lambda_t\right] \geqslant \ell_+\right) \pm o(1)\,,$$

with

$$\lambda_i \coloneqq c \cdot \sum_{j\in[s]} \alpha_j \cdot d_j \cdot \rho_{i-1}^{d_j-1} \qquad \text{for } i \geqslant 1,$$

and ρ_i recursively defined via

$$\rho_i \coloneqq \Pr\left(\mathrm{Po}\left[\lambda_i\right] \geqslant \ell_+ - 1\right) \qquad \text{for } i \geqslant 1,$$
$$\rho_0 \coloneqq 1\,.$$

PROOF. Let D_i be the set of nodes with distance i from v, where $0 \leqslant i \leqslant t$ for constant $t \in \mathbb{N}$. According to Lemma 4.4.3, the event

$$\mathcal{T} \coloneqq \{\text{The set } \bigcup\nolimits_{i=0}^{t} D_i \text{ induces a simple tree } H_D \text{ with root } v \text{ and at most } \log m \text{ nodes.}\}$$

occurs with probability $1 - o(1)$. Hence, in the following we consider all probabilities under the condition that $\mathcal{T}$ holds.

Now, on the simple tree H_D we apply Algorithm 14, a modified peeling method, called `peeling_tree_in_parallel`, which specifies a certain order in which nodes have to be removed. For a description of this algorithm, we make use of the following definitions.

DEFINITION 9: (EDGE TYPES)
A *child edge* of a node u from D_i is an edge incident with u where all nodes but u are from D_{i+1}. The *parent edge* of a node $u \in D_i$, $u \neq v$, is the only edge incident with u that is not a child edge of u.

The algorithm works in rounds. Let i be the number of the current iteration. In each iteration $1 \leqslant i \leqslant t-1$ we consider all nodes of D_{t-i}. A node u is removed from D_{t-i} if and only if it has at most $\ell_+ - 2$ child edges. Removing u includes removing all edges incident with u, especially its parent edge. In the last iteration t we consider $D_0 = \{v\}$ and the root v is removed if it has at most $\ell_+ - 1$ child edges.

ALGORITHM 14: `peeling_tree_in_parallel`

Input : Simple Tree $H = (V, E)$, root $v \in V$, core parameter $\ell_+ \in \mathbb{N}$ with $\ell_+ \geqslant 1$, height parameter $t \in \mathbb{N}$.
Output : Sub-tree of H.

```
for i ← 0 to t do D_i ← {u ∈ V | u has distance i to v} ;
for i ← 1 to t do
    // ⩽ ℓ+ − 2 child edges for u ≠ v, ⩽ ℓ+ − 1 child edges for u = v
    V' ← {u ∈ V | u ∈ D_{t−i} ∧ |{e ∈ E | u ∈ e}| ⩽ ℓ+ − 1};
    V ← V − V';
    E ← E − {e ∈ E | e ∩ V' ≠ ∅};
end
return (V, E)
```

CLAIM 6 (PEELING). The root node v survives t rounds of `peeling_in_parallel` (Algorithm 13) if and only if v survives `peeling_tree_in_parallel` (Algorithm 14).

PROOF OF CLAIM. Assume that v does not survive `peeling_tree_in_parallel`. Observe that all nodes from H_D, except the nodes from D_t, are incident to the same edges in H and H_D, from which it directly follows that v cannot survive t rounds of `peeling_in_parallel`.

Assume that v does not survive t rounds of `peeling_in_parallel`, then there exists a path P of length at most $t - 1$ that ends at v and is *reducible* according to the following definition. A path P of length l between two nodes is reducible if and only if either $l = 0$ and the end node is incident with at most $\ell_+ - 1$ edges, or $l > 0$ and the start node is incident with at most $\ell_+ - 1$ edges and removing these edges from ${}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}}$ results in a reducible path of length smaller than l. It follows that v cannot survive `peeling_tree_in_parallel`. □

So, according to Claim 6 it is sufficient to analyze the probability that the root node v survives `peeling_tree_in_parallel`. Our next step is to show, by induction on $i \in [t]$, the probability that an arbitrary but fixed node u of D_{t-i} survives `peeling_tree_in_parallel` is $\rho_i \pm o(1)$.

Basis, $i = 0$: According to the definition of `peeling_tree_in_parallel` no node of D_t is removed. Hence the survival probability is $1 = \rho_0$.

Induction step, $i - 1 \to i, 1 \leqslant i < t$: Assume that each node of D_{t-i+1} survives with probability $\rho_{i-1} \pm o(1)$. Now consider D_{t-i} and an arbitrary but fixed node u from this set. Let Y_j be a random variable that counts the number of child edges of u with size d_j that have survived round $i - 1$. We are interested in the probability distribution of Y_j.

In order that such a child edge e has survived round $i - 1$, two conditions must be fulfilled:

1. The edge e must have been chosen during the construction of ${}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}}$; the probability of this event is p_j.
2. All of the $d_j - 1$ nodes of e that are from D_{t-i+1} must have survived iteration number $i-1$; according to the assumption, the probability for this event is $\left(\rho_{i-1} \pm o(1)\right)^{d_j-1}$.

Let $X = |D_0 \cup D_1 \cup \ldots \cup D_{t-i}|$ be a random variable that counts the number of nodes from the current level up to the root level of the simple tree H_D. All of the nodes from these levels cannot be elements of the child edges of u, hence Y_j has distribution

$$Y_j \sim \text{Bin}\left(\binom{m-X}{d_j-1}, p_j \cdot (\rho_{i-1} \pm o(1))^{d_j-1}\right).$$

The next step is to show that Y_j is approximately Poisson distributed.

> Claim 7 (Surviving Child Edges of Size d_j). The expected value of Y_j is $\text{Exp}(Y_j) = c \cdot \alpha_j \cdot d_j \cdot \rho_{i-1}^{d_j-1} \pm o(1)$.

Proof of Claim. According to the distribution of Y_j, we have

$$\text{Exp}(Y_j) = \left(\rho_{i-1} \pm o(1)\right)^{d_j-1} \cdot c \cdot \alpha_j \cdot m \cdot \frac{\binom{m-X}{d_j-1}}{\binom{m}{d_j}}.$$

Let x be the realization of X. Since, we assume that event $\mathcal{T}$ holds, we have that $1 \leqslant x \leqslant \log m$. All we need is to show that $\binom{m-x}{d-1} \cdot m / \binom{m}{d} \stackrel{!}{=} d \pm o(1)$, for constant $d \geqslant 3$. Using factorial representation, we get

$$\binom{m-x}{d-1} \cdot \frac{m}{\binom{m}{d}} = d \cdot \frac{(m-x)! \cdot (m-d)!}{(m-d-(x-1))! \cdot (m-1)!} = d \cdot \underbrace{\prod_{i=0}^{x-2} \frac{m-d-i}{m-1-i}}_{=:P}.$$

Since it holds

$$\frac{m-d-i}{m-1-i} \stackrel{!}{\geqslant} \frac{m-d-(i+1)}{m-1-(i+1)} \Leftrightarrow -2m+2d \geqslant -2m+d+1 \Leftrightarrow d \geqslant 1,$$

it follows that

$$\left(\frac{m-d-x+2}{m-x+1}\right)^{x-1} \leqslant P \leqslant \left(\frac{m-d}{m-1}\right)^{x-1}.$$

It remains to show that $P = 1 \pm o(1)$. Using that $1 + z \leqslant \exp(z)$, the limit of the upper bound is bounded by

$$\lim_{m\to\infty} \left(\frac{m-d}{m-1}\right)^{x-1} = \lim_{m\to\infty} \left(1 - \frac{d-1}{m-1}\right)^{x-1} \leqslant \lim_{m\to\infty} \exp\left(\frac{(-d+1)\cdot(x-1)}{m-1}\right) = 1.$$

Using Bernoulli's inequality, the limit of the lower bound is bounded by

$$\lim_{m\to\infty}\left(\frac{m-d-x+2}{m-x+1}\right)^{x-1} = \lim_{m\to\infty}\left(1-\frac{d-1}{m-x+1}\right)^{x-1} \geqslant 1+\lim_{m\to\infty}\frac{(-d+1)\cdot(x-1)}{m-x+1} = 1\,.$$

Hence the claim follows. □

By Claim 7, the expected value of Y_j can be made arbitrarily close to a positive constant. Hence, the distribution $\mathrm{Po}\left(\lim_{n\to\infty}\mathrm{Exp}(Y_j)\right)$ is the limit distribution of Y_j, see, e. g., [MU05, Theorem 5.5]. Therefore, for any fixed y we have

$$\Pr(Y_j = y) = \Pr\left(\mathrm{Po}\left[c\cdot\alpha_j\cdot d_j\cdot\rho_{i-1}^{d_j-1}\right] = y\right) \pm o(1)\,.$$

Since $Y_0, Y_1, \ldots, Y_{s-1}$ are independent, their sum $Y = \sum_{j\in[s]} Y_j$ is also approximately Poisson distributed with parameter $\lambda_i = c\cdot\sum_{j\in[s]}\alpha_j\cdot d_j\cdot\rho_{i-1}^{d_j-1}$. Therefore, we can express the probability that u survives round i using Y, via

$$\Pr\left(Y \geqslant \ell_+ - 1\right) = \Pr\left(\mathrm{Po}\left[\lambda_i\right] \geqslant \ell_+ - 1\right) \pm o(1) = \rho_i \pm o(1)\,.$$

This finishes the induction step.

We conclude that the root node v survives the last iteration t with probability

$$\sigma_t = \Pr\left(\mathrm{Po}\left[\lambda_t\right] \geqslant \ell_+\right) \pm o(1)\,.$$

This concludes the proof of the lemma. ■

Let V_t be the number of nodes of ${}^{\mathrm{bin}}H^{\mathbf{d}}_{m,\mathbf{p}} = (V, E)$ after t of rounds of Algorithm 13, especially let $V_0 = V$. The last building block to complete Lemma 4.4.1 is the following lemma, which states that the number of nodes that survive is concentrated around $m\cdot\sigma_t$, if t is constant.

> Lemma 4.4.5 (Number of Surviving Nodes, slight generalization of [Mol04, Lemma 2.5]). Given (PAR). For any constant $t\in\mathbb{N}$ and constant $\gamma > \frac{1}{2}$ it a. a. s. holds that $\left||V_t| - \sigma_t\cdot m\right| < m^\gamma$.

Proof. For each node v from V let X_v be a binary random variable with

$$X_v \coloneqq \begin{cases} 1, & \text{if } v \text{ survives } t \text{ iterations of } \texttt{peeling\_in\_parallel} \\ 0, & \text{otherwise} \end{cases}.$$

Let $\sigma \coloneqq \sigma_t = \Pr(X_v = 1)$. For each node v from V we define a random variable $Y_v = X_v - \sigma$, i. e.,

$$Y_v \coloneqq \begin{cases} 1-\sigma, & \text{if } X_v = 1 \text{ (with probability } \sigma) \\ -\sigma, & \text{if } X_v = 0 \text{ (with probability } 1-\sigma) \end{cases}.$$

Let $P \coloneqq \Pr\left(\left||V_t| - \sigma_t \cdot m\right| < m^\gamma\right)$, for constant $\gamma > 1/2$. Using the random variables X_v and Y_v, we get

$$P = \Pr\left(\left|\sum_{v\in V} X_v - \sigma \cdot m\right| < m^\gamma\right) = \Pr\left(\left|\sum_{v\in V} Y_v\right| < m^\gamma\right).$$

We will show that $P \stackrel{!}{=} 1 - o(1)$ via bounding $1 - P$ from above. It holds

$$1 - P = \Pr\left(\left|\sum_{v\in V} Y_v\right| \geqslant m^\gamma\right) = \Pr\left(\left(\sum_{v\in V} Y_v\right)^2 \geqslant m^{2\cdot\gamma}\right) \leqslant \frac{\operatorname{Exp}\left(\left(\sum_{v\in V} Y_v\right)^2\right)}{m^{2\cdot\gamma}},$$

by Markov's inequality. Expanding the numerator of the right-hand side gives

$$\operatorname{Exp}\left(\left(\sum_{v\in V} Y_v\right)^2\right) = \underbrace{\operatorname{Exp}\left(\sum_{v\in V} Y_v^2\right)}_{=:S_0} + \underbrace{\operatorname{Exp}\left(\sum_{\substack{u,v\in V\\ u\neq v}} Y_u \cdot Y_v\right)}_{=:S_1}.$$

By linearity of expectation, the first summand is

$$\begin{aligned} S_0 &= m \cdot \operatorname{Exp}\left(Y_v^2\right) = m \cdot \left((1-\sigma)^2 \cdot \sigma + (-\sigma)^2 \cdot (1-\sigma)\right) \\ &= m \cdot \sigma \cdot (1-\sigma) \leqslant 1/4 \cdot m. \end{aligned}$$

For each pair u and v of different nodes from V, consider the event

$$\mathcal{A}_{u,v} \coloneqq \{\text{There is a path of length} \leqslant t \text{ that connects } u \text{ and } v\}.$$

Let u and v be two arbitrary but fixed different nodes. By linearity of expectation, the second summand can be written as $S_1 = m \cdot (m-1) \cdot \operatorname{Exp}(Y_u \cdot Y_v)$, and by the law of total expectation, we get

$$\begin{aligned} S_1 = m \cdot (m-1) \cdot \Big(&\underbrace{\operatorname{Exp}(Y_u \cdot Y_v \mid \overline{\mathcal{A}}_{u,v})}_{=:T_0} \cdot \underbrace{\Pr(\overline{\mathcal{A}}_{u,v})}_{=:T_1} \\ &+ \underbrace{\operatorname{Exp}(Y_u \cdot Y_v \mid \mathcal{A}_{u,v})}_{=:T_2} \cdot \underbrace{\Pr(\mathcal{A}_{u,v})}_{=:T_3} \Big). \end{aligned}$$

It remains to bound the terms T_0, T_1, T_2, and T_3. Conditioned on the event $\overline{\mathcal{A}}_{u,v}$, the survival probabilities of the nodes u and v during the first t iterations of algorithm `peeling_in_parallel` are independent. Hence

$$\begin{aligned} T_0 &= (1-\sigma)^2 \cdot \sigma^2 + (-\sigma)^2 \cdot (1-\sigma)^2 + (-\sigma) \cdot (1-\sigma) \cdot 2 \cdot \sigma \cdot (1-\sigma) \\ &= 2 \cdot (1-\sigma)^2 \cdot \sigma^2 - 2 \cdot (1-\sigma)^2 \cdot \sigma^2 = 0. \end{aligned}$$

We use $T_1 \leqslant 1$, as well as $T_2 \leqslant 1$, since $Y_u, Y_v \in [-1, 1]$. It remains to bound T_3, i. e., the probability of the event $\mathcal{A}_{u,v}$. Let Z_i be a random variable counting the number of paths of length i that connect u and v. It holds by Markov's inequality

$$\Pr(\mathcal{A}_{u,v}) = \Pr\left(\sum_{i=1}^{t} Z_i \geqslant 1\right) \leqslant \sum_{i=1}^{t} \operatorname{Exp}(Z_i)\,.$$

Analogous as in the proof of Lemma 4.4.3, it follows that $\operatorname{Exp}(Z_i) = O(m^{-1})$, for constant i, and therefore $T_3 = O(m^{-1})$. So, the second summand can bounded via $S_1 = O(m)$, and we get

$$1 - P \leqslant \frac{S_0 + S_1}{m^{2\cdot\gamma}} = O(m^{1-2\cdot\gamma})\,,$$

which is in $o(1)$ for constant $\gamma > 1/2$. This completes the proof of the lemma. ■

In order to bound σ_t, we consider λ_t using the following functions

$$f(x) = \Pr(\mathrm{Po}[x] \geqslant \ell_+ - 1) = 1 - \sum_{j=0}^{\ell_+-2} \frac{x^j}{j!} \cdot e^{-x}$$
$$g(x) = c \cdot \sum_{j\in[s]} \alpha_j \cdot d_j \cdot x^{d_j-1}\,.$$

According to the definitions of Lemma 4.4.4, we have $\lambda_i = g(\rho_{i-1})$, for $i \geqslant 1$, and $\rho_i = f(\lambda_i) = f\big(g(\rho_{i-1})\big)$, for $i \geqslant 1$, as well as $\rho_0 = 1$.

> Claim 8 (Limits). Let $\rho \coloneqq \lim_{i\to\infty} \rho_i$ and let $\lambda \coloneqq \lim_{i\to\infty} \lambda_i$ be the limits of the sequences $(\rho_i)_{i\geqslant 0}$ and $(\lambda_i)_{i\geqslant 1}$. Then we have $\rho = \lambda = 0$.

Proof of Claim. First note that $\rho_i = f(\lambda_i)$ is a probability and therefore $\rho_i \in [0, 1]$, as well as $\rho_1 \leqslant \rho_0 = 1$. The first derivative of $f(x)$ is

$$\frac{df(x)}{dx} = \frac{x^{\ell_+-2}}{(\ell_+ - 2)!} \cdot e^{-x}\,,$$

and the first derivative of $g(x)$ is

$$\frac{dg(x)}{dx} = c \cdot \sum_{j\in[s]} \alpha_j \cdot d_j \cdot (d_j - 1) \cdot x^{d_j-2}\,.$$

For $x > 0$, we have that $\frac{df(x)}{dx} > 0$ and $\frac{dg(x)}{dx} > 0$, i. e., the functions f and g are strictly increasing.

First, assume for a contradiction that the sequence $(\rho_i)_{i\geqslant 0}$ is *not* non-increasing. Let $j \geqslant 2$ be the smallest index such that $\rho_j > \rho_{j-1}$. Then we have

$$\rho_j > \rho_{j-1} \Rightarrow f(\rho_j) > f(\rho_{j-1}) \Rightarrow f\big(g(\rho_{j-1})\big) > f\big(g(\rho_{j-2})\big) \Rightarrow \rho_{j-1} > \rho_{j-2}\,,$$

which is a contradiction to the minimality of j. Hence the sequence $(\rho_i)_{i\geqslant 0}$ is non-increasing and the limit ρ exists.

Second, assume for a contradiction that $\rho > 0$. Since we have $\rho = f(g(\rho))$, it follows that $g(\rho) = g\Big(f\big(g(\rho)\big)\Big)$, which by definition of f and g is equivalent to

$$g(\rho) = c \cdot \sum_{j\in[s]} \alpha_j \cdot d_j \cdot \big(\Pr(\mathrm{Po}[g(\rho)] \geqslant \ell_+ - 1)\big)^{d_j - 1}.$$

Note that $\lambda = g(\rho)$. By assumption, we have $\rho > 0$ and therefore $\lambda > 0$. It follows that

$$c = \frac{\lambda}{\sum\limits_{j\in[s]} \alpha_j \cdot d_j \cdot \big(\Pr(\mathrm{Po}[\lambda] \geqslant \ell_+ - 1)\big)^{d_j - 1}}.$$

Since we are in the case $c \leqslant \check{c}_{\ell_+}(\mathbf{d}, \boldsymbol{\alpha}) - \varepsilon$, for constant $\varepsilon > 0$, the last equation contradicts the definition of $\check{c}_{\ell_+}$. Hence, it must hold $\rho = 0$ and $\lambda = 0$, respectively. □

Let $\delta > 0$ be arbitrary but fixed. In consequence of the last claim, there must be a constant $t = t(\delta)$, such that $\lambda_t < \delta/2$ and even $\sigma_t = \lambda_t \pm o(1) < \delta/2$, for sufficiently large m. According to Lemma 4.4.5, we have $|V_t| < \delta \cdot m$ with probability $1 - o(1)$. This finishes the proof of Lemma 4.4.1. ■

4.4.1.2. Small Average Degree

Using the proof idea from [Mol04, Lemma 2.3], we show Lemma 4.4.2, which states that there exists a constant $\delta > 0$, such that a. a. s. the following event does not occur

$$\begin{aligned}\mathcal{A} \coloneqq \{&{}^{\mathrm{sim}}H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}} \text{ has a sub-hypergraph with at most } \delta \cdot m \text{ vertices}\\ &\text{and average degree at least } \ell_+.\}.\end{aligned}$$

Let $\hat{d} = \max\{d_i \mid i \in [s]\}$ and $\check{d} = \min\{d_i \mid i \in [s]\}$. Without loss of generality assume $V = [m]$. For each $y \in V$ define the following event:

$$\begin{aligned}\mathcal{B}^{y}_{(x_i)_{i\in[s]}} \coloneqq \{&\text{The sub-hypergraph on the node set } \{0, 1, \ldots, y-1\}\\ &\text{has exactly } x_i \text{ edges of size } d_i.\}.\end{aligned}$$

By union bound, we get

$$\Pr(\mathcal{A}) \leqslant \sum_{y=\check{d}}^{\delta\cdot m} \sum_{\substack{\mathbf{x}\in\mathbb{N}^s, j\in[\hat{d}] \text{ with}\\ \sum_{i\in[s]} x_i\cdot d_i = \ell_+\cdot y + j}} \binom{m}{y} \cdot \Pr(\mathcal{B}^{y}_{\mathbf{x}}).$$

For each y with $\check{d} \leqslant y \leqslant \delta \cdot m$ consider a vector $\big(\beta_i(y) \cdot n\big)_{i\in[s]}$ of edge counts that maximizes the probability that the sub-hypergraph on the node set $[y]$ has $\beta_i(y) \cdot n$

edges of size d_i for all $i \in [s]$, under the condition that the average node degree is equal to or just above ℓ_+. More formally, we define

$$\big(\beta_i(y) \cdot n\big)_{i\in[s]} = \arg\max_{\boldsymbol{x}} \Big\{ \Pr(\mathcal{B}^y_{\boldsymbol{x}}) \mid \boldsymbol{x} \in \mathbb{N}^s, \sum_{i\in[s]} x_i \cdot d_i = \ell_+ \cdot y + j \text{ for } j \in [\hat{d}] \Big\}.$$

Furthermore, let $\tilde{\ell}_+(y) = \Big(\sum_{i\in[s]} \beta_i(y) \cdot n \cdot d_i \Big)/y$ be the corresponding average degree with $\tilde{\ell}_+(y) \geqslant \ell_+$. Note that $\beta_i = \beta_i(y) \leqslant \alpha_i$ for all $i \in [s]$, since the maximum number of edges of size d_i is $\alpha_i \cdot n$.

Using these definitions, we get

$$\Pr(\mathcal{A}) \leqslant \sum_{y=\check{d}}^{\delta\cdot m} O(y^{s-1}) \cdot \underbrace{\binom{m}{y} \cdot \Pr(\mathcal{B}^y_{(\beta_i(y)\cdot n)_{i\in[s]}})}_{=:P_y},$$

with

$$P_y = \underbrace{\binom{m}{y}}_{=:P_0} \cdot \underbrace{\prod_{i\in[s]} \binom{\alpha_i \cdot n}{\beta_i \cdot n}}_{=:P_1} \cdot \underbrace{\prod_{i\in[s]} \left(\frac{\binom{y}{d_i}}{\binom{m}{d_i}} \right)^{\beta_i \cdot n}}_{=:P_2}.$$

Consider an arbitrary but fixed factor P_y. Let $y = \mu \cdot m$ where $\check{d}/m \leqslant \mu \leqslant \delta < 1$, as well as let $\beta_i \cdot n = \nu_i \cdot y \cdot \tilde{\ell}_+/d_i$ for all $i \in [s]$, where $\sum_{i\in[s]} \nu_i = 1$.

Recall that $\alpha_i \leqslant 1$ and $c \leqslant \ell_+ \leqslant \tilde{\ell}_+$. In order to bound $P_y = P_{\mu\cdot m}$ from above we use the following inequalities.

$$\begin{aligned}
P_0 &= \binom{m}{\mu \cdot m} \leqslant \left(\left(\frac{1}{\mu}\right)^{\mu} \cdot \left(\frac{1}{1-\mu}\right)^{1-\mu} \right)^m \\
P_1 &\leqslant \prod_{i\in[s]} \binom{\alpha_i \cdot c \cdot m}{\nu_i \cdot y \cdot \tilde{\ell}_+/d_i} \leqslant \prod_{i\in[s]} \binom{\tilde{\ell}_+ \cdot m}{\nu_i \cdot \mu \cdot m \cdot \tilde{\ell}_+/d_i} \\
&\leqslant \left(\prod_{i\in[s]} \left(\frac{d_i}{\nu_i \cdot \mu}\right)^{\frac{\nu_i\cdot\mu}{d_i}} \cdot \left(\frac{1}{1-\frac{\nu_i\cdot\mu}{d_i}}\right)^{1-\frac{\nu_i\cdot\mu}{d_i}} \right)^{\tilde{\ell}_+\cdot m} \\
P_2 &\leqslant \prod_{i\in[s]} \left(\left(\frac{y}{m}\right)^{d_i} \right)^{\nu_i\cdot y\cdot\tilde{\ell}_+/d_i} = \mu^{\sum_{i\in[s]} \nu_i\cdot\mu\cdot m\cdot\tilde{\ell}_+} = \mu^{\mu\cdot m\cdot\tilde{\ell}_+}.
\end{aligned}$$

Applying the entropy function $H_e(\varepsilon) = -\varepsilon \cdot \ln(\varepsilon) - (1-\varepsilon) \cdot \ln(1-\varepsilon)$ gives

$$P_y = f(\mu) = \exp\Big(m \cdot \underbrace{\big(H_e(\mu) + \tilde{\ell}_+ \cdot \sum_{i\in[s]} H_e(\nu_i \cdot \mu/d_i) + \mu \cdot \tilde{\ell}_+ \cdot \ln(\mu) \big)}_{=:g(\mu)} \Big).$$

By substituting P_y with $f(\mu)$, we get the following upper bound

$$\begin{aligned}\Pr(\mathcal{A}) &\leqslant \sum_{y=\check{d}}^{\delta\cdot m} O(y^{s-1})\cdot f(\mu)\\ &\leqslant O(\ln(m)^{s-1})\cdot \ln(m)\cdot \max\{f(\mu)\mid \check{d}/m\leqslant\mu\leqslant \ln(m)/m\}\\ &\quad +O(m^{s-1})\cdot m\cdot \max\{f(\mu)\mid \ln(m)/m\leqslant\mu\leqslant\delta\}.\end{aligned}$$

For rest of the proof, we study $f(\mu)$ and $g(\mu)$.

> Claim 9 (Error Terms).
>
> (i) $f(\check{d}/m) = O(m^{-1})$.
>
> (ii) $f\big(\ln(m)/m\big) = O\big((\ln(m)/m)^{\ln(m)/3}\big)$.
>
> (iii) For every suitable ℓ_+, $\check{d}$ and s there exists a constant δ such that for all μ with $\check{d}/m\leqslant\mu\leqslant\delta$, it holds $\frac{dg}{d\mu}(\mu) < 0$.

It follows by claims (i) and (iii) that $\max\{f(\mu)\mid \check{d}/m\leqslant\mu\leqslant\ln(m)/m\} = O(m^{-1})$, and it follows by claims (ii) and (iii) that $\max\{f(\mu)\mid \check{d}/m\leqslant\mu\leqslant\ln(m)/m\} = O\big((\ln(m)/m)^{\ln(m)/3}\big)$. Hence, assuming the claim, we conclude

$$\Pr(\mathcal{A}) \leqslant O\big((\ln(m))^s\big)\cdot O(m^{-1}) + O(m^s)\cdot O\big((\ln(m)/m)^{\ln(m)/3}\big) = o(1)\,.$$

So it remains to show Claim 9.

Proof of Claim. Using that

$$\left(\frac{1}{1-\varepsilon}\right)^{1-\varepsilon} = \left(1+\frac{\varepsilon}{1-\varepsilon}\right)^{1-\varepsilon} \leqslant e^{\varepsilon/(1-\varepsilon)\cdot(1-\varepsilon)} = e^{\varepsilon}\,,$$

we can bound $f(\mu)$ via

$$\begin{aligned}f(\mu) &\leqslant \left(\frac{1}{\mu}\right)^{\mu\cdot m}\cdot e^{\mu\cdot m}\cdot\left(\prod_{i\in[s]}\left(\frac{d_i}{\nu_i\cdot\mu}\right)^{\nu_i\cdot\mu/d_i}\cdot e^{\nu_i\cdot\mu/d_i}\right)^{\tilde{\ell}_+\cdot m}\cdot\mu^{\mu\cdot m\cdot\tilde{\ell}_+}\\ &\leqslant \left(\frac{e}{\mu}\right)^{\mu\cdot m}\cdot\left(\frac{1}{\mu}\right)^{\mu\cdot\tilde{\ell}_+\cdot m/\check{d}}\cdot\prod_{i\in[s]}\left(\frac{d_i}{\nu_i}\right)^{\nu_i\cdot\mu\cdot\tilde{\ell}_+\cdot m/d_i}\cdot\mu^{\mu\cdot m\cdot\tilde{\ell}_+}\\ &\leqslant l^{\mu\cdot m}\cdot\left(\frac{1}{\mu}\right)^{\mu\cdot m+\mu\cdot\tilde{\ell}_+\cdot m/\check{d}-\mu\cdot m\cdot\tilde{\ell}_+},\end{aligned}$$

for some constant l.

ad (i): Using the above formula, we get

$$f(\check{d}/m) \leqslant l^{\check{d}}\cdot\left(\frac{m}{\check{d}}\right)^{\check{d}+\tilde{\ell}_+-\check{d}\cdot\tilde{\ell}_+}.$$

We want to have $\check{d} + \tilde{\ell}_+ - \check{d} \cdot \tilde{\ell}_+ \overset{!}{\leqslant} -1$, which is equivalent to $\check{d} \overset{!}{\geqslant} (\tilde{\ell}_+ + 1)/(\tilde{\ell}_+ - 1)$ using that $\tilde{\ell}_+ \geqslant 2$. This inequality is true, since $\check{d} \geqslant 3$ and $(\tilde{\ell}_+ + 1)/(\tilde{\ell}_+ - 1)$ is monotonically decreasing with growing $\tilde{\ell}_+$.

ad (ii): Analogously, it holds that

$$f(\ln(m)/m) \leqslant l^{\ln(m)} \cdot \left(\frac{m}{\ln(m)}\right)^{\ln(m)\cdot(1+\tilde{\ell}_+/\check{d}-\tilde{\ell}_+)}.$$

With $\check{d} \overset{!}{\geqslant} (\tilde{\ell}_+ + 1)/(\tilde{\ell}_+ - 1)$, it follows that $1 + \tilde{\ell}_+/\check{d} - \tilde{\ell}_+ \leqslant -(\tilde{\ell}_+ - 1)/(\tilde{\ell}_+ + 1)$, which is at most $-1/3$, since $(\tilde{\ell}_+ - 1)/(\tilde{\ell}_+ + 1)$ is monotonically increasing with growing $\tilde{\ell}_+ \geqslant 2$.

ad (iii): With $\frac{dH_e(\mu)}{d\mu} = -\ln(\mu) + \ln(1-\mu) = \ln\big((1-\mu)/\mu\big)$, we get for the first derivative of g

$$\begin{aligned}
\frac{dg(\mu)}{d\mu} &= \ln\left(\frac{1-\mu}{\mu}\right) + \tilde{\ell}_+ \cdot (\ln(\mu)+1) + \tilde{\ell}_+ \cdot \sum_{i\in[s]} \ln\left(\frac{1-\nu_i\cdot\mu/d_i}{\nu_i\cdot\mu/d_i}\right)\cdot\frac{\nu_i}{d_i} \\
&\leqslant \ln\left(\frac{1-\mu}{\mu}\right) + \ln(\mu^{\tilde{\ell}_+}) + \tilde{\ell}_+ + \tilde{\ell}_+ \cdot \sum_{i\in[s]} \ln\left(\frac{1}{\nu_i\cdot\mu/d_i}\right)\cdot\frac{\nu_i}{d_i} \\
&\leqslant \ln\left(\frac{1}{\mu}\right) + \ln(\mu^{\tilde{\ell}_+}) + \tilde{\ell}_+ + \tilde{\ell}_+ \cdot \ln\left(\prod_{i\in[s]}\left(\frac{1}{\mu}\right)^{\frac{\nu_i}{d_i}} \cdot \underbrace{\left(\frac{1}{\nu_i/d_i}\right)^{\frac{\nu_i}{d_i}}}_{=:h(d_i/\nu_i)}\right).
\end{aligned}$$

Since the only root of $\frac{dh(x)}{dx} = -x^{\frac{1}{x}-2}\cdot\ln(x) + x^{\frac{1}{x}-2}$ is e, which is a local maximum point of $h(x)$, we can bound $h(d_i/\nu_i)$ by $e^{1/e} < 1.5$. Hence, we get

$$\begin{aligned}
\frac{dg(\mu)}{d\mu} &\leqslant \ln\left(\mu^{\tilde{\ell}_+-1}\right) + \tilde{\ell}_+ + \tilde{\ell}_+ \cdot \ln\left(\left(\frac{1}{\mu}\right)^{\sum_{i\in[s]}\nu_i/\check{d}} \cdot 1.5^s\right) \\
&\leqslant \ln\left(\mu^{\tilde{\ell}_+-1}\right) + \ln\left(\left(\frac{1}{\mu}\right)^{\tilde{\ell}_+/\check{d}}\right) + k,
\end{aligned}$$

for some constant k. Furthermore, it holds

$$\begin{aligned}
k + \ln\left(\mu^{\tilde{\ell}_+-1-\tilde{\ell}_+/\check{d}}\right) \overset{!}{<} 0 &\Leftrightarrow \underbrace{(\tilde{\ell}_+ - 1 - \tilde{\ell}_+/\check{d})}_{>0}\cdot\underbrace{\ln(\mu)}_{<0} < -k \\
&\Leftrightarrow \mu^{\tilde{\ell}_+-1-\tilde{\ell}_+/\check{d}} < e^{-k} \Leftrightarrow \mu < e^{-k/(\tilde{\ell}_+-1-\tilde{\ell}_+/\check{d})},
\end{aligned}$$

where $\tilde{\ell}_+ - 1 - \tilde{\ell}_+/\check{d} \geqslant 1$, analogously as above. That is, there is some (small) constant $\delta < e^{-k/(\tilde{\ell}_+-1-\tilde{\ell}_+/\check{d})}$, such that $g(\mu)$ as well as $f(\mu)$ are monotonically decreasing for increasing $\mu \leqslant \delta$.

This finishes the proof of the claim. □

The proof of the last claim finishes the proof of Lemma 4.4.2. ■

4.4.2. Upper Bound

Analogously to the lower bound of the threshold for the appearance of an ℓ_+-core, a proof of the upper bound can be obtained from a slight generalization of the proof of [Mol04, Theorem 1.2 b]. The approach can be sketched as follows:

Let $c \geqslant \check{c}_{\ell_+}(\mathbf{d}, \boldsymbol{\alpha}) + \varepsilon$ for constant $\varepsilon > 0$. We define σ_t, the probability that an arbitrary but fixed node survives t rounds of `peeling_in_parallel`, as well as λ_i, and ρ_i as in Lemma 4.4.4. Consider the limits $\lambda := \lim_{i\to\infty} \lambda_i$ and $\rho := \lim_{i\to\infty} \rho_i$. Following the reasoning of the proof of Claim 8, one concludes that the sequences $(\rho_i)_{i\geqslant 0}$ and $(\lambda_i)_{i\geqslant 0}$ are non-increasing but the equation $c = \text{key}(\lambda, \mathbf{d}, \boldsymbol{\alpha}, \ell_+ - 1)$ holds, i. e., $\rho > 0$ and $\lambda > 0$. Hence, for arbitrary constant $\delta > 0$ and sufficiently large $t(\delta)$, we can bound σ_t via $\Pr(\text{Po}[\lambda \geqslant \ell_+]) \leqslant \sigma_t \leqslant \Pr(\text{Po}[\lambda \geqslant \ell_+]) + \delta$. Since, according to Lemma 4.4.5, the number of nodes after t rounds of algorithm `peeling_in_parallel` is a. a. s. concentrated around $m \cdot \sigma_t$, it remains to show that the subsequent peeling steps remove less than $\Pr(\text{Po}[\lambda \geqslant \ell_+]) \cdot m$ nodes, such that the result of Algorithm 1 is non-empty. This can be shown along the lines of the proof of [Mol04, Lemma 2.2 b].

In addition, we propose an alternative way, following the work of Kim [Kim06, Kim08]. Based on a method for determining realizations of ${}^{\text{poi}}H^{\mathbf{d}}_{m,\boldsymbol{\Lambda}}$ that supersedes Algorithm 12, one could possibly derive a proof of the upper bound for random non-uniform Poisson cloning hypergraphs as slight generalization of [Kim06, Theorem 6.2]. Then, starting with Lemma 4.3.3, one uses the asymptotic equivalence of different hypergraph models in order to transfer the bound to $H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$, see Section 4.3.2.2.

We close the section with a description of the alternative construction algorithm, a variation of the *cut-off line algorithm* from [Kim06, Section 3]. This part can be skipped by a reader who is not interested in elaborating all the details for a generalization of [Kim06, Section 6], especially for a full proof of [Kim06, Theorem 6.2] concerning the size of the ℓ_+-core of ${}^{\text{poi}}H^{\mathbf{d}}_{m,\boldsymbol{\Lambda}}$ for $c \geqslant \check{c}_{\ell_+}(\mathbf{d}, \boldsymbol{\alpha}) + \varepsilon$ with constant $\varepsilon > 0$.

Given (PAR) and a node set V. Let $\Lambda = \sum_{i\in[s]} \Lambda_i$ and without loss of generality assume $V = [m]$. First determine the total number of clones m_{cl} as realization of a random variable $\text{Po}[\Lambda \cdot m]$. Then obtain a sequence of edge sizes $(\tilde{d}_j)_{j\in[l]}$, where l is the smallest integer such that $\sum_{j\in[l]} \tilde{d}_j \geqslant m_{\text{cl}}$, where for all $i \in [s]$ and for all $j \in [l]$ we have $\Pr(\tilde{d}_j = d_i) = \alpha_i$. Now, analogously to Algorithm 12, if $\sum_{j\in[l]} \tilde{d}_j$ is strictly larger than m_{cl}, then the algorithm fails and we return a hypergraph with empty edge set; otherwise we continue with the generation of m_{cl} clones. Each such clone is represented by a triple (x, y, z), where

- ▷ The x-coordinate is fully randomly chosen from the real interval $[0, \Lambda]$.
- ▷ The y-coordinate is fully randomly chosen from the set $[m]$, with replacement.
- ▷ The z-coordinate is fully randomly chosen from $[m_{\text{cl}}]$, without replacement.

The y-value of each clone equals the node that corresponds to this clone, while the xz-values play a central role in the way the edge set is determined, which comes next.

For a short description, we use the following definitions. A clone is *matched* if it is used to build an edge; otherwise the clone is *unmatched*. A node is *light* if its number of unmatched clones is smaller than ℓ_+; otherwise the node is *heavy*. In addition, a clone is light/heavy if its corresponding node is light/heavy. Let $\Lambda = \tilde{\Lambda}_0$. The edge selection process is carried out in l rounds, determining one edge per round. Assume, we are in round $j \in [l]$. The first clone $\mathbf{u}_0 = (x_0, y_0, z_0)$ that is used to build the current edge e is the unmatched clone that has the smallest z-value among all unmatched light clones, or if no such clones exist, among all unmatched heavy clones. The remaining clones $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_{\tilde{d}_j - 1}$ that are associated with e are the $\tilde{d}_j - 1$ (unmatched) clones whose x-values are largest among all clones (except u) that have an x-value of at most $\tilde{\Lambda}_i$. Finally, we set $\tilde{\Lambda}_{i+1}$ to the smallest x-value of $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_{\tilde{d}_j - 1}$ and define the edge e as multiset of the y-values of $\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_{\tilde{d}_j - 1}$, i. e., the $\tilde{d}_j$ clones become matched.

Remark. Choosing the $\tilde{d}_j - 1$ clones can be visualized as follows. Consider the projection of the clones on the xy-plane and a vertical line through $(\tilde{\Lambda}_i, 0)$, called the cut-off line. Move the cut-off line to the left until $\tilde{d}_j - 1$ unmatched clones (except) $\mathbf{u}$ are on the line or to its right. The advantage of this algorithm compared to Algorithm 12 is that the process of determining a realization H of ${}^{\text{poi}}H^{\mathbf{d}}_{m,\Lambda}$ until no unmatched light clones exist is equal to the peeling of H using Algorithm 1.

4.5. Maximum Thresholds for the Appearance of 2-Cores in Non-Uniform Hypergraphs

In this section we give the full proof of Theorem 4.5 — presented before in [Rin12, Rin13]. Given a constant vector $\mathbf{d} = (d_0, d_1)$ with two edge sizes, we show how to determine the optimal (expected) fraction of edges of each size $\boldsymbol{\alpha}^* = (\alpha^*, 1 - \alpha^*)$, such that the threshold of the appearance of a 2-core in $H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ (type B) is maximal. More precisely, we solve the non-linear optimization problem (OPT) on page 104 for the case $s = 2$. As is to be expected, the proof mainly employs methods from calculus.

The structure of this section is as follows. First, we restate the optimization problem using a transformed objective function with adjusted domain. Then, in Section 4.5.2, we give the first and second partial derivatives of the objective function; in addition, we define helper functions and list some of their properties as far as they are applied in the analysis of the optimization problem. It follows the central part, Section 4.5.3, which contains the characterization of optimum solutions using the helper functions, as well as an algorithm for computing these solutions efficiently. Experimental results described in Section 4.5.4 confirm the theoretical findings. The section closes with Section 4.5.5, which contains the proofs of the properties of the auxiliary functions that were stated before.

! In order to improve readability, we substitute d_0 by l and d_1 by k throughout most of the section.

4.5.1. Problem Transformation

Our first step is to transform the key function key $\big(\lambda, (l,k), (\alpha, 1-\alpha)\big)$ from page 103 into a more manageable function $T(z,l,k,\alpha)$, called *threshold function*, using a monotonic and bijective domain mapping via

$$z = 1 - e^{-\lambda} \text{ and } \lambda = -\ln(1-z)\,.$$

Then, we restate the optimization problem (OPT) as follows.

Given $l \in \mathbb{N}$ and $k \in \mathbb{N}$.

Objective $\max\limits_{\alpha} \min\limits_{z} T(z,l,k,\alpha)$.

Objective Function

$$T(z,l,k,\alpha) = \frac{-\ln(1-z)}{\alpha \cdot l \cdot z^{l-1} + (1-\alpha)\cdot k \cdot z^{k-1}}\,. \tag{4.1}$$

Constraints $3 \leqslant l < k$, $\alpha \in (0,1]$, and[13] $z \in (0,1)$.

4.5.2. Preparations

Here we state some basic facts about the threshold function T and several auxiliary functions which we will use in the subsequent proof.

4.5.2.1. Derivatives

We need to determine the partial derivatives of $T(z,l,k,\alpha)$ with respect to z and α. To shorten and simplify notation, we use the following definitions. For all $j \in \mathbb{N}$ let

$$\begin{aligned} D_j(z,l,k,\alpha) &= \alpha \cdot l \cdot (l-1)^j \cdot z^{l-1} + (1-\alpha)\cdot k \cdot (k-1)^j \cdot z^{k-1} \\ Z_j(z,l,k) &= l \cdot (l-1)^j \cdot z^{l-1} - k \cdot (k-1)^j \cdot z^{k-1}\,. \end{aligned}$$

The first partial derivatives of $T(z,\alpha)$ are

$$\frac{\partial T(z,\alpha)}{\partial z} = \frac{1}{1-z} \cdot \frac{1}{D_0(z,\alpha)} + \frac{\ln(1-z)}{z} \cdot \frac{D_1(z,\alpha)}{D_0(z,\alpha)^2} \tag{4.2}$$

$$\frac{\partial T(z,\alpha)}{\partial \alpha} = \frac{\ln(1-z)\cdot Z_0(z)}{D_0(z,\alpha)^2}\,. \tag{4.3}$$

[13] We can exclude the case $\alpha = 0$, since if $3 \leqslant d_0 < d_1$, then it holds that $\check{c}(d_0) > \check{c}(d_1)$, see Section 4.1.1.3.

The second partial derivatives of $T(z,\alpha)$ are

$$\frac{\partial^2 T(z,\alpha)}{(\partial z)^2} = \frac{1}{(1-z)^2}\cdot\frac{1}{D_0(z,\alpha)} - \frac{2}{z\cdot(1-z)}\cdot\frac{D_1(z,\alpha)}{D_0(z,\alpha)^2} + \frac{\ln(1-z)}{z^2}\cdot\frac{D_2(z,\alpha)-D_1(z,\alpha)}{D_0(z,\alpha)^2} - \frac{2\cdot\ln(1-z)}{z^2}\cdot\frac{D_1(z,\alpha)^2}{D_0(z,\alpha)^3} \tag{4.4}$$

$$\frac{\partial^2 T(z,\alpha)}{(\partial\alpha)^2} = -\frac{2\cdot\ln(1-z)\cdot Z_0(z)^2}{D_0(z,\alpha)^3} \tag{4.5}$$

$$\frac{\partial}{\partial z}\left(\frac{\partial T(z,\alpha)}{\partial\alpha}\right) = -\frac{1}{1-z}\cdot\frac{Z_0(z)}{D_0(z,\alpha)^2} + \frac{\ln(1-z)}{z}\cdot\frac{Z_1(z)}{D_0(z,\alpha)^2} - \frac{2\cdot\ln(1-z)}{z}\cdot\frac{Z_0(z)\cdot D_1(z,\alpha)}{D_0(z,\alpha)^3}\,. \tag{4.6}$$

4.5.2.2. Auxiliary Functions

Our analysis is heavily based on three functions

$$f(z) = \frac{-\ln(1-z)\cdot(1-z)}{z} \tag{4.7}$$

$$g(z,l,k) = f(z)\cdot(k-1)\cdot(l-1) + \frac{1}{1-z} + 2 - k - l \tag{4.8}$$

$$h(z,l,k) = \frac{l\cdot z^{l-k} - k - f(z)\cdot\big(l\cdot(l-1)\cdot z^{l-k} - k\cdot(k-1)\big)}{k\cdot\big((k-1)\cdot f(z) - 1\big)} \tag{4.9}$$

$$= \frac{Z_0(z,l,k) - f(z)\cdot Z_1(z,l,k)}{k\cdot z^{k-1}\cdot\big((k-1)\cdot f(z)-1\big)}\,, \tag{4.10}$$

which are shown in Figure 4.5.2 for selected parameters l and k. Furthermore, we define some "special" points

$$z_{in} = \left(\frac{l}{k}\right)^{\frac{1}{k-l}} \qquad \text{(indicator point)}$$

$$z_{le} = f^{-1}\left(\frac{1}{l-1}\right) \qquad \text{(left boundary)}$$

$$z_{ri} = f^{-1}\left(\frac{1}{k-1}\right) \qquad \text{(right boundary)}$$

$$\check{z}_0 = \min\{z\in(0,1) \mid g(z)=0\} \qquad \hat{z}_0 = \max\{z\in(0,1) \mid g(z)=0\}\,.$$

Our line of argument will rely on essential properties of f, g, h as well as $z_{le}, z_{ri}, \check{z}_0, \hat{z}_0$ and z_{in}. Proving these properties is standard calculus but unfortunately lengthy. Therefore the proofs of the next three lemmas are moved to the end of Section 4.5. We start with properties of the three auxiliary functions.

LEMMA 4.5.1 (PROPERTIES OF $f(z)$). Let $z \in (0,1)$, then it holds:

(i) $f(z) > 1 - z > 0$.

(ii) $\lim_{z\to 0} f(z) = 1$.

(iii) $\lim_{z\to 1} f(z) = 0$.

(iv) $f(z)$ is strictly decreasing.

(v) $f(z)$ is concave.

(vi) $f(z_{\text{in}}) > f(z_{\text{ri}}) = \frac{1}{k-1}$.

The proof of Lemma 4.5.1 is given in Section 4.5.5.1. A plot of $f(z)$ is shown in Figure 4.5.2 (a).

LEMMA 4.5.2 (PROPERTIES OF $g(z)$). Let $3 \leqslant l < k$ and let $z \in (0,1)$, then it holds:

(i) $g(z)$ is strictly decreasing, reaches a global minimum and is then strictly increasing. The global minimum point is the only point where $\frac{\partial g(z)}{\partial z} = 0$.

(ii) $\forall z \in (0, z_{\text{le}}]$: $g(z) > 0$.

(iii) $\forall z \in [z_{\text{ri}}, 1)$: $g(z) > 0$.

(iv) If $\min g(z) < 0$, then $g(z)$ has exactly two distinct roots, $\check{z}_0$ and $\hat{z}_0$, where $\check{z}_0 < \hat{z}_0$ and $\check{z}_0, \hat{z}_0 \in (z_{\text{le}}, z_{\text{ri}})$.

(v) Let $z > z_{\text{le}}$, then it holds $g(z,l,k) > g(z,l,k+1)$.

(vi) For fixed l there is a threshold k', $k' \geqslant l+1$, such that the following holds: If $k < k'$, then $\min_z g(z,k) \geqslant 0$; and if $k \geqslant k'$, then we have $\min_z g(z,k) < 0$.

The proof of Lemma 4.5.2 is given in Section 4.5.5.2. Example plots of $g(z,l,k)$ are shown in Figure 4.5.2 (b).

LEMMA 4.5.3 (PROPERTIES OF $h(z)$). Let $3 \leqslant l < k$ and let $z \in (0,1)$, then it holds:

(i) $h(z)$ has a pole at $z = z_{\text{ri}}$.

(ii) $\lim_{z\to 0} h(z) = -\infty$.

(iii) $\lim_{z\to z_{\text{ri}}} h(z) = +\infty$.

(iv) $\forall z \in (0, z_{\text{le}}]$: $h(z) \in (-\infty, 1]$.

(v) $\forall z \in (z_{\text{le}}, z_{\text{ri}})$: $h(z) \in (1, +\infty)$, and $h(z_{\text{le}}) = 1$.

(vi) $\forall z \in (z_{\text{ri}}, 1)$: $h(z) \in (-\infty, 1)$.

(vii) $\frac{\partial h(z)}{\partial z} > 0 \Leftrightarrow g(z) > 0$.

(viii) $h(z)$ is strictly increasing in $z \in (0, z_{\text{le}}]$.

(ix) $h(z)$ is strictly increasing in $z \in (z_{\text{ri}}, 1)$.

(x) If $\min_z g(z) \geqslant 0$, then $h(z)$ is strictly increasing in $z \in [z_{le}, z_{ri})$.

(xi) If $\min_z g(z) < 0$, then $h(z)$ is strictly increasing to a local maximum at $\check{z}_0$, then strictly decreasing to a local minimum at $\hat{z}_0$, then strictly increasing afterwards.

The proof of Lemma 4.5.3 is given in Section 4.5.5.3. Example plots of $h(z, l, k)$ are shown in Figure 4.5.2 (c).

Concerning the defined special points, we are only interested in how they are related to each other.

LEMMA 4.5.4 (PROPERTIES OF SPECIAL POINTS). Let $3 \leqslant l < k$ and let $z \in (0, 1)$, then it holds:

(i) $0 < z_{le} < z_{ri} < 1$.

(ii) If $\check{z}_0$ and $\hat{z}_0$ exist (not necessarily distinct), then $z_{le} < \check{z}_0$ and $\hat{z}_0 < z_{ri}$.

(iii) $z_{in} \in (0, z_{ri})$.

PROOF. Consider the properties of $f(z)$ and $g(z)$.

ad (i): This follows by Lemma 4.5.1 (ii), (iii), and (iv).

ad (ii): This follows from Lemma 4.5.2 (iv).

ad (iii): This follows by Lemma 4.5.1 (vi). ■

All of the points but z_{in} are visualized in Figure 4.5.2. Now we are prepared for solving the optimization problem.

4.5.3. ANALYSIS

Assume first that α is arbitrary but fixed, i. e., we are looking for a global minimum of (4.1) in z-direction. Since

$$\lim_{z \to 0} T(z) = \lim_{z \to 1} T(z) = +\infty\,, \tag{4.11}$$

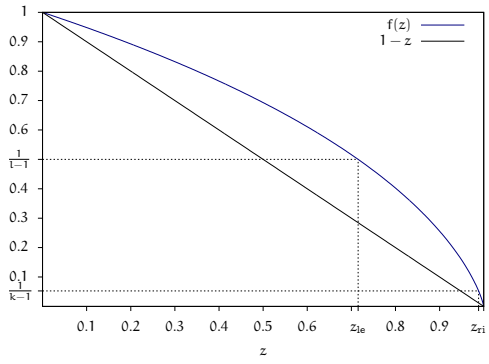
and $T(z)$ is continuous for $z \in (0, 1)$, a global minimum must be a point where the first derivative of $T(z)$ is zero, i. e., a critical point. According to (4.2) critical points in z-direction for *unbounded* α, i. e., $\alpha \in \mathbb{R}$, can be described via

$$\frac{\partial T(z)}{\partial z} = 0 \Leftrightarrow \frac{D_0(z)}{D_1(z)} = f(z) \tag{4.12}$$

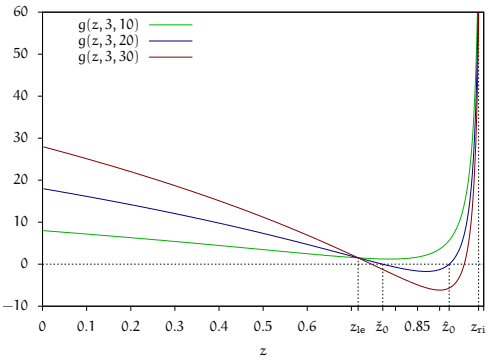
$$\Leftrightarrow \qquad \alpha = \frac{k \cdot z^{k-1} \cdot \big((k-1) \cdot f(z) - 1\big)}{Z_0(z) - f(z) \cdot Z_1(z)} \tag{4.13}$$

$$\Leftrightarrow \qquad \alpha = \frac{1}{h(z)}\,. \tag{4.14}$$

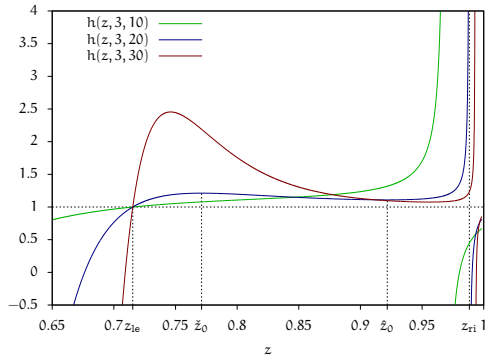
The next lemma identifies and classifies critical points of $T(z)$ for *bounded* α that is for $\alpha \in (0, 1]$.



(a) Function $f(z)$.



(b) Function $g(z, l, k)$.



(c) Function $h(z, l, k)$.

Figure 4.5.2.: Auxiliary functions $f(z)$, $g(z, l, k)$, and $h(z, l, k)$ for parameters $l = 3$ and $k = 10, 20, 30$; as well as special points $\check{z}_0$, $\hat{z}_0$, z_{le}, and z_{ri} for $l = 3$ and $k = 20$.

LEMMA 4.5.5 (CRITICAL POINTS). Let $\alpha \in (0,1]$ be arbitrary but fixed. If $\frac{\partial T}{\partial z}(\tilde{z}) = 0$ for some $\tilde{z} \in (0,1)$, then the following holds:

(i) $\tilde{z} \in [z_{le}, z_{ri})$.

(ii) If $g(\tilde{z}) > 0$, then $T(\tilde{z})$ is a local minimum.

(iii) If $g(\tilde{z}) < 0$, then $T(\tilde{z})$ is a local maximum.

PROOF. ad (i): According to (4.14), we have that $\alpha = 1/h(\tilde{z})$. Hence, with $\alpha \in (0,1]$, we get $h(\tilde{z}) \in (1, +\infty)$. By Lemma 4.5.3 (iv), (v), (vi), it follows that $\tilde{z} \in [z_{le}, z_{ri})$.

ad (ii): Consider the second derivative of $T(z)$ with respect to z. According to (4.4), we have

$$\frac{\partial^2 T(z)}{(\partial z)^2} > 0 \Leftrightarrow \frac{1}{(1-z)^2} - \frac{2}{z\cdot(1-z)} \cdot \frac{D_1(z)}{D_0(z)} + \frac{\ln(1-z)}{z^2} \cdot \frac{D_2(z) - D_1(z)}{D_0(z)} - \frac{2\cdot\ln(1-z)}{z^2} \cdot \frac{D_1(z)^2}{D_0(z)^2} > 0\,.$$

Assume that $\tilde{z} \in [z_{le}, z_{ri})$ is a critical point.

! For the rest of the proof let $z = \tilde{z}$.

Utilizing that $D_0(z)/D_1(z) = f(z)$ (4.12), it follows that

$$\begin{aligned}\frac{\partial^2 T(z)}{(\partial z)^2} > 0 \Leftrightarrow \frac{1}{(1-z)^2} - \frac{2}{z\cdot(1-z)\cdot f(z)} + \frac{\ln(1-z)}{z^2} \cdot \frac{D_2(z)}{D_0(z)} \\ - \frac{\ln(1-z)}{z^2\cdot f(z)} - \frac{2\cdot\ln(1-z)}{z^2\cdot f(z)^2} > 0 \\ \Leftrightarrow \qquad \frac{1}{(1-z)^2} - \frac{f(z)}{z\cdot(1-z)} \cdot \frac{D_2(z)}{D_0(z)} + \frac{1}{z(1-z)} > 0\,.\end{aligned}$$

Hence, we obtain

$$\frac{\partial^2 T(z)}{(\partial z)^2} > 0 \Leftrightarrow \frac{D_0(z)}{D_2(z)} > f(z)\cdot(1-z) \Leftrightarrow \frac{D_1(z)}{D_2(z)} > (1-z)\,.$$

Factoring out α from $D_1(z)/D_2(z) > (1-z)$ gives that $D_1(z) > (1-z)\cdot D_2(z)$ is equivalent to

$$\alpha \cdot \big(Z_1(z) - (1-z)\cdot Z_2(z)\big) > -k\cdot(k-1)\cdot z^{k-1} + (1-z)\cdot k\cdot(k-1)^2\cdot z^{k-1}\,.$$

Substituting α according to (4.13) and dividing by $k\cdot z^{k-1}$ leads to

$$\frac{\partial^2 T(z)}{(\partial z)^2} > 0 \Leftrightarrow \frac{(k-1)\cdot f(z) - 1}{Z_0(z) - f(z)\cdot Z_1(z)} \cdot \big(Z_1(z) - (1-z)\cdot Z_2(z)\big) > - (k-1) + (1-z)\cdot(k-1)^2\,.$$

Consider (4.13). Given $\alpha \in (0,1]$ and $z < z_{ri}$, we have that $(k-1)\cdot f(z) - 1 > 0$, according to the definition of z_{ri} and Lemma 4.5.1 (iv). Hence, the numerator of (4.13) is positive. Since $\alpha > 0$, it follows that $Z_0(z) - f(z)\cdot Z_1(z) > 0$, i. e., the denominator is positive too. Therefore, we get

$$\begin{aligned}
\frac{\partial^2 T(z)}{(\partial z)^2} > 0 \Leftrightarrow \quad & \big((k-1)\cdot f(z) - 1\big)\cdot\big(Z_1(z) - (1-z)\cdot Z_2(z)\big) > \\
& \big((k-1)^2\cdot(1-z) - (k-1)\big)\cdot\big(Z_0(z) - f(z)\cdot Z_1(z)\big) \\
\Leftrightarrow \quad & Z_2(z)\cdot\big(1 - z - (1-z)\cdot(k-1)\cdot f(z)\big) \\
& + Z_1(z)\cdot\big((k-1)^2\cdot(1-z)\cdot f(z) - 1\big) > \\
& Z_0(z)\cdot\big((1-z)\cdot(k-1)^2 - (k-1)\big) \\
\Leftrightarrow \quad & (l-1)^2\cdot\big(1 - z - (1-z)\cdot(k-1)\cdot f(z)\big) \\
& + (l-1)\cdot\big((k-1)^2\cdot(1-z)\cdot f(z) - 1\big) > \\
& (1-z)\cdot(k-1)^2 - (k-1)\,.
\end{aligned}$$

Factoring out $f(z)\cdot(k-1)\cdot(l-1)$ gives

$$\begin{aligned}
\frac{\partial^2 T(z)}{(\partial z)^2} > 0 &\Leftrightarrow f(z)\cdot(k-1)\cdot(l-1)\cdot(k-l) > (k-1)^2 - (l-1)^2 - \frac{k-l}{1-z} \\
&\Leftrightarrow f(z)\cdot(k-1)\cdot(l-1) + \frac{1}{1-z} + 2 - k - l > 0 \\
&\overset{\text{def}}{\Leftrightarrow} g(z) > 0\,.
\end{aligned}$$

ad (iii): This case is analogous to case (ii).
This finishes the proof of the lemma. ■

The next lemma can be seen as the central building block for understanding the behavior of the threshold function. Using the function $g(z)$ we decide how many and which kind of extremal points $T(z)$ has.

LEMMA 4.5.6 (CLASSIFYING EXTREMA). Let $\alpha \in (0,1]$ be arbitrary but fixed.

1. Let $\min_z g(z) \geqslant 0$, then the function $T(z)$ has exactly one critical point $\tilde{z}$, and $\tilde{z} \in [z_{le}, z_{ri})$ is a *global minimum* point.
2. Let $\min_z g(z) < 0$, then there are four pairwise distinct points

   $$z_{l0} < \check{z}_0 < \hat{z}_0 < z_{0r}$$

   from the interval $[z_{le}, z_{ri})$, such that the following holds:
   (i) For all α with $1/\alpha \in [1, h(\hat{z}_0))$ the function $T(z)$ has exactly one critical point $\tilde{z}$, and $\tilde{z} \in (z_{le}, z_{l0})$ is a *global minimum* point.

(ii) For α with $1/\alpha = h(\hat{z}_0)$ the function $T(z)$ has exactly two distinct critical points $\tilde{z}_1 < \tilde{z}_2$, where $\tilde{z}_1 = z_{l0}$ is a *global minimum* point, and $\tilde{z}_2 = \hat{z}_0$ is an *inflection* point.

(iii) For all α with $1/\alpha \in (h(\hat{z}_0), h(\check{z}_0))$ the function $T(z)$ has exactly three pairwise distinct critical points $\tilde{z}_1 < \tilde{z}_3 < \tilde{z}_2$, where $\tilde{z}_1, \tilde{z}_2$ are *local minimum* points, and $\tilde{z}_3$ is a *local maximum* point.

(iv) For α with $1/\alpha = h(\check{z}_0)$ the function $T(z)$ has exactly two distinct critical points $\tilde{z}_1 < \tilde{z}_2$, where $\tilde{z}_1 = \check{z}_0$ is an *inflection* point, and $\tilde{z}_2 = z_{0r}$ is an *global minimum* point.

(v) For all α with $1/\alpha \in (h(\check{z}_0), \infty)$ the function $T(z)$ has exactly one critical point $\tilde{z}$, and $\tilde{z} \in (z_{0r}, z_{ri})$ is a *global minimum* point.

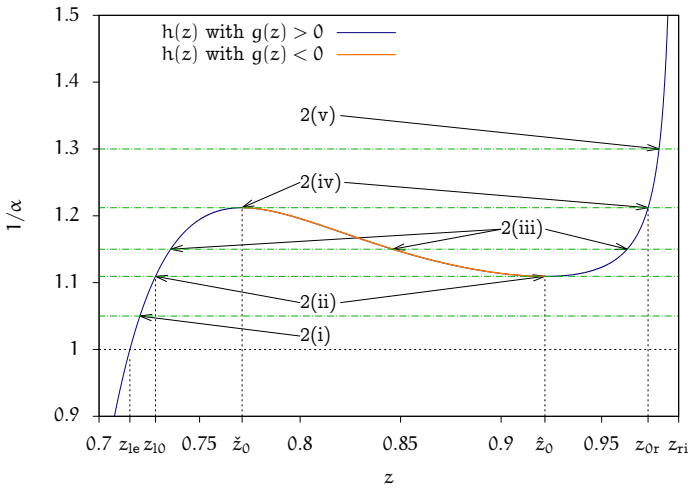


Figure 4.5.3.: Function $h(z)$ for $l = 3$, $k = 20$ visualizing case 2 of Lemma 4.5.6. The intersection points between the function $1/\alpha$ (horizontal lines) and the function $h(z)$ are the extrema of $T(z)$. They are classified depending on the part of $h(z)$ where the intersection takes place.

PROOF. ad 1.: By Lemma 4.5.5 (i), it follows that all critical points $\tilde{z}$ must be from $[z_{le}, z_{ri})$. Consider the function $h(z)$. According to Lemma 4.5.3 (v) and (x), it holds that for each x from $[1, +\infty)$ there is exactly one z from $[z_{le}, z_{ri})$ such that $h(z) = x$. Furthermore, according to (4.14) we have

$$\frac{\partial T(z)}{\partial z} = 0 \Leftrightarrow \alpha = 1/h(z)\,.$$

It follows that for each $\alpha \in (0, 1]$ there is exactly one $z = \tilde{z}$ with $\alpha = 1/h(\tilde{z})$.

Furthermore, $\min_z g(z) \geqslant 0$ implies that $g(\tilde{z}) \geqslant 0$. Since $\tilde{z}$ is the only critical point of $T(z)$, it follows with (4.11) that $\tilde{z}$ must be a global minimum point.

ad 2.: Figure 4.5.3 illustrates the complete case 2 of Lemma 4.5.6. We know from Lemma 4.5.3 (xi) that for $z \in [z_{le}, z_{ri})$ the function $h(z)$ is strictly increasing, reaches a local maximum at $\check{z}_0$, is strictly decreasing, reaches a local minimum at $\hat{z}_0$, and is strictly increasing to $+\infty$ afterwards. According to the definition of $\check{z}_0$ and $\hat{z}_0$ and by Lemma 4.5.3 (vii), we have that $g(z) > 0$ for $z \in [z_{le}, \check{z}_0)$, $g(z) = 0$ for $z = \check{z}_0$, $g(z) < 0$ for $z \in (\check{z}_0, \hat{z}_0)$, $g(z) = 0$ for $z = \hat{z}_0$, and $g(z) > 0$ for $z \in (\hat{z}_0, z_{ri})$, Now, consider the condition (4.14).

ad (i): For all α with $1/\alpha \in [1, h(\hat{z}_0))$ there is, according to Lemma 4.5.3 (xi), exactly one z with $1/\alpha = h(z)$. In addition we have that $z \in [z_{le}, \check{z}_0)$, and therefore $g(z) > 0$, see Lemma 4.5.2 (i) and (ii). Hence, the claim follows by Lemma 4.5.5 (ii).

ad (ii): Let $1/\alpha = h(\hat{z}_0)$ and let $\tilde{z}_2 = \hat{z}_0$. According to Lemma 4.5.3 there is exactly one other point $\tilde{z}_1$, such that $\alpha = 1/h(\tilde{z}_1)$. Furthermore, it holds $g(\tilde{z}_1) > 0$ and $g(\tilde{z}_2) = 0$. According to Lemma 4.5.5 (ii), the point $\tilde{z}_1$ must be a local minimum point. Because of the monotonicity of $T(z)$ (4.11) the other critical point must be an inflection point. Hence, $\tilde{z}_1$ is also a global minimum point.

ad (iii): According to Lemma 4.5.3, there are exactly 3 different points $\tilde{z}_i$, $1 \leqslant i \leqslant 3$, such that $1/\alpha = h(\tilde{z}_i)$ and $\frac{\partial T}{\partial z}(\tilde{z}_i) = 0$, respectively. Furthermore, it holds $\tilde{z}_1 < \check{z}_0 < \tilde{z}_3 < \hat{z}_0 < \tilde{z}_2$ and $g(\tilde{z}_1) > 0, g(\tilde{z}_2) > 0, g(\tilde{z}_3) < 0$. By Lemma 4.5.5 (ii) and (iii), it follows that $\tilde{z}_1$ and $\tilde{z}_2$ are local minimum points of $T(z)$ and $\tilde{z}_3$ is a local maximum point of $T(z)$.

ad (iv): The case $1/\alpha = 1/h(\check{z}_0)$ is analogous to case 2(ii).

ad (v): The case $1/\alpha \in (h(\check{z}_0), \infty)$ is analogous to case 2(i).

This finishes the proof of the lemma. ■

The last lemma gives a complete characterization of the local extrema of the threshold function (4.1) in z-direction including the global minimum for arbitrary but fixed α. It remains to find a value α^* that maximizes the threshold function at the corresponding global minimum in z-direction. So the point we are looking for could be a saddle point of $T(z, \alpha)$. Indeed the following lemma shows that $T(z, \alpha)$ has exactly one saddle point for *unbounded* α, i. e., $\alpha \in \mathbb{R}$, and the following Theorem 4.5 finally shows under which conditions this point is the optimum we are looking for.

> Lemma 4.5.7 (Saddle Point). Let $\alpha \in \mathbb{R}$. Then $T(z, \alpha)$ has exactly one saddle point $(\tilde{z}, \tilde{\alpha})$, where
>
> $$(\tilde{z}, \tilde{\alpha}) = \left(\left(\frac{l}{k}\right)^{\frac{1}{k-l}}, \frac{k-1}{k-l} - \frac{1}{f(\tilde{z}) \cdot (k-l)} \right).$$

PROOF. Solving the linear system $\left\{\frac{\partial T(z,\alpha)}{\partial z} = 0, \frac{\partial T(z,\alpha)}{\partial \alpha} = 0\right\}$ gives

$$\begin{aligned}
&\frac{\partial T(z,\alpha)}{\partial z} = 0 \Leftrightarrow \alpha = 1/h(z)\\
&\frac{\partial T(z,\alpha)}{\partial \alpha} = 0 \Leftrightarrow \frac{\ln(1-z)\cdot Z_0(z)}{D_0(z,\alpha)^2} = 0 \Leftrightarrow Z_0(z) = 0 \Leftrightarrow l\cdot z^{l-1} = k\cdot z^{k-1}\\
&\qquad\qquad \Leftrightarrow z = \left(\frac{l}{k}\right)^{\frac{1}{k-l}} \qquad\qquad \overset{\text{def}}{\Leftrightarrow} z = z_{\text{in}}\,.
\end{aligned}$$

There is only one solution of $\frac{\partial T(z,\alpha)}{\partial \alpha} = 0$ and according to Lemma 4.5.3 (i) and Lemma 4.5.1 (vi) the function $h(z)$ is defined at z_{in}. Hence, we get a unique critical point $(\tilde{z},\tilde{\alpha})$, with $\tilde{z} = z_{\text{in}}$, and

$$\begin{aligned}
\tilde{\alpha} &= 1/h(\tilde{z})\\
&= \frac{k\cdot\left(\frac{l}{k}\right)^{\frac{k-1}{k-l}}\cdot(f(\tilde{z})\cdot(k-1)-1)}{-f(\tilde{z})\cdot\left(l^2\cdot\left(\frac{l}{k}\right)^{\frac{l-1}{k-l}} - k^2\cdot\left(\frac{l}{k}\right)^{\frac{k-1}{k-l}}\right)} = \frac{k\cdot\left(\frac{l}{k}\right)^{\frac{k-1}{k-l}}\cdot\left(f(\tilde{z})\cdot(k-1)-1\right)}{-f(\tilde{z})\cdot k^2\cdot\left(\frac{l}{k}\right)^{\frac{k-1}{k-l}}\cdot(\frac{l}{k}-1)}\\
&= \frac{f(\tilde{z})\cdot(k-1)-1}{f(\tilde{z})\cdot(k-l)} = \frac{k-1}{k-l} - \frac{1}{f(\tilde{z})\cdot(k-l)}\,.
\end{aligned}$$

In order to classify this critical point, we consider the second partial derivatives of $T(z,\alpha)$, see Section 4.5.2.1. We have $Z_0(\tilde{z}) = 0$ and $Z_1(\tilde{z}) < 0$, since

$$\begin{aligned}
Z_1(\tilde{z}) < 0 &\Leftrightarrow l\cdot(l-1)\cdot\left(\frac{l}{k}\right)^{(l-1)/(k-l)} - k\cdot(k-1)\cdot\left(\frac{l}{k}\right)^{(k-1)/(k-l)} < 0\\
&\Leftrightarrow \frac{l\cdot(l-1)}{k\cdot(k-1)} < \left(\frac{l}{k}\right)^{(k-l)/(k-l)} \Leftrightarrow \frac{l-1}{k-1} < 1\,,
\end{aligned}$$

and $3 \leqslant l < k$. It follows that $\frac{\partial^2}{(\partial\alpha)^2}T(\tilde{z},\tilde{\alpha}) = 0$ as well as $\frac{\partial^2}{\partial z\partial\alpha}T(\tilde{z},\tilde{\alpha}) > 0$. Therefore, the Hessian matrix at point $(\tilde{z},\tilde{\alpha})$, denoted by

$$\text{Hess}_T(\tilde{z},\tilde{\alpha}) = \begin{pmatrix} \frac{\partial^2}{(\partial z)^2}T(\tilde{z},\tilde{\alpha}) & \frac{\partial^2}{\partial z\partial\alpha}T(\tilde{z},\tilde{\alpha}) \\ \frac{\partial^2}{\partial z\partial\alpha}T(\tilde{z},\tilde{\alpha}) & \frac{\partial^2}{(\partial\alpha)^2}T(\tilde{z},\tilde{\alpha}) \end{pmatrix} \doteq \begin{pmatrix} = 0 & > 0 \\ > 0 & \frac{\partial^2}{(\partial z)^2}T(\tilde{z},\tilde{\alpha}) \end{pmatrix},$$

has determinant $\det\left(\text{Hess}_T(\tilde{z},\tilde{\alpha})\right) < 0$, which implies that $(\tilde{z},\tilde{\alpha})$ is a saddle point. ■

4.5.3.1. PUTTING IT ALL TOGETHER

For given l and k, each point (z^*,α^*) with the property

$$T(z^*,\alpha^*) = \max_{\alpha\in(0,1]}\min_{z\in(0,1)} T(z,\alpha)$$

is called *optimal point*. Now, we can restate Theorem 4.5 more precisely.

THEOREM 4.5 (MAXIMUM 2-CORE THRESHOLD)
Let $l, k \in \mathbb{N}$ *be fixed with* $3 \leqslant l < k$*, then the following holds:*

1. *If* $\min_z g(z) \geqslant 0$*, then we have a unique optimal point.*
   (i) *If* $h(z_{in}) \leqslant 1$*, then the optimal point is*

   $$(z^*, \alpha^*) = (z_{le}, 1)$$

   and the maximum threshold is given by

   $$T(z^*, \alpha^*) = \frac{-\ln(1 - z_{le})}{l \cdot z_{le}^{l-1}} .$$

   (ii) *If* $h(z_{in}) > 1$*, then the optimal point is the saddle point*

   $$(z^*, \alpha^*) = \left(\left(\frac{l}{k} \right)^{\frac{1}{k-l}}, \frac{k-1}{k-l} - \frac{1}{f(z^*) \cdot (k-l)} \right)$$

   and the maximum threshold is given by

   $$T(z^*, \alpha^*) = -\ln \left(1 - \left(\frac{l}{k} \right)^{\frac{1}{k-l}} \right) \cdot \left(\frac{k^{l-1}}{l^{k-1}} \right)^{\frac{1}{k-l}} .$$

2. *If* $\min_z g(z) < 0$*, then we have at most two different optimal points.*
   (i) *If* $h(z_{in}) \leqslant 1$*, then the optimum is the same as in case 1(i).*
   (ii) *If* $h(z_{in}) \in (1, h(\hat{z}_0)]$*, then the optimum is the same as in case 1(ii).*
   (iii) *If* $h(z_{in}) \in (h(\hat{z}_0), h(\check{z}_0))$*, then there are exactly two distinct optimal points* (z^*, α^*) *and* (z^{**}, α^*)*, where* $1/\alpha^* = h(z^*) = h(z^{**})$ *and* $T(z^*, \alpha^*) = T(z^{**}, \alpha^*)$.
   The two optimal points can be determined numerically using binary search for the value $\alpha = \alpha^*$*, within the interval* $[1/h(z_{up}), 1/h(z_{lo})]$*, that gives* $T(\tilde{z}_1, \alpha) = T(\tilde{z}_2, \alpha)$ *with* $h(\tilde{z}_1) = h(\tilde{z}_2) = 1/\alpha$*, for*

   $$\tilde{z}_1 \in (z_{le}, z_{up}) \text{ and } \tilde{z}_2 \in (z_{lo}, z_{ri}) .$$

   The (initial) interval borders are determined as follows:
   - ▷ *If* $z_{in} < \check{z}_0$*, then* $z_{up} = z_{in}$ *and* $z_{lo} = \hat{z}_0$.
   - ▷ *If* $\check{z}_0 < z_{in} < \hat{z}_0$*, then* $z_{up} = \check{z}_0$ *and* $z_{lo} = \hat{z}_0$.
   - ▷ *If* $z_{in} > \hat{z}_0$*, then* $z_{up} = \check{z}_0$ *and* $z_{lo} = z_{in}$.

   (iv) *If* $h(z_{in}) \in [h(\check{z}_0), \infty)$*, then the optimum is the same as in case 1(ii).*

PROOF. Using (4.14) we can define a *function of critical points* $\tilde{T}(z)$ of $T(z, \alpha)$ as

follows

$$\begin{aligned}\tilde{T}(z) \coloneqq T(z, 1/h(z)) &= \frac{-\ln(1-z)}{1/h(z) \cdot Z_0(z) + k \cdot z^{k-1}} \\ &= \frac{\frac{z}{1-z} \cdot Z_0(z) + \ln(1-z) \cdot Z_1(z)}{k \cdot l \cdot (k-l) \cdot z^{k+l-2}} .\end{aligned}$$

The first derivative of $\tilde{T}(z)$ is

$$\begin{aligned}\frac{\partial \tilde{T}(z)}{\partial z} = &\frac{1}{k \cdot l \cdot (k-l) \cdot z^{k+l-2}} \cdot \left(\frac{Z_0(z)}{(1-z)^2} + \frac{\ln(1-z) \cdot Z_2(z)}{z} \right) \\ &- \frac{k+l-2}{k \cdot l \cdot (k-l) \cdot z^{k+l-2}} \cdot \left(\frac{Z_0(z)}{1-z} + \frac{\ln(1-z) \cdot Z_1(z)}{z} \right) .\end{aligned}$$

We are interested in the monotonicity of $\tilde{T}(z)$.

$$\begin{aligned}&\frac{\partial \tilde{T}(z)}{\partial z} \overset{!}{>} 0 \\ &\Leftrightarrow \frac{Z_0(z)}{(1-z)^2} + \frac{\ln(1-z) \cdot Z_2(z)}{z} - (k+l-2) \cdot \left(\frac{Z_0(z)}{1-z} + \frac{\ln(1-z) \cdot Z_1(z)}{z} \right) > 0 \\ &\Leftrightarrow \frac{Z_0(z)}{1-z} - (k+l-2) \cdot Z_0(z) - f(z) \cdot (Z_2(z) - (k+l-2) \cdot Z_1(z)) > 0 \\ &\Leftrightarrow \frac{Z_0(z)}{1-z} - (k+l-2) \cdot Z_0(z) + f(z) \cdot Z_0(z) \cdot (k-1) \cdot (l-1) > 0 \\ &\Leftrightarrow Z_0(z) \cdot g(z) > 0 .\end{aligned}$$

We have that $Z_0(z) > 0 \Leftrightarrow z < \left(\frac{l}{k}\right)^{\frac{1}{k-l}} = z_{in}$. By division of $Z_0(z)$ for $z \neq z_{in}$, we get

$$\begin{aligned}&\forall z < z_{in}\colon \frac{\partial \tilde{T}(z)}{\partial z} > 0 \Leftrightarrow g(z) > 0 \\ &\forall z > z_{in}\colon \frac{\partial \tilde{T}(z)}{\partial z} < 0 \Leftrightarrow g(z) > 0 .\end{aligned} \tag{4.15}$$

ad 1.: If $\min_z g(z) > 0$, then according to (4.15) the function of critical points has a global maximum in α-direction at z_{in}. Now consider the special case $\min_z g(z) = 0$ and let $z_{min} = \arg\min_z g(z)$. (Note that according to Lemma 4.5.2 (i) and the definition of $\check{z}_0$ and $\hat{z}_0$ we have $\check{z}_0 = \hat{z}_0 = z_{min}$.) If $z_{min} \neq z_{in}$, then z_{min} must be an inflection point of $\tilde{T}(z)$ since before and after z_{min} the monotonicity is the same. Hence, the function of critical points has a global maximum in α-direction at z_{in} also in this case.

ad (i): If $h(z_{in}) \leqslant 1$, then z_{in} must be from the interval $(0, z_{le}]$ (Lemma 4.5.3 (iv)) and not from the interval $(z_{ri}, 1)$ (Lemma 4.5.3 (vi)), since we have $f(z_{in}) > f(z_{ri})$ (Lemma 4.5.1 (vi)), and $f(z)$ is monotonically decreasing (Lemma 4.5.1 (iv)). However, if $z_{in} \leqslant z_{le}$, then because of the monotonicity of $\tilde{T}(z)$ (4.15) the optimal z value is the nearest feasible critical point. That is, the optimal point is the (degenerated) solution $(z_{le}, 1)$.

ad (ii): If $h(z_{in}) > 1$, then according to Lemma 4.5.3 (v) we have $z_{in} \in (z_{le}, z_{ri})$. It follows from Lemma 4.5.6 1 that $\tilde{T}(z_{in})$ is a global minimum in z-direction. Hence, $(z_{in}, 1/h(z_{in}))$ is the optimum point, which is according to Lemma 4.5.7 the saddle point.

ad 2.: Since $\min_z g(z) < 0$, it follows from Lemma 4.5.3 (xi) that for $z \in [z_{le}, z_{ri})$ the function $h(z)$ is strictly increasing, reaches a maximum at $\check{z}_0$, is strictly decreasing, reaches a minimum at $\hat{z}_0$, and is strictly increasing afterwards. Furthermore we have $g(z) > 0$ for $z \in [z_{le}, \check{z}_0)$, $g(z) < 0$ for $z \in (\check{z}_0, \hat{z}_0)$, $g(z) > 0$ for $z \in (\hat{z}_0, z_{ri})$, and $g(z) = 0$ for $z = \check{z}_0$ and $z = \hat{z}_0$. An optimal z must be a global minimum point of $T(z, \alpha)$ in z-direction. According to Lemma 4.5.5 (ii) and case 2 of Lemma 4.5.6 global minimum points are points from $[z_{le}, \check{z}_0) \cup (\hat{z}_0, z_{ri})$.

ad (i): If $h(z_{in}) \leqslant 1$, then an optimal z cannot be from $(\hat{z}_0, z_{ri})$, since for each $z \in (\hat{z}_0, z_{ri})$ there is an $\varepsilon > 0$ such that $z - \varepsilon \in (\hat{z}_0, z_{ri})$ and $\tilde{T}(z - \varepsilon) > \tilde{T}(z)$. This converges to $\hat{z}_0$, but according to case 2 of Lemma 4.5.6 the point $\hat{z}_0$ is an inflection point and not a global minimum point. Hence the optimal z must be from $[z_{le}, \check{z}_0)$. For each $z \in (z_{le}, \check{z}_0)$ there is an $\varepsilon > 0$ such that $z - \varepsilon \in [z_{le}, \check{z}_0)$ and $\tilde{T}(z - \varepsilon) > \tilde{T}(z)$. This converges to z_{le}.

ad (ii): If $h(z_{in}) \in (1, h(\hat{z}_0)]$, then it follows that $z_{in} \in (z_{le}, z_{l0})$ or $z_{in} = \hat{z}_0$. An optimal z cannot be from $[\hat{z}_0, z_{ri})$ for the same reasons as in case 2(i) And if $z_{in} = \hat{z}_0$, then there would be no optimal z at all, see Lemma 4.5.6 2. Hence, we have $z_{in} \in (z_{le}, z_{l0})$ and the optimum is the same as in 1(ii).

ad (iii): Let $h(z_{in}) \in (h(\hat{z}_0), h(\check{z}_0))$ and consider an arbitrary but fixed α with $1/\alpha \in (h(\hat{z}_0), h(\check{z}_0))$. According to Lemma 4.5.6 2(iii) we have two different points $\tilde{z}_1$ and $\tilde{z}_2$, with $\tilde{z}_1 < \check{z}_0 < \hat{z}_0 < \tilde{z}_2$, that are local minimum points of the threshold function $T(z, \alpha)$ in z-direction.

- ▷ Let $\check{z}_0 < z_{in} < \hat{z}_0$. Decreasing α (increasing $1/\alpha$) by an arbitrary small but fixed positive value gives two new local minimum points in z-direction, $\tilde{z}_1 + \varepsilon$, $\tilde{z}_2 + \delta$, where $\varepsilon, \delta > 0$. According to (4.15) it holds that $\tilde{T}(\tilde{z}_1) < \tilde{T}(\tilde{z}_1 + \varepsilon)$ and $\tilde{T}(\tilde{z}_2) > \tilde{T}(\tilde{z}_2 + \delta)$. Hence, for the left critical point the local minimum in z-direction becomes smaller while the potential threshold becomes larger and for the right critical point the local minimum in z-direction becomes larger while the potential threshold becomes smaller. Increasing α by an arbitrary small but fixed positive value reverses the behavior. Assume we have found an optimal $\alpha = \alpha^*$. Decreasing α by some small fixed positive value increases the threshold for the left critical point, but because of the optimality of α we have no global minimum for the left critical point but only a local minimum. Increasing α increases the threshold for the right critical point but because of the optimality of α we have no global minimum for the right critical point but only a local minimum. Hence, for $\alpha = \alpha^*$ exactly two different critical points z^* and z^{**}, with $1/\alpha^* = h(z^*) = h(z^{**})$, lead to the same minimum in z-direction, i. e., both local

minimum points are also global minimum points and $T(z^*, \alpha^*) = T(z^{**}, \alpha^*)$ is the optimal threshold.

- ▷ Let $z_{in} < \check{z}_0$. Assume that $1/\alpha \in (h(z_{in}), h(\check{z}_0))$, then α cannot be optimal since increasing α by an arbitrary small but fixed positive value increases $\tilde{T}(\tilde{z}_1)$ as well as $\tilde{T}(\tilde{z}_2)$ and one of the critical points must be the global minimum point in z-direction. Hence the optimum $1/\alpha$ must be in the interval $(h(\hat{z}_0), h(z_{in})]$.
- ▷ The case $z_{in} > \hat{z}_0$ is handled analogously to the case $z_{in} < \check{z}_0$.

ad (iv): If $h(z_{in}) \in [h(\check{z}_0), \infty)$, then we have $z_{in} \in [z_{0r}, z_{ri})$ or $z_{in} = \check{z}_0$. Analogously to the proof of case 2(ii), we conclude that neither an optimal z is from $[z_{le}, \check{z}_0)$ nor $z_{in} = \check{z}_0$. Hence, the optimum is the same as in case 1(ii). ■

The distinction between case 1 and case 2 of Theorem 4.5 can be done via solving $\frac{\partial g(z)}{\partial z} = 0$ for $z \in (0, 1)$, since the function $g(z)$ has only one critical point and this point is a global minimum point, see Lemma 4.5.2 (i). Hence, the theorem is easily transferred into an algorithm that determines α^*, z^*, and $T(z^*, \alpha^*)$ for given $\mathbf{d} = (l, k)$, as carried out in Algorithm 15.

Optimal thresholds $T(z^*, l, k, \alpha^*) = \check{c}(\mathbf{d})$ for selected $\mathbf{d} = (l, k)$ are given in Table 4.1.2 and Appendix A.2. They show that the optimal 2-core threshold of some non-uniform hypergraphs $H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ can be above the 2-core threshold for 3-uniform hypergraphs.

If one wants to determine the optimal points for fixed l and increasing k, one can make use of the following observation. According to Lemma 4.5.2 (vi) there is a value k', such that for $l < k < k'$ it holds that $\min_z g(z, k) \geqslant 0$, and for $k \geqslant k'$ it holds that $g(z, k) < 0$. That is, after reaching k' we don't need to further calculate the minimum of $g(z)$. Table 4.5.3 lists some values for k'.

l	3	4	5	6	7	8	9	10
k'	16	29	45	62	79	98	117	137

Table 4.5.3.: Values $k'(l)$ where $g(z, k)$ switches from non-negative to negative.

4.5.4. Experiments

To underpin our theoretical results we experimentally approximated the point $\check{c}(\mathbf{d})$ of the phase transition from empty to non-empty 2-core in *pseudorandom* hypergraphs $H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ of type B.

Setup and Measurements For each vector $\mathbf{d} \in \{(3, 4), (3, 8), (3, 16), (3, 21)\}$ and the corresponding optimal fractions of edge sizes $\boldsymbol{\alpha}^* = (\alpha^*, 1 - \alpha^*)$ we performed

ALGORITHM 15: `maximum_thresholds`

Input : Integers l, k with $3 \leqslant l < k$ and $\varepsilon > 0$.
Purpose : Solve the optimization problem stated in Section 4.5.1. The resulting threshold is at most ε below the optimal threshold.
Require : Function `numSolve(`*eq*`,`*in*`)` that returns numerical solution of equation *eq* within the given interval *in*.

Initialization:

$z_{le} \leftarrow \texttt{numSolve}(f(z) = \frac{1}{l-1}, z \in (0,1))$; $z_{ri} \leftarrow \texttt{numSolve}(f(z) = \frac{1}{k-1}, z \in (z_{le}, 1))$;
$z_{min} \leftarrow \texttt{numSolve}(\frac{\partial g(z)}{\partial z} = 0, z \in (0,1))$; $z_{in} \leftarrow \left(\frac{l}{k}\right)^{\frac{1}{k-l}}$;
$\check{z}_0 \leftarrow z_{in}$; $\hat{z}_0 \leftarrow z_{in}$;
if $g(z_{min}, l, k) < 0$ then
  $\check{z}_0 \leftarrow \texttt{numSolve}(g(z,l,k) = 0, z \in (z_{le}, z_{min}))$;
  $\hat{z}_0 \leftarrow \texttt{numSolve}(g(z,l,k) = 0, z \in (z_{min}, z_{ri}))$;
end
Optimization:
if $h(z_{in}, l, k) \leqslant 1$ then
  $z^* \leftarrow z_{le}$; $\alpha^* \leftarrow 1$; $T^* \leftarrow \frac{-\ln(1-z_{le})}{l \cdot z_{le}^{l-1}}$;
else
  if $h(z_{in}, l, k) \leqslant h(\hat{z}_0, l, k)$ or $h(z_{in}, l, k) \geqslant h(\check{z}_0, l, k)$ then
    $z^* \leftarrow z_{in}$; $\alpha^* \leftarrow \frac{k-1}{k-l} - \frac{1}{f(z^*)\cdot(k-l)}$; $T^* \leftarrow \ln\left(1 - \left(\frac{l}{k}\right)^{\frac{1}{k-l}}\right) \cdot \left(\frac{k^{l-1}}{l^{k-l}}\right)^{\frac{1}{k-l}}$;
  else
    $z_{up} \leftarrow \check{z}_0$; $z_{lo} \leftarrow \hat{z}_0$;
    if $z_{in} < \check{z}_0$ then $z_{up} \leftarrow z_{in}$;
    if $z_{in} > \hat{z}_0$ then $z_{lo} \leftarrow z_{in}$;
    $\alpha_{min} \leftarrow \frac{1}{h(z_{up},l,k)}$; $\alpha_{max} \leftarrow \frac{1}{h(z_{lo},l,k)}$;
    while true do
      $\alpha^* \leftarrow \frac{\alpha_{max} - \alpha_{min}}{2} + \alpha_{min}$;
      $z^* \leftarrow \texttt{numSolve}(h(z,l,k) - \frac{1}{\alpha^*} = 0, z \in (z_{le}, z_{up}))$;
      $z^{**} \leftarrow \texttt{numSolve}(h(z,l,k) - \frac{1}{\alpha^*} = 0, z \in (z_{lo}, z_{ri}))$;
      $T^* \leftarrow T(z^*, l, k, \alpha^*)$; $T^{**} \leftarrow T(z^{**}, l, k, \alpha^*)$;
      if $|T^* - T^{**}| < \varepsilon$ then break;
      else
        if $T^* > T^{**}$ then $\alpha_{min} \leftarrow \alpha^*$;
        else $\alpha_{max} \leftarrow \alpha^*$;
      end
    end
  end
end
return(z^*, α^*, T^*)

$\mathbf{d}$	$[c_{\text{start}}, c_{\text{end}}]$	γ	$\lvert\check{c}(\mathbf{d}) - \gamma\rvert$	Σ_{sre}
$(3,\ 4)$	$[0.81951, 0.82351]$	0.821483	$0.247696 \cdot 10^{-4}$	0.00510340
$(3,\ 8)$	$[0.84938, 0.85338]$	0.851361	$0.211353 \cdot 10^{-4}$	0.00840384
$(3, 16)$	$[0.90889, 0.91289]$	0.910705	$1.898272 \cdot 10^{-4}$	0.00682809
$(3, 21)$	$[0.91804, 0.92204]$	0.919809	$2.296122 \cdot 10^{-4}$	0.00858563

Table 4.5.4.: Comparison of experimentally approximated and theoretical optimal 2-core thresholds for pseudorandom hypergraphs $H^{\mathbf{d}}_{m,n,\alpha}$ (type B) with two edge sizes. The values for γ are rounded to the nearest multiple of 10^{-6}.

the following experiments. We fixed the number of nodes to $m = 10^7$ and considered growing equidistant edge densities $c = n/m$ covering an interval $[c_{\text{start}}, c_{\text{end}}]$ of size 0.004 with the optimal theoretical 2-core threshold $\check{c}(\mathbf{d})$ in its center. For each quintuple $(c, m, d_0, d_1, \alpha^*)$ we constructed 10^2 pseudorandom random hypergraphs of type B with node set $[m]$, as well as $\alpha^* \cdot c \cdot m$ edges of size d_0 and $(1 - \alpha^*) \cdot c \cdot m$ edges of size d_1 in expectation. As in Section 3.4.4, all random choices were made using the pseudorandom number generator Mersenne Twister. Given a concrete hypergraph we applied Algorithm 1 to determine if its 2-core is empty. A non-empty 2-core was considered a failure, an empty 2-core was considered a success. We measured the failure rate and determined an approximation of the 2-core threshold, via fitting the sigmoid function (3.3) to the measured failure rate using a least squares fit, see Section 3.4.4. The resulting fit parameter γ is shown in Table 4.5.4. The quality of the approximation is quantified in terms of the sum of squares of residuals Σ_{sre}.

Results Theoretical and estimated thresholds are given in Table 4.5.4. The results show an absolute difference of less than $2.3 \cdot 10^{-4}$ at a very low fitting error of less than 0.009. The corresponding plots of the measured failure rates and the fit function are shown in Figure 4.5.4. They indicate a very sharp phase transition from success to failure.

4.5.5. Auxiliary Functions

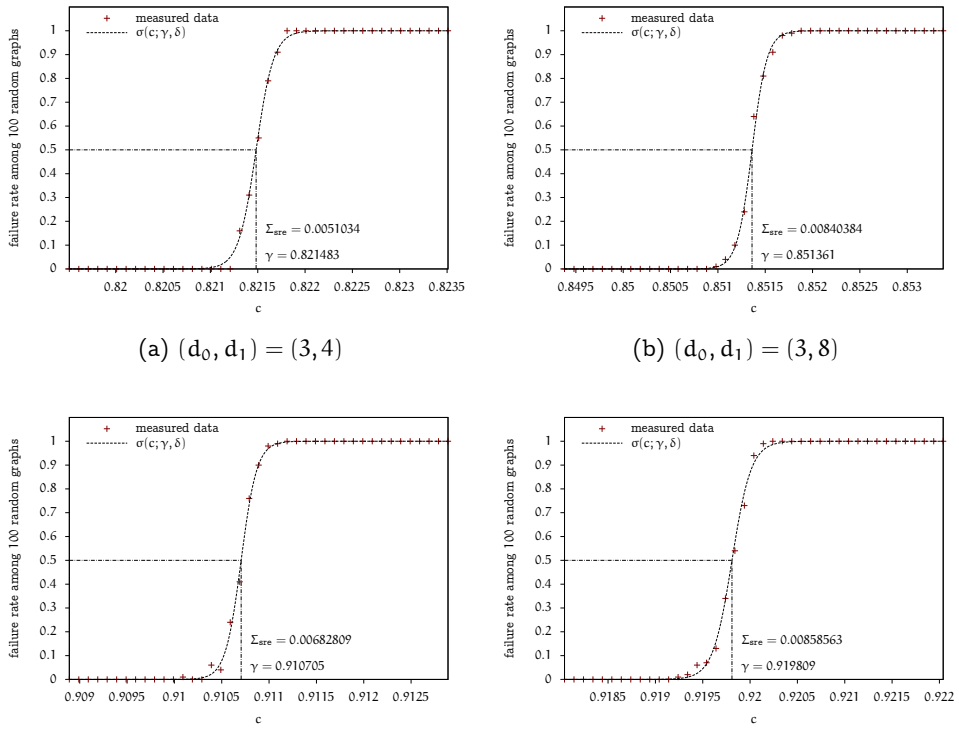
In the following we prove properties of the auxiliary functions f, g, and h as stated in Lemmas 4.5.1 to 4.5.3.

4.5.5.1. Properties of $f(z)$

In this section we show Lemma 4.5.1. Let $z \in (0, 1)$ and let $3 \leqslant l < k$.

ad (i): $f(z) = \dfrac{-\ln(1-z) \cdot (1-z)}{z} \overset{!}{>} 1 - z$

$$\Leftrightarrow -\ln(1-z) > z \Leftrightarrow \frac{1}{1-z} > e^z \Leftrightarrow e^{-z} > 1 - z\,.$$



(a) $(d_0, d_1) = (3, 4)$

(b) $(d_0, d_1) = (3, 8)$

(c) $(d_0, d_1) = (3, 16)$

(d) $(d_0, d_1) = (3, 21)$

Figure 4.5.4.: Approximation of optimal 2-core thresholds $\check{c}(\mathbf{d})$ for pseudorandom hypergraphs $H^{\mathbf{d}}_{m,n,\alpha}$ (type B) with $m = 10^7$ nodes.

ad (ii): Applying L'Hôpital's rule it follows that

$$\lim_{z\to 0} f(z) = \lim_{z\to 0} \frac{-\ln(1-z)\cdot(1-z)}{z} = \lim_{z\to 0} \frac{\frac{1}{|1-z|}\cdot(1-z)+\ln(1-z)}{1} = 1\,.$$

ad (iii): Applying L'Hôpital's rule it follows that

$$\begin{aligned}\lim_{z\to 1} f(z) &= \lim_{z\to 1} \frac{-\ln(1-z)\cdot(1-z)}{z} = \lim_{z\to 1} \frac{-\ln(1-z)}{\frac{z}{1-z}} = \lim_{z\to 1} \frac{\frac{1}{|1-z|}}{\frac{1-z+z}{(1-z)^2}} \\ &= \lim_{z\to 1} 1-z = 0\,.\end{aligned}$$

ad (iv): $\dfrac{df(z)}{dz} = \dfrac{z+\ln(1-z)}{z^2} \overset{!}{<} 0 \Leftrightarrow \ln(1-z) < -z \Leftrightarrow 1-z < e^{-z}\,.$

ad (v): $\dfrac{d^2 f(z)}{(dz)^2} = \dfrac{-2z+z^2-2\ln(1-z)\cdot(1-z)}{(1-z)\cdot z^3} \overset{!}{<} 0 \Leftrightarrow \underbrace{\dfrac{z^2-2\cdot z}{1-z}}_{f_1(z)} < \underbrace{2\ln(1-z)}_{f_2(z)}\,,$

which is true since it holds $\lim_{z\to 0} f_1(z) = \lim_{z\to 0} f_2(z) = 0$ and

$$\frac{df_1(z)}{dz} = \frac{-z^2+2\cdot z-2}{(1-z)^2} < \frac{df_2(z)}{dz} = \frac{-2}{1-z} < 0\,.$$

ad (vi): First we show that z_{in} is strictly increasing for growing l and fixed k, since

$$\begin{aligned}\frac{\partial z_{in}}{\partial l} \overset{!}{>} 0 &\Leftrightarrow z_{in}\cdot\left(\frac{\ln(l/k)}{(k-l)^2}+\frac{1}{l\cdot(k-l)}\right) > 0 \\ &\Leftrightarrow \frac{k-l}{l} > \ln(k/l) \Leftrightarrow \exp\left(\frac{k-l}{l}\right) > \frac{k}{l} \\ &\Leftrightarrow \sum_{i=0}^{\infty}\left(\frac{k-l}{l}\right)^i\cdot\frac{1}{i!} > \frac{k}{l} \\ &\Leftrightarrow 1+\frac{k-l}{l}+\underbrace{\left(\frac{k-l}{l}\right)^2\cdot\frac{1}{2}+\left(\frac{k-l}{l}\right)^3\cdot\frac{1}{6}+\ldots}_{>0} > \frac{k}{l}\,.\end{aligned}$$

Utilizing Lemma 4.5.1 (iv), this implies that $f(z_{in})$ is strictly monotonically decreasing for growing l. Now all we need to show is that our assumption holds for the maximum value of l, that is $l = k-1$.

$$\begin{aligned}f\left(\left(\frac{k-1}{k}\right)^{\frac{1}{k-(k-1)}}\right) \overset{!}{>} \frac{1}{k-1} &\Leftrightarrow \frac{-\ln\left(1-\frac{k-1}{k}\right)\cdot\left(1-\frac{k-1}{k}\right)}{\frac{k-1}{k}} > \frac{1}{k-1} \\ &\Leftrightarrow \frac{-\ln(1/k)}{k-1} > \frac{1}{k-1} \Leftrightarrow \ln(k) > 1\,,\end{aligned}$$

which is true since $k \geqslant 4$.

This finishes the proof of Lemma 4.5.1. ■

4.5.5.2. Properties of $g(z, l, k)$

In this section we prove Lemma 4.5.2. Let $z \in (0,1)$ and let $3 \leqslant l < k$.

ad (i): Considering the first derivative of $g(z)$, we get

$$\begin{aligned}
\frac{\partial g(z)}{\partial z} \overset{!}{<} 0 &\Leftrightarrow \frac{\ln(1-z)\cdot(k-1)\cdot(l-1)}{z^2} + \frac{1}{(1-z)^2} + \frac{(k-1)\cdot(l-1)}{z} < 0 \\
&\Leftrightarrow \frac{-f(z)\cdot(k-1)\cdot(l-1)}{z\cdot(1-z)} + \frac{1}{(1-z)^2} + \frac{(k-1)\cdot(l-1)}{z} < 0 \\
&\Leftrightarrow 1 - z + \frac{z}{1-z}\cdot\frac{1}{(k-1)\cdot(l-1)} < f(z) \\
&\Leftrightarrow \frac{1}{(k-1)\cdot(l-1)} < \underbrace{\frac{1-z}{z}\cdot(f(z)-(1-z))}_{g_1(z)} .
\end{aligned}$$

Hence, we have $\frac{\partial g(z)}{\partial z}$ R $0 \Leftrightarrow \frac{1}{(k-1)\cdot(l-1)}$ R $g_1(z)$ for all R $\in \{<, >, =\}$. Now consider $g_1(z)$. It holds that

- $\lim_{z\to 0} g_1(z) = 0.5$, since

$$\begin{aligned}
\lim_{z\to 0} g_1(z) &= \lim_{z\to 0} \frac{(1-z)\cdot(f(z)-1+z)}{z} \\
&= \lim_{z\to 0} \frac{-1\cdot(f(z)-1+z) + (1-z)\cdot\left(\frac{df(z)}{dz}+1\right)}{1} = 1 + \lim_{z\to 0}\frac{df(z)}{dz} \\
&= 1 + \lim_{z\to 0}\frac{z+\ln(1-z)}{z^2} = 1 + \lim_{z\to 0}\frac{1+\frac{-1}{1-z}}{2\cdot z} \\
&= 1 + \lim_{z\to 0}\frac{-z}{2\cdot z\cdot(1-z)} = 1 + \lim_{z\to 0}\frac{-1}{2\cdot(1-z)-2\cdot z} = 0.5 ,
\end{aligned}$$

  by applying L'Hôpital's rule three times.

- $\lim_{z\to 1} g_1(z) = \frac{1-1}{1}\cdot(0-1+1) = 0$.

- $g_1(z)$ is strictly decreasing for growing $z \in (0,1)$, since

$$\begin{aligned}
&\frac{dg_1(z)}{dz} = -\frac{-2\cdot\ln(1-z)\cdot(1-z)+z^3+z^2-2\cdot z}{z^3} \overset{!}{<} 0 \\
\Leftrightarrow &2\cdot\ln(1-z)\cdot(1-z) < z^3+z^2-2\cdot z \Leftrightarrow \ln(1-z) < \underbrace{\frac{z^3+z^2-2\cdot z}{2\cdot(1-z)}}_{g_2(z)} ,
\end{aligned}$$

  which is true because $\lim_{z\to 0}\ln(1-z) = 0 = \lim_{z\to 0} g_2(z) = 0$ and

$$\frac{d\ln(1-z)}{dz} = \frac{-1}{1-z} < \frac{dg_2(z)}{dz} = -1-z < 0 .$$

Using that $0 < \frac{1}{(k-1)\cdot(l-1)} < 0.5$, it follows that for growing z there is a first phase with $g_1(z) > \frac{1}{(k-1)\cdot(l-1)}$, which implies $\frac{dg(z)}{dz}<0$. Then there is exactly one z where $g_1(z) = \frac{1}{(k-1)\cdot(l-1)}$, which is a local minimum point for $g(z)$. After this point we have $g_1(z) < \frac{1}{(k-1)\cdot(l-1)}$, which implies $\frac{dg(z)}{dz}>0$. It follows that the local minimum is actual a global minimum.

ad (ii): If $z \in (0, z_{le}]$, then it holds $f(z) \geqslant \frac{1}{l-1} > \frac{1}{k-1}$. Furthermore, according to Lemma 4.5.1 (i), we have $\frac{1}{1-z} > \frac{1}{f(z)}$. Let $f(z) = \frac{1+\varepsilon}{l-1}$ and $f(z) = \frac{1+\delta}{k-1}$ as well as $\frac{1}{1-z} = \frac{1+\gamma}{f(z)}$ with $\varepsilon \geqslant 0$ and $\delta, \gamma > 0$. Using that $f(z) > 0$, for $z \in (0, 1)$, it follows that

$$\begin{aligned} g(z) \overset{!}{>} 0 &\Leftrightarrow f(z)\cdot(k-1)\cdot(l-1) + \frac{1+\gamma}{f(z)} - (l-1) - (k-1) > 0 \\ &\Leftrightarrow f(z)^2\cdot(k-1)\cdot(l-1) + (1+\gamma) - f(z)\cdot(l-1) - f(z)\cdot(k-1) > 0 \\ &\Leftrightarrow (1+\varepsilon)\cdot(1+\delta) + (1+\gamma) - (1+\varepsilon) - (1+\delta) > 0 \\ &\Leftrightarrow \varepsilon\cdot\delta + \gamma > 0\,. \end{aligned}$$

ad (iii): If $z \in [z_{ri}, 1)$, then it holds $f(z) \leqslant \frac{1}{k-1} < \frac{1}{l-1}$. Furthermore, according to Lemma 4.5.1 (i), we have $\frac{1}{1-z} > \frac{1}{f(z)}$. Let $f(z) = \frac{1-\varepsilon}{l-1}$ and $f(z) = \frac{1-\delta}{k-1}$ as well as $\frac{1}{1-z} = \frac{1+\gamma}{f(z)}$ with $\varepsilon \geqslant 0$ and $\delta, \gamma > 0$. Following the proof of Lemma 4.5.2 (ii) we get

$$\begin{aligned} g(z) \overset{!}{>} 0 &\Leftrightarrow (1-\varepsilon)\cdot(1-\delta) + (1+\gamma) - (1-\varepsilon) - (1-\delta) > 0 \\ &\Leftrightarrow \varepsilon\cdot\delta + \gamma > 0\,. \end{aligned}$$

ad (iv): For $\min g(z) < 0$, the existence of the roots $\check{z}_0$ and $\hat{z}_0$ follows directly from Lemma 4.5.2 (i). Moreover, from Lemma 4.5.2 (ii) and (iii) it follows that if $g(z) \leqslant 0$ for $z \in (0, 1)$, then it holds $z \in (z_{le}, z_{ri})$.

ad (v): Let $z > z_{le}$, i. e., $f(z) = \frac{1-\varepsilon}{l-1}$ for $\varepsilon > 0$. It follows that

$$\begin{aligned} g(z, l, k) &= f(z)\cdot(k-1)\cdot(l-1) + \frac{1}{1-z} + 2 - k - l \\ &= f(z)\cdot k\cdot(l-1) + \frac{1}{1-z} + 2 - (k+1) - l - f(z)\cdot(l-1) + 1 \\ &= g(z, l, k+1) - f(z)\cdot(l-1) + 1 = g(z, l, k+1) - (1-\varepsilon) + 1 \\ &> g(z, l, k+1)\,. \end{aligned}$$

ad (vi): Assume that there is some k' such that $z_{min} = \arg\min_z g(z, l, k') < 0$. Then by Lemma 4.5.2 (ii) and (iii) we have that $z_{min} > z_{le}$. Using Lemma 4.5.2 (v), we conclude that for all $k \geqslant k'$ it holds that $g(z_{min}, l, k) < 0$ and therefore $\min_z g(z, l, k) < 0$ as well. It remains to find one such k'.

Consider the inequality $g(z_{in}, l, k) \geqslant 0$, which is equivalent to

$$(k-1)\cdot\Big(\underbrace{f(z_{in})\cdot(l-1)}_{g_1(l,k)} + \underbrace{\frac{1}{1-z_{in}}\cdot\frac{1}{k-1}}_{g_2(l,k)} - 1\Big) - (l-1) \geqslant 0\,.$$

Assume that $\lim_{k\to\infty} g_1(k) = 0$ and $\lim_{k\to\infty} g_2(k) \leqslant 0$. It follows that there must be a k' with $g(z_{in}(l,k'),l,k') < 0$ and thus $\min_z g(z_{in}(l,k'),l,k') < 0$.

- ▷ $\lim_{k\to\infty} g_1(k) = 0$: It holds $\lim_{k\to\infty} z_{in} = 1$, since

$$\begin{aligned}\lim_{k\to\infty} z_{in} &= \lim_{k\to\infty} \exp\left(\ln\left(\frac{l}{k}\right)^{\frac{1}{k-l}}\right) = \lim_{k\to\infty}\exp\left(\frac{\ln(l)-\ln(k)}{k-l}\right)\\ &= \exp\left(\lim_{k\to\infty}\frac{\ln(l)}{k-l} - \lim_{k\to\infty}\frac{\ln(k)}{k-l}\right) = \exp\left(0 - \lim_{k\to\infty}\frac{1/k}{1}\right) = 1\,.\end{aligned}$$

  By Lemma 4.5.1 (iii), we have that $\lim_{k\to\infty} f(z_{in}(k)) = 0 = \lim_{k\to\infty} g_1(k)$.

- ▷ $\lim_{k\to\infty} g_2(k) \leqslant 0$: Since $\frac{2-z}{1-z} > \frac{1}{1-z}$, for $z \in (0,1)$, it is sufficient to show that $\lim_{k\to\infty}\frac{2-z_{in}}{1-z_{in}}\cdot\frac{1}{k-1} = 0$. We get

$$\begin{aligned}&\lim_{k\to\infty}\frac{(2-z_{in})/(k-1)}{1-z_{in}} = \lim_{k\to\infty}\frac{-\frac{1}{k-1}\cdot\frac{\partial z_{in}}{\partial k} + \frac{-2+z_{in}}{(k-1)^2}}{-\frac{\partial z_{in}}{\partial k}}\\ &= \lim_{k\to\infty}\left(\frac{1}{k-1} + \frac{\frac{1}{(k-1)^2}\cdot(2-z_{in})}{\frac{\partial z_{in}}{\partial k}}\right).\end{aligned}$$

  Using that $\frac{\partial z_{in}}{\partial k} = z_{in}\cdot\left(\frac{-\ln(l/k)}{(k-l)^2} - \frac{1}{k\cdot(k-l)}\right)$, it follows that

$$\begin{aligned}&\lim_{k\to\infty}\frac{(2-z_{in})/(k-1)}{1-z_{in}} = \lim_{k\to\infty}\frac{2-z_{in}}{(k-1)^2}\cdot\frac{1}{z_{in}}\cdot\left(\frac{-\ln(\frac{l}{k})}{(k-l)^2} - \frac{1}{k\cdot(k-l)}\right)^{-1}\\ &= \lim_{k\to\infty}\underbrace{\frac{2-z_{in}}{z_{in}}}_{\to 1}\cdot\Big(\underbrace{(k-1)^2\cdot\frac{-\ln(\frac{l}{k})}{(k-l)^2}}_{\to\infty} - \underbrace{\frac{(k-1)^2}{k\cdot(k-l)}}_{\to 1}\Big)^{-1} = 0\,.\end{aligned}$$

This finishes the proof of Lemma 4.5.2. ■

4.5.5.3. Properties of $h(z,l,k)$

In this section we prove Lemma 4.5.3. Let $z \in (0,1)$ and let $3 \leqslant l < k$.

ad (i): Considering the denominator of $h(z)$ leads to

$$k\cdot\big((k-1)\cdot f(z) - 1\big) \stackrel{!}{=} 0 \Leftrightarrow f(z) = \frac{1}{k-1}\,,$$

which is true for exactly one z from $(0,1)$, which is per definition $z = z_{ri}$.

ad (ii): With $\lim_{z\to 0} f(z) = 1$ (Lemma 4.5.1 (ii)) and

$$\lim_{z\to 0} \big(1 - f(z)\cdot(l-1)\big) = 2 - l \leqslant -1\,,$$

we get

$$\lim_{z\to 0} h(z) = \lim_{z\to 0} \frac{l\cdot\frac{1}{z^{k-l}}\cdot\big(1-f(z)\cdot(l-1)\big) - k + f(z)\cdot k\cdot(k-1)}{k\cdot\big((k-1)\cdot f(z) - 1\big)} = -\infty\,.$$

ad (iii): $\forall z \in (0, z_{ri})$: $f(z) > \frac{1}{k-1}$. Let $f(z) = \frac{1+\varepsilon}{k-1}$. Consider the limit of the numerator of $h(z)$:

$$\begin{aligned}&\lim_{\varepsilon\to 0} l\cdot\frac{1}{z_{ri}^{k-l}}\cdot\left(1-\frac{1+\varepsilon}{k-1}\cdot(l-1)\right) - k + \frac{1+\varepsilon}{k-1}\cdot k\cdot(k-1)\\ =&\lim_{\varepsilon\to 0} l\cdot\frac{1}{z_{ri}^{k-l}}\cdot\left(1-\frac{l-1}{k-1}\right) = K\,,\end{aligned}$$

for some positive constant $K = K(l,k)$. For the denominator of $h(z)$ it holds that

$$\lim_{\varepsilon\to+0} k\cdot\left((k-1)\cdot\frac{1+\varepsilon}{k-1} - 1\right) = k\cdot\varepsilon = +0\,.$$

Hence, $\lim_{z\to z_{ri}} h(z) = +\infty$.

ad (iv): $\forall z \in (0, z_{le}]$: $f(z) \geqslant \frac{1}{l-1}$. Let $\varepsilon \geqslant 0$ and let $f(z) = \frac{1+\varepsilon}{l-1}$. Hence, we have

$$\begin{aligned}& h(z) \overset{!}{\leqslant} 1\\ \Leftrightarrow\quad & \frac{l\cdot z^{l-k} - k - \frac{1+\varepsilon}{l-1}\cdot\big(l\cdot(l-1)\cdot z^{l-k} - k\cdot(k-1)\big)}{k\cdot\big((k-1)\cdot\frac{1+\varepsilon}{l-1} - 1\big)} < 1\\ \Leftrightarrow\quad & l\cdot z^{l-k} - (1+\varepsilon)\cdot l\cdot z^{l-k} - k + (1+\varepsilon)\cdot k\cdot\tfrac{k-1}{l-1} \leqslant -k + (1+\varepsilon)\cdot k\cdot\tfrac{k-1}{l-1}\\ \Leftrightarrow\quad & l\cdot z^{l-k} - (1+\varepsilon)\cdot l\cdot z^{l-k} \leqslant 0 \Leftrightarrow \varepsilon \geqslant 0\,.\end{aligned}$$

ad (v): $\forall z \in (z_{le}, z_{ri})$: $\frac{1}{l-1} \geqslant f(z) > \frac{1}{k-1}$. Let $\varepsilon > 0$ and let $\delta > 0$ with $\frac{1-\varepsilon}{l-1} = f(z) = \frac{1+\delta}{k-1}$. Hence, we have

$$\begin{aligned}h(z) \overset{!}{>} 1 &\Leftrightarrow \frac{l\cdot z^{l-k} - k - \frac{1-\varepsilon}{l-1}\cdot\big(l\cdot(l-1)\cdot z^{l-k} + \frac{1+\delta}{k-1}\cdot k\cdot(k-1)\big)}{k\cdot\big((k-1)\cdot\frac{1+\delta}{k-1} - 1\big)} > 1\\ &\Leftrightarrow l\cdot z^{l-k} - (1-\varepsilon)\cdot l\cdot z^{l-k} + (1+\delta)\cdot k - k > \delta\cdot k\\ &\Leftrightarrow \varepsilon\cdot l\cdot z^{l-k} > 0 \Leftrightarrow \varepsilon > 0\,.\end{aligned}$$

Note that for $z = z_{le}$, that is $\varepsilon = 0$, we have $h(z) = 1$.

ad (vi): $\forall z \in (z_{ri}, 1)$: $f(z) < \frac{1}{k-1}$. Let $\varepsilon \in (0,1)$ and let $f(z) = \frac{1-\varepsilon}{k-1}$. Hence, we have

$$\begin{aligned} h(z) \overset{!}{<} 1 &\Leftrightarrow \frac{l \cdot z^{l-k} - k - \frac{1-\varepsilon}{k-1} \cdot \big(l \cdot (l-1) \cdot z^{l-k} - k \cdot (k-1)\big)}{k \cdot \big((k-1) \cdot \frac{1-\varepsilon}{k-1} - 1\big)} < 1 \\ &\Leftrightarrow l \cdot z^{l-k} \cdot \left(1 - (1-\varepsilon) \cdot \frac{l-1}{k-1}\right) - k + k \cdot (1-\varepsilon) > -\varepsilon \cdot k \\ &\Leftrightarrow 1 - (1-\varepsilon) \cdot \frac{l-1}{k-1} > 0 \Leftarrow \varepsilon \in (0,1)\,. \end{aligned}$$

ad (vii): Recall (4.10) for representing $h(z)$:

$$h(z) = \frac{Z_0(z) - f(z) \cdot Z_1(z)}{k \cdot z^{k-1} \cdot \big((k-1) \cdot f(z) - 1\big)}\,.$$

Let $h_1(z) = (k-1) \cdot f(z) - 1$. Note that $h_1(z) \neq 0$, if $z \neq z_{ri}$.
The first derivative of $h(z)$ is

$$\begin{aligned} \frac{\partial h(z)}{\partial z} &= \frac{f(z)}{k \cdot z^k \cdot h_1(z)} \\ &\quad \cdot \left(\frac{Z_1(z)}{1-z} - Z_2(z) - (Z_0(z) - f(z) \cdot Z_1(z)) \cdot \frac{(k-1) \cdot \big(k - 1 - \frac{1}{1-z}\big)}{h_1(z)}\right). \end{aligned}$$

The function $h(z)$ is strictly increasing if and only if

$$\begin{aligned} \frac{\partial h(z)}{\partial z} \overset{!}{>} 0 &\Leftrightarrow \frac{\partial h(z)}{\partial z} \cdot h_1(z)^2 > 0 \\ &\Leftrightarrow f(z) \cdot \Big[(h_1(z) \cdot \big(\frac{Z_1(z)}{1-z} - Z_2(z)\big) \\ &\qquad - \big(Z_0(z) - f(z) \cdot Z_1(z)\big) \cdot (k-1) \cdot \Big(k - 1 - \frac{1}{1-z}\Big)\Big] > 0\,. \end{aligned}$$

Note that $f(z)$ is positive for $z \in (0,1)$. So we get

$$\begin{aligned} \frac{\partial h(z)}{\partial z} \overset{!}{>} 0 \Leftrightarrow \left(\frac{Z_1(z)}{1-z} - Z_2(z)\right) &\cdot \big((k-1) \cdot f(z) - 1\big) > \\ &\big(Z_0(z) - f(z) \cdot Z_1(z)\big) \cdot \Big((k-1)^2 - \frac{k-1}{1-z}\Big)\,. \end{aligned}$$

This inequality is equivalent to

$$\begin{aligned} &\frac{Z_1(z)}{1-z} \cdot (k-1) \cdot f(z) - \frac{Z_1(z)}{1-z} - Z_2(z) \cdot (k-1) \cdot f(z) + Z_2(z) > \\ &\frac{Z_1(z)}{1-z} \cdot (k-1) \cdot f(z) + Z_0(z) \cdot (k-1)^2 - Z_0(z) \cdot \frac{k-1}{1-z} - f(z) \cdot Z_1(z) \cdot (k-1)^2\,, \end{aligned}$$

which is equivalent to

$$f(z)\cdot(k-1)\cdot\big(Z_1(z)\cdot(k-1)-Z_2(z)\big) + \frac{1}{1-z}\cdot\big(Z_0\cdot(k-1)-Z_1(z)\big)+Z_2(z)-Z_0(z)\cdot(k-1)^2>0\,.$$

Expanding the functions $Z_j(z)$, $j\in\{0,1,2\}$, gives

$$\begin{aligned}\frac{\partial h(z)}{\partial z}\overset{!}{>}0 &\Leftrightarrow f(z)\cdot(k-1)\cdot l\cdot z^{l-1}\cdot\big((l-1)\cdot(k-1)-(l-1)^2\big)\\ &\quad+\frac{1}{1-z}\cdot l\cdot z^{l-1}\cdot\big((k-1)-(l-1)\big)\\ &\quad+l\cdot z^{l-1}\cdot\big((l-1)^2-(k-1)^2\big)>0\\ &\Leftrightarrow f(z)\cdot(k-1)\cdot(l-1)\cdot(k-l)\\ &\quad+\frac{1}{1-z}\cdot(k-l)+(l-1)^2-(k-1)^2>0\\ &\Leftrightarrow f(z)\cdot(k-1)\cdot(l-1)+\frac{1}{1-z}+2-k-l>0\\ &\overset{\text{def}}{\Leftrightarrow} g(z)>0\,,\end{aligned}$$

where we divided by $k-l$, which is larger than 0 by definition.

ad (viii): This follows directly from Lemma 4.5.2 (ii) and Lemma 4.5.3 (vii).

ad (ix): This follows directly from Lemma 4.5.2 (iii) and Lemma 4.5.3 (vii)

ad (x): For $\min_z g(z)>0$, this follows directly by Lemma 4.5.3 (vii). Now assume $\min_z g(z)=0$. According to Lemma 4.5.2 (i), the point $z_{\min}=\arg\min_z g(z)$ is the only point where $g(z)=0$. It follows that $z_{\min}$ is the only inflection point of $h(z)$. Therefore, by Lemma 4.5.3 (vii), $h(z)$ is strictly increasing.

ad (xi): According to Lemma 4.5.2 (iv), the function $g(z)$ has exactly two different roots $\check{z}_0,\hat{z}_0$ in the interval (z_{le},z_{ri}) and by Lemma 4.5.3 (vii) and Lemma 4.5.2 (i) it follows that $h(z)$ is strictly increasing for $z<\check{z}_0$, strictly decreasing for z with $\check{z}_0<z<\hat{z}_0$, and strictly increasing for $z>\hat{z}_0$. Hence the claim follows.

This finishes the proof of Lemma 4.5.3. ■

4.6. Perfect Hashing via Matchings in Bipartite Graphs

In this section we discuss our construction of an irregular mutable Bloomier filter from Sections 4.1.3 and 4.1.4, which is a perfect hash function build upon a combination of a left-regular random bipartite graph and a left-irregular random bipartite graph — the construction was sketched before in [Rin12, Rin13]. In an experimental case study,

which includes all aspects one might encounter when using perfect hashing as, e. g., normalization of keys, we show that with simple, non-ideal hash functions a space consumption of 1.76 bits per key can be obtained in linear time for natural key sets of size $n \geqslant 10^7$ and range $1.1 \cdot n$.

4.6.1. Construction Algorithm

To get started, we restate the construction algorithm, sketched in Section 4.1.4, in more detail. We split the algorithm in three phases, a preprocessing phase (phase 0), and two main phases (phases 1 and 2).

Phase 0 The first phase is optional. It is used to map keys from some (almost) arbitrary universe to pairwise distinct integers. This allows to uniformly handle inputs from different universes avoiding adjustments in the main phases.

Remark. In applications where one does not emphasize the *normalization* of the keys but rather the reduction of their description length, this technique is called *domain reduction* or *collapsing the universe*, see Section 5.4.1.

Let $\check{U}$ be the set of all possible inputs, let $\check{S}$ be the actual key set, and let $U \subseteq \mathbb{N}$ be the universe that is used in phases 1 and 2. If $U = \check{U}$, then phase 0 is skipped. Otherwise, phase 0 starts with a selection of a random hash function

$$h^0 \colon \check{U} \to U\,,$$

which is used to define a new key set S, via

$$S \coloneqq \{h^0(\check{x}) \mid \check{x} \in \check{S}\}\,.$$

If $|S| < |\check{S}|$, i. e., there are keys from $\check{S}$ that collide under h^0, then phase 0 starts again with a new random hash function h^0.

Phase 1 The phase begins with a random selection of $\acute{d}$ hash functions

$$h^1_0, h^1_1, \ldots, h^1_{\acute{d}-1} \colon U \to [\acute{m}]\,,$$

which are used to define a d-left-regular bipartite graph

$$G^{\acute{d}}_{n,\acute{m}} = (S \cup [\acute{m}], \acute{E})$$

of type A with edge set

$$\acute{E} = \left\{(x, h^1_i(x)) \mid x \in S, i \in [\acute{d}]\right\}.$$

The generalized selfless algorithm (Algorithm 3) is applied to the hypergraph representation $H^{\acute{d}}_{\acute{m},n}$ of $G^{\acute{d}}_{n,\acute{m}}$ in order to determine a left-perfect matching M in $G^{\acute{d}}_{n,\acute{m}}$.

REMARK. Alternatively, one can use *local search allocation* by Khosla [Kho13, Section 2], which always finds a left-perfect matching in $G^{\acute{d}}_{n,\acute{m}}$ if such one exists, has linear running time w. h. p., and is fast in practice, see Section 1.1.

In the case that no left-perfect matching is found, phase 1 starts again with new random hash functions. Otherwise, a vector $\boldsymbol{\nu} = (\nu_x)_{x \in S}$ is built, where $\nu_x = i \in [\acute{d}]$, the index of a hash function, if and only if edge $(x, h^1_i(x))$ is a matching edge from M.

PHASE 2 Let $\boldsymbol{\alpha} = (\alpha_i)_{i \in [s]} \in [0,1]^s$ as well as $\mathbf{d} = (d_i)_{i \in [s]} \in \mathbb{N}^s$ be two vectors of length $s > 0$. Furthermore, define $\hat{d} = \max\{d_i \mid i \in [s]\}$. The last phase starts with a random selection of $\hat{d}$ hash functions

$$h^2_0, h^2_1, \ldots, h^2_{\hat{d}-1} \colon U \to [m] \text{ with } h^2_i(x) \neq h^2_j(x) \text{ for all } x \in U \text{ and } 0 \leqslant i < j < \hat{d},$$

see Section 5.3.3.2, as well as one additional hash function

$$h^2_{\hat{d}} \colon U \to [s] \text{ with } \Pr(h^2_{\hat{d}}(x) = i) = \alpha_i \text{ for all } i \in [s].$$

The hash functions are used to build a random $\mathbf{d}$-left-irregular bipartite graph

$$G^{\mathbf{d}}_{n,m,\boldsymbol{\alpha}} = (S \cup [m], E)$$

of type B with edge set

$$E = \left\{(x, h^2_i(x)) \mid x \in S, i \in [d_j], j = h^2_{\hat{d}}(x)\right\},$$

where the degree of each left node x is determined via hash function $h^2_{\hat{d}}$.

Peeling with back substitution (Algorithm 7) is applied to the matrix representation $\boldsymbol{M}^{\mathbf{d}}_{n,m,\boldsymbol{\alpha}}$ of $G^{\mathbf{d}}_{n,m,\boldsymbol{\alpha}}$ and the vector $\boldsymbol{\nu}$, which consists of elements from the abelian group $(\mathbb{Z}_{\acute{d}}, +)$, in order to solve the system of equations

$$\boldsymbol{M}^{\mathbf{d}}_{n,m,\boldsymbol{\alpha}} \cdot \mathbf{t} = \boldsymbol{\nu}.$$

In the case that $G^{\mathbf{d}}_{n,m,\boldsymbol{\alpha}}$ has an order generating matching, peeling with back substitution returns a solution vector $\mathbf{t}$. Otherwise, phase 2 starts again with new random hash functions. If a solution $\mathbf{t}$ is obtained, it is optionally compressed. A standard approach is to consider every a many consecutive entries of $\mathbf{t}$ as a base-$\acute{d}$ representation of a number that replaces the a entries and has description length of at most b bit. More precisely, if there are non-zero constants $a, b \in \mathbb{N}$ with $\lceil \log(\acute{d}) \rceil > \frac{b}{a} \geqslant \log(\acute{d})$, then encode the solution vector $\mathbf{t} = (t_i)_{i \in [m]}$, $t_i \in [\hat{d}]$, to a vector $\mathbf{t}^{\text{enc}}$ according to

$$\mathbf{t}^{\text{enc}} = (t^{\text{enc}}_j)_{j \in [\lceil m/a \rceil]} \text{ with } t^{\text{enc}}_j \coloneqq (t_{j \cdot a + 0} \circ t_{j \cdot a + 1} \circ \ldots \circ t_{j \cdot a + a - 1})_{\text{base } \acute{d}}$$
$$\text{and } t_i \coloneqq 0 \text{ for all } i \geqslant m,$$

where each element t_j^{enc} needs b bits. For fast decoding a lookup matrix $\mathbf{T}^{\text{dec}}$ can be used, where

$$\mathbf{T}^{\text{dec}} = (t_{i,j}^{\text{dec}})_{i\in[\acute{d}^a],j\in[a]} \text{ with } (t_{i,0}^{\text{dec}} \circ t_{i,1}^{\text{dec}} \circ \ldots \circ t_{i,a-1}^{\text{dec}})_{\text{base } \acute{d}} = i, \text{ for all } i \in [\acute{d}^a].$$

The parameters for all three phases have to be chosen with respect to the desired lookup time as well as construction time and space consumption, which are mainly limited by the probability of the event that $G_{n,\acute{m}}^{\acute{d}}$ has a left-perfect matching and $G_{n,m,\alpha}^{\mathbf{d}}$ has an order generating matching. We will discuss appropriate parameters in the forthcoming Section 4.6.2.

Lookup Operation After a successful construction, we obtain the desired hash function $h\colon \breve{U} \to [\acute{m}]$ that is injective with respect to $\breve{S}$, i. e., a perfect hash function for $\breve{S}$. Now, in order to simplify the description of the evaluation of the perfect has function we will ignore key normalization and compression for a moment. Accordingly, the lookup for a key $x \in U$ is carried out as follows:

1. Determine the index j of the number d_j of phase 2 hash functions, using $h_{\hat{d}}^2$ via
$$j \coloneqq h_{\hat{d}}^2(x)\,.$$
2. Access the vector $\mathbf{t}$ with the d_j hash functions $h_0^2, h_1^2, \ldots, h_{d_j-1}^2$ and determine an index k of a phase 1 hash function, according to
$$k \coloneqq \left(\sum_{i\in[d_j]} t_{h_i^2(x)}\right) \bmod \acute{d}\,.$$
3. The value of the perfect hash function is equal to the value of the phase 1 hash function h_k^1, i. e.,
$$h(x) \coloneqq h_k^1(x)\,.$$

Space and Time Consumption If we assume constant evaluation time for the hash functions h^0, h_i^1, $i \in [\acute{d}]$, and h_j^2, $j \in [\hat{d}+1]$, and ignore their space needs, we obtain the following performance characteristics:

space usage	$m \cdot \lceil \log \acute{d} \rceil$	without compression,
	$\lceil m/a \rceil \cdot b$	with simple compression,

lookup time $O(\hat{d})$.

Suitable values are discussed below.

Remark. Using more sophisticated compression the space complexity can be reduced to $\lceil m \cdot \log \acute{d} \rceil + m/(\log m)^{O(1)}$ [Pă08, Theorem 1] and even to $\left(\lceil m \cdot \log \acute{d} \rceil + o(1)\right)$ [DPT10, Theorem 1 and Section 4.2], while maintaining constant lookup time.

4.6.2. Experiments

In our experiments we applied the algorithm on natural key sets, using one single set of parameters and practical, i. e., non-ideal, hash functions.

Algorithm Parameters Based on the results from Section 4.5, the following parameters were chosen according to Table 4.1.2:

- $\acute{d} = 3$ and $\mathbf{d} = (d_0, d_1) = (3, 16)$ with $\hat{d} = \max\{d_0, d_1\} = 16$,
- $\acute{m} = m = 1.1 \cdot n$,
- $\boldsymbol{\alpha} = (\alpha^*, 1 - \alpha^*)$ with $\alpha^* \approx 0.88684$,
- simple compression scheme with $a = 5$, $b = 8$, and decompression matrix $\mathbf{T}^{\text{dec}}$.

It follows that, if one excludes the hash functions, the space usage is $\lceil 1.1 \cdot n/5 \rceil \cdot 8$ bits, i. e., asymptotically 1.76 bits per key.

Key Sets We defined $\breve{U}$ to be the set of binary sequences with length 1 up to 252 bytes, i. e.,

$$\breve{U} := \bigcup_{i=1}^{252} \{0,1\}^{8 \cdot i}.$$

The keys for our experiments were restricted to elements from some set $\breve{U}_w \subseteq \breve{U}$ with 10151094 elements[14]. The set $\breve{U}_w$ is the set of *titles* and the set of *category labels* of the *English Wikipedia*, extracted from dumps generated in 2012, see [Wik12].

Choosing a key set $\breve{S}$ of size n from $\breve{U}$ was done according to the following randomized procedure. Determine a fully random index i from $[|\breve{U}_w| - n + 1]$ and define $\breve{S}$ as the set of n consecutive elements from the sorted list of all elements from $\breve{U}_w$, starting at position i.

The idea behind this approach was to get *natural key sets* with a reduced amount of entropy in order to shrink the influence of the random choice of the keys on the performance of our randomized construction algorithm, i. e., the main source of randomness of the algorithm should be the choice of the coefficients of the hash functions, see Section 5.2 for more background.

Hash Functions Although the analysis of the data structure is carried out only for ideal hash functions, see Section 2.2.2, in the experiments we replace these functions by practical ones with weak random properties and not by some pseudorandom number generator.

[14] The reason why we did not define $\breve{U}_w = \breve{U}$ is our assumption that the universe has a simple structure, see Section 2.1.

REMARK. The discrepancy between assuming fully randomness in theoretical analysis and providing limited randomness in practical solutions is addressed in Chapter 5.

For each $\check{x} \in \check{U}$ let $\check{x} = \check{x}_0 \circ \check{x}_1 \circ \ldots \circ \check{x}_{l-1}$, with $\check{x}_i \in \{0,1\}^8$ for $i \in [l]$. We reduced the domain $\check{U}$ to a smaller domain $U = [p]$, with $p = 2^{61} - 1$, via appending the smallest non-negative number of blanks (byte 00100000) on each key such that the number of bytes of the extended key is divisible by 4, evaluating a separate hash function on each four consecutive bytes, which are interpreted as integers, and finally adding up the results. More precisely, we use the following hash function $h^0 \colon \check{U} \to U$

$$h^0(\check{x}) \coloneqq \Bigg(\bigoplus_{i=0}^{\lfloor l/4 \rfloor - 1} h_i^0(\check{x}_{i\cdot 4} \circ \check{x}_{i\cdot 4+1} \circ \check{x}_{i\cdot 4+2} \circ \check{x}_{i\cdot 4+3}) \\ \oplus\, h_{\lfloor l/4 \rfloor}^0(\check{x}_{\lfloor l/4 \rfloor \cdot 4} \circ \check{x}_{\lfloor l/4 \rfloor \cdot 4+1} \circ \ldots \circ \check{x}_{l-1} \circ \underbrace{00100000 \circ \ldots \circ 00100000}_{(l \bmod 4) \text{ times}}) \Bigg) \bmod p\,,$$

where for all $i \in [63]$, we have $h_i^0 \colon \{0,1\}^{32} \to [p]$ according to

$$h_i^0(\check{x}) \coloneqq \left(a_{0,i}^0 + a_{1,i}^0 \cdot \check{x}\right) \bmod p \text{ with } a_{0,i}^0, a_{1,i}^0 \in [p].$$

All hash functions $h_i^1 \colon U \to [\acute{m}]$, $i \in [\acute{d}]$, of phase 1 are realized as polynomials with $\kappa \geqslant 2$ coefficients, i. e., of degree $\kappa - 1$, according to

$$h_i^1(x) \coloneqq \left(\left(\sum_{j=0}^{\kappa-1} a_{j,i}^1 \cdot x^j \right) \bmod p \right) \bmod \acute{m}\,.$$

All hash functions $h_i^2 \colon U \to [m]$, $i \in [\hat{d}+1]$, of phase 2 are also realized via polynomials of degree $\kappa - 1$, according to

$$h_i^2(x) \coloneqq \left(\left(\sum_{j=0}^{\kappa-1} a_{j,i}^2 \cdot x^j \right) \bmod p \right) \bmod m\,,$$

and

$$h_{\hat{d}}^2(x) \coloneqq \begin{cases} 3, & \text{if } \left(\left(\sum_{j=0}^{\kappa-1} a_{j,\hat{d}}^2 \cdot x^j \right) \bmod p \right) \bmod m < \alpha^* \cdot m \\ 16, & \text{otherwise} \end{cases}.$$

Storing the hash functions boils down to storing the coefficients $a_{j,i}^k$ of the polynomials, as well as storing the range $\acute{m} = m$ and the prime number p.

REMARK. Although possible, we did not share coefficients between distinct polynomials, since we wanted to demonstrate the efficiency of the construction algorithm without the need for further modifications.

The random selection of the hash functions for each phase was done via generating pseudorandom coefficients $a_{j,i}^k$ using the Mersenne Twister, see Section 3.4.4.

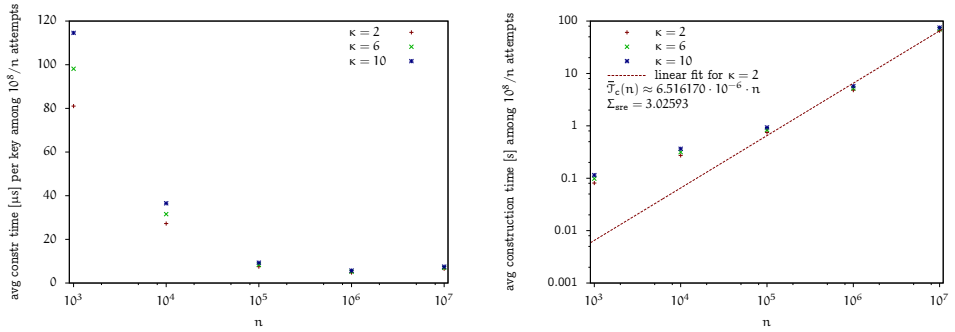
SETUP AND MEASUREMENTS For each pair (n, κ) from $\{10^3, 10^4, 10^5, 10^6, 10^7\} \times \{2, 3, \ldots, 10\}$ we selected $a = 10^8/n$ key sets $\check{S}$ as described above and applied the construction algorithm on input $(\check{S}, \acute{m})$, using hash functions for phases 1 and 2 based on polynomials of degree $\kappa - 1$. Given (n, κ), we measured sample mean and sample variance of the following quantities:

- ▷ construction time T_c, broken down to the different phases,
- ▷ the number of iterations for each phase i^0, i^1, i^2,
- ▷ the ratio $r_{\alpha^*} = |\{x \in S \mid h^2_{\hat{d}}(x) = 3\}|/n$, i. e., the approximation of α^*,
- ▷ the overall space consumption S,
- ▷ the lookup time, i. e., the evaluation time T_e of the perfect hash function.

RESULTS For the sake of conciseness, we present results only for the cases $\kappa = 2, 6, 10$. Qualitative results for $\kappa = 3, 4, 5$ and $\kappa = 7, 8, 9$ can be deduced easily from them.

I. Figure 4.6.5 shows the average construction time $\bar{T}_c(n)$ and the average construction time per key $\bar{T}_c(n)/n$. The average time consumption per key for the different phases are visualized in Figure 4.6.6. In contrast to what is expected for the theoretical unit-cost RAM, $n \to \infty$, and ideal hash functions, the measured average construction time is not linear. Reasons for the non-linear behavior are:

0. The average construction time per key for phase 0 increases slightly for increasing n from $n = 10^4$ to $n = 10^6$, while the number of iterations for this phase is always 1, see Table 4.6.5. This is, at least partially, because of the way the key sets are constructed, the expected number of large keys increases with increasing n.
1. The average construction time per key for phase 1 increases with increasing n, while the number of iterations for this phase is constantly close to 1, see Table 4.6.5. This is probably due to caching effects.
2. The average construction time per key for phase 2 decreases for increasing n from $n = 10^3$ to $n = 10^6$, since the larger n, the less likely phase 2 fails, i. e., the fewer iterations are needed, see Table 4.6.5. In contrast, from $n = 10^6$ to $n = 10^7$ the average construction time for phase 2 increases in conformity with the behavior of phase 1.
   For $n = 10^3, 10^4, 10^5$ the number of iteration of phase 2 is larger than the number of iterations of phase 1, since $n/\acute{m} = n/m \approx 0.9091$ is much closer to $\check{c}\big((3, 16), (\alpha^*, 1 - \alpha^*)\big) \approx 0.91089$ than to $\hat{c}(3) \approx 0.9179$ and therefore the failure probability of phase 2 is likely to be much higher. Furthermore, there is a strong dependence on the fraction of nodes of degree 3 in the left-irregular bipartite graph but an adequate approximation of α^* can be only realized for relatively large n, see Table 4.6.5.



(a) time per key [µs] ($\log_{10}$ plot) (b) overall time [s] ($\log_{10}$-$\log_{10}$ plot)

Figure 4.6.5.: Average construction time of the perfect hash function. Ideally, the time consumption should be linear in n, i. e., measurements in Figure 4.6.5 (a) should be on a line parallel to the abscissa. However, the data show a slight non-linear behavior.

The crucial observation is that for fixed n, the average number of iterations per phase remains approximately the same for all $\kappa = 2, 3, \ldots, 10$. Hence, the case $\kappa = 2$ gives the minimum construction time.

In order to get a linear prediction model for $n > 10^7$, we fitted a linear function to the measured average construction time for $\kappa = 2$ using the method of least squares provided by gnuplot [WKL+12], see also Section 3.4.4. The fit is shown in Figure 4.6.5 (b), where Σ_{sre} denotes the sum of squares of the residuals.

Remark. Using a weighted fit, up to ignoring $n = 10^3$ and $n = 10^4$ completely, changed the prediction only marginally, so that we simply used the standard method and all five measurement points.

II. In contrast to the construction time, the average evaluation time behaves linearly as one can see in Figure 4.6.7. We derived a good linear prediction for $n > 10^7$ and $\kappa = 2$ via the method of least squares as shown in Figure 4.6.7 (b). Clearly, also with respect to minimum evaluation time, the value $\kappa = 2$ is the best choice.

III. As shown in Table 4.6.6, for fixed κ the average space consumption per key $\overline{S}/n$ decreases with increasing n, and for $n \geqslant 10^7$ we are close to the theoretical optimum for our parameter setting and compression scheme, i. e., we have $\overline{S}/n \approx 1.76$ bits per key. For fixed n the average space consumption increases with increasing κ, with a considerable effect for $n = 10^3, 10^4, 10^5$. This is because the space overhead is mainly due to the space consumption of the hash functions, or more precisely, their coefficients. The variance of the space consumption is only caused by the varying number of hash functions used in phase 0. For $n = 10^7$ this number is constant, i. e., the variance is

n	a	$\overline{i^0}$	$s^2[i^0]$	$\overline{i^1}$	$s^2[i^1]$	$\overline{i^2}$	$s^2[i^2]$	$\lvert\overline{r_{\alpha^*}} - \alpha^*\rvert$
10^3	10^5	1	0	1.25517	0.31776	95.00980	8961.13672	0.00265
10^4	10^4	1	0	1.00270	0.00269	24.34600	562.85537	0.00107
10^5	10^3	1	0	1	0	3.23700	7.30213	0.00016
10^6	10^2	1	0	1	0	1.01000	0.01000	0.00006
10^7	10^1	1	0	1	0	1	0	0.00002

(a) $\kappa = 2$

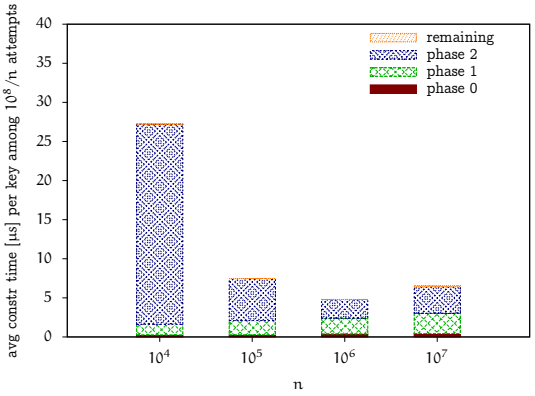
n	a	$\overline{i^0}$	$s^2[i^0]$	$\overline{i^1}$	$s^2[i^1]$	$\overline{i^2}$	$s^2[i^2]$	$\lvert\overline{r_{\alpha^*}} - \alpha^*\rvert$
10^3	10^5	1	0	1.25174	0.31485	95.05419	9010.90196	0.00261
10^4	10^4	1	0	1.00280	0.00279	23.95630	544.29362	0.00109
10^5	10^3	1	0	1	0	3.36200	7.69465	0.00022
10^6	10^2	1	0	1	0	1	0	0.00001
10^7	10^1	1	0	1	0	1	0	0.00001

(b) $\kappa = 6$

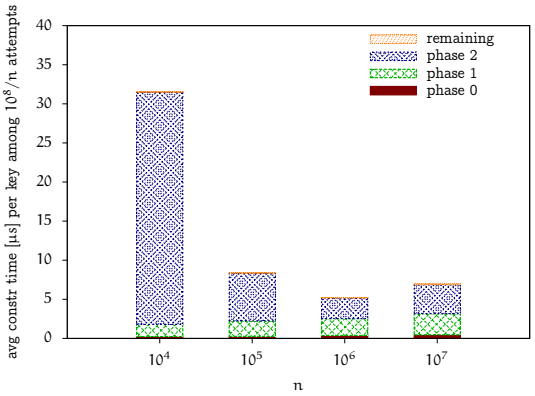
n	a	$\overline{i^0}$	$s^2[i^0]$	$\overline{i^1}$	$s^2[i^1]$	$\overline{i^2}$	$s^2[i^2]$	$\lvert\overline{r_{\alpha^*}} - \alpha^*\rvert$
10^3	10^5	1	0	1.25455	0.31732	95.49563	8957.40863	0.00265
10^4	10^4	1	0	1.00230	0.00229	24.22400	569.87581	0.00115
10^5	10^3	1	0	1	0	3.32000	7.72332	0.00017
10^6	10^2	1	0	1	0	1.01000	0.01000	0.00003
10^7	10^1	1	0	1	0	1	0	0.00001

(c) $\kappa = 10$

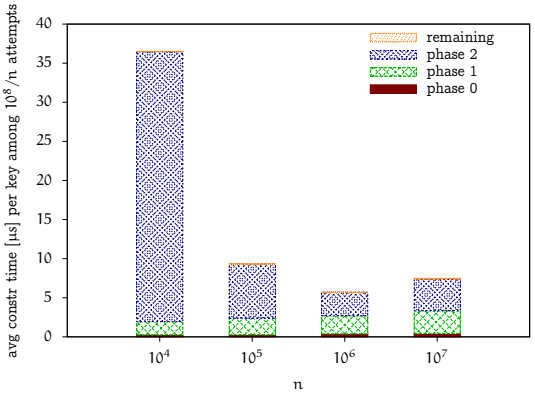
Table 4.6.5.: Average number of iterations per phase during construction of the perfect hash function. The values are rounded to five decimal places.



(a) $\kappa = 2$

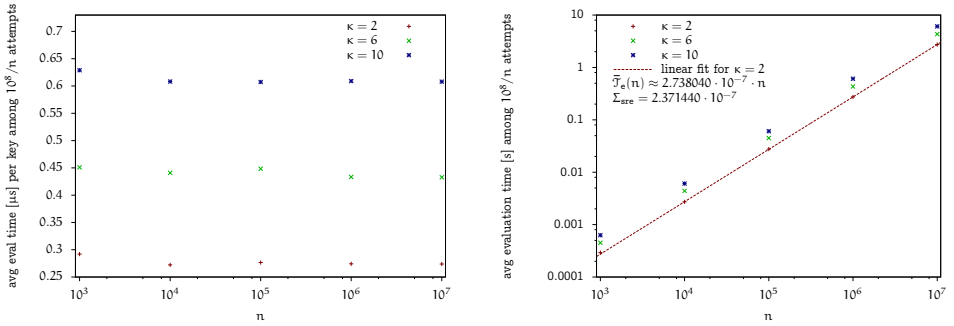


(b) $\kappa = 6$



(c) $\kappa = 10$

Figure 4.6.6.: Average construction time per phase and key of the perfect hash function. Ideally, the bars should have the same height.



(a) per key, $\log_{10}$ plot (b) overall, $\log_{10}$-$\log_{10}$ plot

Figure 4.6.7.: Average evaluation time of the perfect hash function. As predicted, the time consumption is approximately linear, since in Figure 4.6.7 (a) for each κ the corresponding measurement points are close to one line parallel to the abscissa.

		$\kappa = 2$		$\kappa = 6$		$\kappa = 10$	
n	a	$\bar{S}/n$	$s^2[S]/n^2$	$\bar{S}/n$	$s^2[S]/n^2$	$\bar{S}/n$	$s^2[S]/n^2$
10^3	10^5	11.61107	0.48924	16.73446	0.49940	21.85553	0.49157
10^4	10^4	2.86380	0.01157	3.37643	0.01201	3.88703	0.01121
10^5	10^3	1.89059	0.00015	1.94095	0.00016	1.99204	0.00016
10^6	10^2	1.77503	0.00000	1.78018	0.00000	1.78530	0.00000
10^7	10^1	1.76156	0	1.76207	0	1.76258	0

Table 4.6.6.: Average space consumption of the perfect hash function. The values are rounded to five decimal places. If one omits the lookup matrix $\mathbf{T}^{\text{dec}}$ for fast decompression, one can save up to $3888/n$ bits per key.

zero, since the key source U_w has size about 10^7 and therefore each key set $\check{S} \subseteq U_w$ got a key of maximum length.

4.7. Conclusion

Following [Mol04, Kim06, DGM+10], we *determined thresholds* $\check{c}_{\ell_+}(\mathbf{d}, \boldsymbol{\alpha})$ for the appearance of an ℓ_+-core in left-irregular bipartite graphs $G^{\mathbf{d}}_{n,m,\boldsymbol{\alpha}}$ or non-uniform hypergraphs $H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$, respectively. For the special case $\ell_+ = 2$ and $\mathbf{d} \in \mathbb{N}^2$, we showed how to analytically *obtain optimal vectors* $\boldsymbol{\alpha} = \boldsymbol{\alpha}^*$ that maximize the ratio n/m up to which a. a. s. the 2-core is empty. For larger vectors $\mathbf{d}$ as well as core parameter $\ell_+ > 2$ this is still open. It turned out that for some vectors $\mathbf{d} \in \mathbb{N}^2$ the optimal thresholds $\check{c}(\mathbf{d}, \boldsymbol{\alpha}^*) = \check{c}(\mathbf{d})$ are larger than the maximum 2-core threshold $\check{c}(3) \approx 0.818$ for regular bipartite graphs $G^d_{n,m}$ and uniform hypergraphs $H^d_{m,n}$. While experiments indicate

that using more irregularity allows to overcome the local optimum of $\check{c}((3, 21)) \approx 0.920$ found so far, it is open to prove or disprove the conjecture that the 2-core threshold can be made arbitrarily close to 1.

Using the results on optimal thresholds $\check{c}(\mathbf{d})$ for two different degrees as well as the generalized selfless algorithm from Section 3.4, we gave an *improved construction of a perfect hash function* with small range based on the irregular Bloomier filter. Using ideal auxiliary hash functions according to Section 2.2.2, this perfect hash function has expected linear construction time, constant evaluation time, and range $1.1 \cdot n$, as well as space consumption of about 1.76 bits per key. Moreover, we showed in experiments that by replacing the ideal hash functions with practical ones, this performance is also obtainable in real-life applications, making this algorithm one of the top perfect hash function constructions available.

5

Uniform Hashing

A hash function h is a function that maps keys from a universe U to some range R, where the hash values $h(x), x \in U$, are random in a certain way. In Chapters 3 and 4 we require the hash functions to be *fully random* on some fixed set S, $S \subseteq U$, which means that $(h(x))_{x \in S}$ must be *uniformly distributed* in $R^{|S|}$. Furthermore, we assume that the evaluation time of h is constant. How these assumptions can be justified using only linear space is discussed in the following.

5.1. Methods

In the literature there are basically three approaches for realizing uniform hashing in constant time:

Dictionary of Random Values — Given S, build a dictionary or retrieval data structure that stores for each $x \in S$ a fully random value from R.

Simulation of Full Randomness — Given n, build a data structure that represents a hash function that with high probability is uniform on *any* set S with $|S| \leqslant n$.

Split-and-Share — Split S into disjoint subsets S_i by a "splitting" hash function and modify the application such that each subset is handled independently of the other subsets. Generate a hash function with range R' that is fully random on any set of size $|\hat{S}|$, where $\hat{S}$ is one of the subsets S_i with maximum size. Share this hash function among the subsets S_i.

In this chapter we describe a randomized algorithm for the space-efficient simulation of full randomness, which internally uses the split-and-share approach — details to be discussed shortly.

Dictionary of Random Values It follows by a standard information theoretical argument, see Section 5.3.1, that any representation of a hash function h that is fully random on S needs at least $|S| \cdot \log|R|$ (random) bits in expectation. Hence in principle, one cannot do better than randomly choosing a hash value for each key from S and then use a dictionary to represent the function h given by the key-value pairs.

Examples for suitable (static) dictionaries with worst-case constant evaluation time are the famous dictionary by Fredman, Komlós, and Szemerédi [FKS84] using space $O\big(|S| \cdot (\log|U| + \log|R|)\big)$, and the dictionary by Pagh [Pag01a, Section 6], using space about[1] $\log \binom{|U|}{|S|} + |S| \cdot \log|R|$. The space usage for both is within a constant factor of the lower bound, under the condition that U has size polynomial in the size of R.

Simulation of Full Randomness In order to avoid building a dictionary with random function values for each given set S separately, one can apply an approach known as simulation of full randomness: Given an integer n, using randomization one builds a data structure $\mathfrak{D}_h$ that represents a hash function $h\colon U \to R$, such that for each set $S \subseteq U$ with $|S| \leqslant n$, there is a certain probability of failure, but if the failure event does not occur, then h is fully random on S. The advantage of this approach is that $\mathfrak{D}_h$ can be built in advance without knowledge of a concrete key set S, and then can be used by many applications. This flexibility comes at the price of introducing a failure probability as an additional performance characteristic beside evaluation time and space usage.

For the following discussion let $(R, \oplus)$ be an abelian group. Moreover, we assume that the size of U is polynomial in $|S| = n$, which can be justified via a standard method known as domain reduction that introduces some additional small but asymptotically negligible error probability, see, e. g., Section 5.4.1. The first randomized data structures for simulation of full randomness that use space essentially linear in the lower bound $n \cdot \log|R|$, have constant evaluation time, as well as high success probability were provided independently by Pagh and Östlin [ÖP03] and Dietzfelbinger and Woelfel [DW03]. The construction by Pagh and Östlin is based on Siegel's hash functions [Sie04], which add huge constants to the evaluation time, while the construction by Dietzfelbinger and Woelfel uses faster hash functions derived from a family of functions initially studied by Dietzfelbinger and Meyer auf der Heide [DM90, DM92]. Both approaches were later improved leading to two constructions with almost optimal space usage:

Let $\varepsilon > 0$ and $d > 0$, with $d \in \mathbb{N}$, be arbitrary but fixed.

- ▷ In [PP08] Pagh and Pagh reduced the space needs of the data structure from [ÖP03] further to $(1 + \varepsilon) \cdot n \cdot \log|R| + o(n) + O(\log\log|U|)$, while maintaining evaluation time $O(1/\varepsilon^2)$, and failure probability $O(1/n^d)$.
- ▷ Recently in [ADW12, Section 6] Aumüller, Dietzfelbinger and Woelfel gave a construction based on a generalization of the hash function family from [DW03] that has a space bound[2] of $2 \cdot (1 + \varepsilon) \cdot n \cdot \log|R| + o(n) + O(\log\log|U|)$, evaluation time $O(d)$, and failure probability $O(1/n^{d+1})$.

[1] A lower order term that depends on the size of U is ignored.
[2] This follows by [ADW12, Theorem 2] in combination with domain reduction.

SPLIT-AND-SHARE There is a simple method, known as split-and-share, that under some conditions makes it possible to circumvent the assumption that a hash function $h\colon U \to R$ is available that is fully random with respect to S. In this case, the required mapping can be realized in sublinear space. The approach was mentioned in [FPSS05, Section 6, suggestion by Dietzfelbinger], and was used in principle, e. g., in [HT01, Section 5]. Two application scenarios are described explicitly in [DW07, Section 2.3] and [Die07, Section 2], and a detailed discussion as well as a variety of different applications are given in [DR09].

The general idea is to use a hash function h^{split} in order to *split* the key set $S \subseteq U$ into disjoint subsets S_i. Let $\hat{S}$ be a largest of these subsets. Then one builds a hash function $h'\colon U \to R'$ that is fully random on each subset of U of size at most $|\hat{S}|$. For the construction of h' neither S needs to be given in advance, nor the subsets need to be explicitly determined. Both functions together, h^{split} and h', can be represented using space $o(|S| \cdot \log|R|)$ bits. Now, instead of building a hashing-based data structure for the whole set S, one builds a small hashing-based data structure for each S_i separately.

The crucial insight is that since h' is fully random on each subset of U that has size at most $|\hat{S}|$, it can be *shared* among all of these data structures.

RESULTS

Using the split-and-share approach, we give a construction of a randomized data structure $\mathfrak{D}_h$ for simulating full randomness that has asymptotically the same space bound as the construction given in [PP08], but faster evaluation time. Moreover, as in [ADW12], the use of Siegel's functions is avoided.

> THEOREM 5.1 (UNIFORM HASHING [DR09, THEOREM 1])
> *Let $(R, \oplus)$ be an abelian group. There is a randomized algorithm that on input $n \in \mathbb{N}$, $d \in \mathbb{N}$, and $\varepsilon > 0$ constructs a data structure $\mathfrak{D}_h$ that represents a function $h\colon U \to R$ with the following properties: For each set $S \subseteq U$ with $|S| \leqslant n$ there is a set $\mathcal{B}_S$ of "bad hash functions for S" such that*
>
> *(i) Under the condition that $h \notin \mathcal{B}_S$ the function h behaves fully randomly on S.*
>
> *(ii) The probability that $h \in \mathcal{B}_S$ is $O\left(n^{1-(d+2)/9}\right)$.*
>
> *(iii) The space usage for h is $(1+\varepsilon) \cdot n \cdot \log|R| + o(n) + O(\log\log|U|)$.*
>
> *(iv) The evaluation time of h is $O\big(\max\{\log^2(1/\varepsilon), d^2\}\big)$.*[3]
>
> *(v) The construction time of $\mathfrak{D}_h$ is $O(n)$, if $|R| \leqslant |U|$ and $\log|U| = O(2^{n^{1/4}}/n^{d/2})$.*

This data structure has the asymptotically best performance characteristic for the simulation of full randomness known so far.

[3] In [DR09, Theorem 1] the term $\log(1/\sigma)$ must be replaced by $\log^2(1/\sigma)$.

5.1.1. Overview of the Chapter

The subsequent section puts the result into a wider context. Afterwards, in Section 5.3, we state some definitions and facts that are presumed to be known in the proof of Theorem 5.1, which is then given in Section 5.4. The chapter finishes with some concluding remarks.

5.2. Further Background and Related Work

This section provides an overview of approaches for the analysis of hashing-based data structures. More precisely, we consider various randomness assumptions that are used in order to obtain provable performance bounds. Since it is of purely informational nature, it can be skipped by the expert reader.

Hashing-based data structure like those discussed in Chapters 3 and 4 are built for a set of (known or unknown) keys $S \subseteq U$. In order that the construction of such a data structure is successful or to meet specific performance requirements for its operations, respectively, the hash functions must be "well-behaved". Beside fast evaluation time and small space consumption one usually would like the hash values to satisfy certain randomness properties. A general approach based on [CW79] is to choose a hash function

$$h\colon U \to R$$

from a family of hash functions, usually called *hash class*. Let $\mathcal{H} = \{h_i\}_{i\in I}$ be such a hash class with index set I and $h_i\colon U \to R$ for all $i \in I$. Choosing a function from $\mathcal{H}$ translates into choosing an index i from I uniformly at random. Since $\mathcal{H}$ is a family of hash functions, different probabilities concerning the choice of a function can easily be modeled by the multiplicity of these functions within the hash class.

Two scenarios are discussed in the analysis of hashing-based data structures:

1. Keys are considered as random variables and $|\mathcal{H}| \geqslant 1$. (average case)
2. Keys are arbitrary, but fixed and $|\mathcal{H}| \gg 1$. (worst case)

When using average-case analysis in the first case, one usually tries to avoid strong randomness assumptions concerning the keys since they are considered to be unrealistic. The second case, i. e., the worst-case analysis on the keys, is by far the most common one. We give a general survey of both approaches in the following.

5.2.1. Keys are Random Variables

Building upon the work on *randomness extractors* [CG85, CG88, PRB-source, Definition 1 and 2] a standard model [MV08, CV08] assumes that the key set S is a set of random variables $X_0, X_1, \ldots, X_{n-1}$ which form a *block source*. That means

for each $i \in [n]$ and arbitrary $x_0, x_1, \ldots, x_{i-1}$ the variable X_i restricted to the event $\{X_0 = x_0 \wedge X_1 = x_1 \wedge \ldots \wedge X_{i-1} = x_{i-1}\}$ has at least l bits of *collision entropy* or *min-entropy*.[4] It was shown that if a choice of h gives that each sequence of κ pairwise distinct random variables from the sequence $(h(X_i))_{i\in[n]}$ is uniformly distributed, and if l is sufficiently large, then the distribution of the whole sequence $(h(X_i))_{i\in[n]}$ is close to the uniform distribution in terms of *collision probability* or *statistical distance*[5]. Under these assumptions the performance of several hashing-based data structures is asymptotically the same as when using a fully random hash function, like, e. g., the expected lookup time of linear probing [MV08, Theorem 5.1]. Typically, κ is assumed to be small, e. g., $\kappa = 2$, and the range of h is assumed to be linear in n. Under these conditions it is sufficient if l is about $\log n$ times a small constant that depends on the specific application, see [CV08, Tables 1 and 2].

5.2.2. Keys are Arbitrary but Fixed

There are three different but related approaches to this situation:

Universal Hashing — an influential paradigm suggested by Carter and Wegman [CW77, CW79]. Here one identifies hash classes where randomly choosing a hash function ensures that the sequence $(h(x))_{x\in U}$ satisfies certain randomness properties, like κ-*wise independence* [WC79, WC81, strongly universal], which means that each sequence of κ pairwise distinct random variables $h(x)$ is uniformly distributed in R^κ. Then in the analysis of the performance of a hashing-based data structure one uses only these properties, which can be seen as applying some abstract hash class where the specific structure of the function h is irrelevant.

Full Randomness — the ideal situation and in a certain sense the extreme case of universal hashing. This is the classical and most convenient approach. Here one assumes that $(h(x))_{x\in U}$ is uniformly distributed in $R^{|U|}$, or with other words, h is $|U|$-wise independent, which can be modeled by defining $\mathcal{H}$ as the set of all functions from U to R.

Remark. Often it is sufficient to assume full randomness solely for the hash values $h(x), x \in S$, where S is the set of keys that occur in an application.

Specific Hashing — In contrast to universal hashing, one utilizes the specific structure of the functions of $\mathcal{H}$ for analyzing a hashing-based data structure.

Universal Hashing Consider the examples of hash tables for dictionaries from Section 3.2. Let $|S| = n$ be the number of keys and let $|R| = m$ be the size of the table.

[4] The collision entropy $-\log\left(\sum_x \Pr(X = x)^2\right)$ is greater than or equal to the min-entropy $\min_x -\log\left(\Pr(X = x)\right)$.

[5] The collision probability is $\sum_x \Pr(X = x)^2$, and the statistical distance between two random variables X_i and X_j is $\sum_x |\Pr(X_i = x) - \Pr(X_j = x)|$.

The construction of a hash table applying *chaining* works with every function h that maps from U to $[m]$, but for the lookup of a key-value pair the linked lists should be short, at least on average. To bound the expected lookup time for a key x, it suffices to bound the expected number of keys y that *collide* with x, i. e., have the same hash value $h(x) = h(y)$. Hence, already with only 2-wise independence one can show an expected time of $O(1 + n/m)$ for a lookup (successful and not successful).

Similarly, a construction of a hash table with *linear probing* is successful regardless of the hash function h, as long as $n < m$, but now the insertion time depends on the lookup time. A classical result[6] due to Knuth [Knu63],[Knu98, page 528] bounds the expected number of comparisons for unsuccessful lookup and insertion, respectively, to about $1/2 \cdot (1 + (1 - n/m)^{-2})$ and for successful lookup to about $1/2 \cdot (1 + (1 - n/m)^{-1})$. His analysis assumes a fully random hash function. Hence, a natural question by Carter and Wegman was how linear probing works with κ-wise independence [CW77, page 111(6)][7]. For n/m bounded away from 1, Pagh, Pagh, and Ružić proved that 5-wise independence is sufficient to guarantee a constant number of expected comparisons [PPR07, PPR09], where the difference to full randomness is only by a factor of $\Omega((1 - n/m)^{-1})$.

The advantage of *cuckoo hashing* and its variants is that the number of cell probes or comparisons, respectively, for lookup is constant in the worst-case. The price for this is that a successful construction of such hash tables is not possible with every choice of hash functions. Let $m = (2 + \varepsilon) \cdot n$ for some $\varepsilon > 0$, then $O(\log(n))$-wise independence is sufficient for cuckoo hashing to work with high probability [Pag01b, PR04]. Hash classes whose functions give such a high degree of independence and simultaneously have constant evaluation time and relatively small space needs are highly nontrivial. The first constructions of such *high performance* hash classes were given by Siegel [Sie89, Sie04].

Specific Hashing Unfortunately, in terms of κ-wise independence (and related properties), in general one cannot define *necessary* conditions to gain specific performance results. For example, if n/m is constant, then linear probing has a constant number of expected comparisons if one uses a common, practical *tabulation* based 3-wise independent hash class [PT11, PT12]; however there exists an artificial 4-wise independent hash class, such that the expected search time is $\Omega(\ln(n))$ [PT10]. Similarly, cuckoo hashing works with w. h. p. using a hash class that combines κ-wise independence with tabulation [DW03, Section 4.1], [Aum10, Corollary 5.10] and works w. h. p. even with a simple tabulation based 3-wise independent hash class [PT12, Theorem 1.2]; but fails w. h. p. if one uses a pair of artificial 5-wise independent hash classes [CK09] and "the number of keys n is large enough", and also fails w. h. p. if one uses a common 2-wise independent hash class and "the set of keys is relatively dense in the universe" [DS09, Sch09][8].

[6] This results is considered as the beginning of the area of analysis of algorithms, see, e. g., [PS98].

[7] Probably Carter and Wegman used the term *open addressing* for *linear probing* like Knuth in [Knu63].

[8] The paper also covers another hash class called *multiplicative class* with weaker random properties.

In recent years the focus on analyzing specific hashing-based data structures using specific hash classes increased and recent results show the power of hash classes that use tabulation, in a simple form, see, e. g., [TZ04, TZ10, TZ12, PT12, PT13], as well as in a more involved form [DW03, Aum10, ADW12] that is based on a variant of the hash class from [DM90].

Full Randomness After discussing the examples above, it could seem that the full randomness assumption will soon become obsolete with regard to the progress in the analysis of randomized algorithms and data structures. While this would be a preferable development, unfortunately there are still many hashing-based data structures where the analysis assumes full randomness to gain desired performance characteristics. Notably this includes data structures that apply more than one hash function and rely on certain properties of random graphs or hypergraphs at their core — which is exactly the case in Chapters 3 and 4. Examples are dictionaries (and membership tester) based on generalized cuckoo hashing with at least three hash functions [FPSS05, FM09, FP10, DGM+10], cells that are grouped together into (disjoint or non-disjoint) blocks of size at least two [CSW07, FR07, DW07, LP09, Bey12], as well as their combination [GW10, FKP11, Lel12a, PS12]; also retrieval data structures (and perfect hash functions) [CKRT04, BPZ07, Die07, DP08b, BBD09]; as well as modern Bloom filter based data structures [GM11, MV12].

Recent results from [MT12] indicate that the amount of randomness needed for such data structures that are based on properties of cores of random hypergraphs can be reduced by replacing $d \geqslant 3$ fully random hash functions by only two fully random hash functions and determine the hash values using *double hashing*, see, e. g., [CLRS01, page 240]. But as discussed in Section 5.1, providing or simulating even only one fully random hash function is expensive [ÖP03, DW03, PP08, DR09, ADW12], especially in terms of space consumption. However, often the split-and-share approach can be applied in order to reduce the amount of additional space to a sublinear number of words, and simultaneously justifying the assumption of fully random hash values, see, e. g., [DR09].

5.3. Basics

In this section we introduce some definitions and recall basic facts related to hashing.

5.3.1. Full Randomness

For the analysis of hashing-based data structures it is very convenient to assume ideal hash functions in the following sense.

Definition 10: (Fully Random Hash Function)
A hash function $h\colon U \to R$ determined by some random experiment is called *fully random on* S for some $S \subseteq U$, if the hash values $h(x), x \in S$, are independent random variables that are uniformly distributed in R. If $S = U$, then the hash function is simply called *fully random*.

The minimum expected number of bits needed by any method for writing down a hash function that is fully random on S, even if we ignore the evaluation time, is essentially the space usage of a list of $|S|$ random hash values.

Lemma 5.3.1 (Information Theoretical Optimum). Let $h\colon U \to R$ be fully random on $S \subseteq U$, then the following holds:

(i) The expected space usage for h is at least $|S| \cdot \log|R|$ bits.

(ii) The expected number of *random bits* for constructing h is at least $|S| \cdot \log|R|$.

Proof. Let $|S| = n$. If h is fully random on S, then the sequence $(h(x))_{x \in S}$ is a random variable Y, uniformly distributed in the range R^n.

ad (i): The expected number of bits to represent a realization y of Y is at least $n \cdot \log|R|$. One way to derive this result is to choose an arbitrary but fixed binary representation (pairwise distinct) for each $y \in R^n$, and build a binary tree with exactly $|R|^n$ leaves, where each leaf is labeled with one $y \in R^n$ (pairwise distinct) and each edge is labeled with 0 or 1 in such a way that each sequence of bits along the path from the root of the tree to a leaf y is equal to the binary representation of the leaf. Let L be a random variable that denotes the length (number of bits) of such a sequence or path for Y. Then the expected length $\mathrm{Exp}(L)$ is bounded from below by the entropy of Y, see, e. g., [CT06, Theorem 5.3.1 combined with Theorem 5.5.1], which is $\sum_{y \in R^n} |R|^{-n} \cdot \log\left(|R|^n\right)$.

ad (ii): The lower bound of the expected number of random bits for determining a realization of Y is also at least $n \cdot \log|R|$. The idea behind this is as follows: Consider a (possibly infinite) binary decision tree with at least $|R|^n$ leaves or outcomes, respectively. If the path from the root to a leaf has length l, then the number of random bits used for this outcome is l and its probability is 2^{-l}. Now each leaf of the tree is labeled with one realization $y \in R^n$ (duplicates allowed) such that the sum of the probabilities of the leaves with label y is $|R|^{-n}$. Let L be a random variable that denotes the length of the random decision path for Y. Then, as above, the expected length $\mathrm{Exp}(L)$ is bounded from below by the entropy of Y, see, e. g., [CT06, Theorem 5.11.1].

5.3.2. Universal Hashing

A common way to circumvent the assumption of fully random hash functions is by the use of universal hashing [CW79]. Here a hash function h is (efficiently) chosen from

a class of hash functions $\mathcal{H}$, where depending on $\mathcal{H}$ the random choice of h implies certain randomness properties for $(h(x))_{x\in U}$.

> Definition 11: (Hash Class)
> An indexed family $\mathcal{H} = \{h_i\}_{i\in I}$ of functions h_i from $\{h \mid h\colon U \to R\}$ is called a *hash class*. Randomly choosing a function from $\mathcal{H}$ is done via selecting its index i from the index set I uniformly at random.

Usually a hash classes $\mathcal{H}$ is represented by some uninitialized data structure $\mathfrak{D}_{\mathcal{H}}$, and choosing a hash function h from $\mathcal{H}$ translates into randomly initializing the corresponding data structure $\mathfrak{D}_{\mathcal{H}}$ to $\mathfrak{D}_h$, the representation of h.

A property of the sequence $(h(x))_{x\in U}$ that is often assumed is that sub-sequences of $(h(x))_{x\in U}$ of length at most κ are independent and (almost) uniformly distributed.

> Definition 12: ((ε,κ)-Wise Independence)
> A hash class $\mathcal{H}$ with functions from $\{h \mid h\colon U \to R\}$ is called (ε,κ)-*wise independent*, if for each sequence $x_0, x_1, \ldots, x_{\kappa-1}$ of pairwise distinct elements from U, all sequences $y_0, y_1, \ldots, y_{\kappa-1}$ of elements from R, and h randomly chosen from $\mathcal{H}$ we have
>
> $$\Pr\left(h(x_0) = y_0 \wedge \ldots \wedge h(x_{\kappa-1}) = y_{\kappa-1}\right) \in \left[(1-\varepsilon)\cdot|R|^{-\kappa}, (1+\varepsilon)\cdot|R|^{-\kappa}\right] .$$
>
> If $\varepsilon = 0$ then $\mathcal{H}$ is simply called κ-wise independent.

Classical (ε,κ)-wise independent hash classes, which go back to Wegman and Carter [WC81, Section 1, strongly universal], are based on polynomials of degree at most $\kappa-1$ over some field. Examples are given in the following.

Let $U = \mathbb{F}_{p^k}$ be a field of characteristic p and let $R = (\mathbb{F}_p)^l$, where $l \leqslant k$. Furthermore let π be a vector space projection of $(\mathbb{F}_p)^k$, which is isomorphic to U, onto R. Then the hash class $\mathcal{H} = \{h_a\}_{a\in U^\kappa}$ of functions mapping from U to R via

$$h_{(a_0,a_1,\ldots,a_{\kappa-1})}(x) \coloneqq \pi\left(\sum_{i=0}^{\kappa-1} a_i \cdot x^i\right)$$

is κ-wise independent.

More flexibility for the size of the range R allows the following standard construction. Let $U = [|U|]$ and $R = [|R|]$, where $|U| \leqslant p$ for some prime number p, and $|R| \leqslant |U|$. Then the hash class $\mathcal{H} = \{h_a\}_{a\in[p]^\kappa}$ of functions mapping from U to R via

$$h_{(a_0,a_1,\ldots,a_{\kappa-1})}(x) \coloneqq \left(\left(\sum_{i=0}^{\kappa-1} a_i \cdot x^i\right) \bmod p\right) \bmod |R| ,$$

is (ε,κ)-wise independent for $\varepsilon \leqslant \frac{\kappa\cdot(|R|-1)}{p-\kappa\cdot(|R|-1)}$, if $\kappa\cdot|R| - \kappa < p$. This is because the projection via mod $|R|$ gives that the probability $\Pr\left(\bigwedge_{i\in[\kappa]} h(x_i) = y_i\right)$ is from the

interval

$$\left[\left(\frac{1}{p}\cdot\left\lfloor\frac{p}{|R|}\right\rfloor\right)^{\kappa},\left(\frac{1}{p}\cdot\left\lceil\frac{p}{|R|}\right\rceil\right)^{\kappa}\right]\subseteq\left[\left(\frac{p-|R|+1}{p}\right)^{\kappa}\cdot|R|^{-\kappa},\left(\frac{p+|R|-1}{p}\right)^{\kappa}\cdot|R|^{-\kappa}\right],$$

and then the bound for ε follows by Bernoulli's inequality and the inequality $e^z \leqslant \frac{1}{1-z}$, for $z < 1$. Usually p is much larger than $|R| \cdot \kappa$, and in this case the difference to exact κ-wise independence is negligible. Storing such a function h translates into storing the κ coefficients $a_0, a_1, \ldots, a_{\kappa-1}$ from U, as well as the prime number p, and the size of R. According to Bertrand's postulate, see, e. g., [AZ10, Chapter 2], there is always a prime number between $|U|$ and $2 \cdot |U|$. Hence h can be represented using space $O(\kappa \cdot \log|U| + \log|R|)$; the evaluation time is $O(\kappa)$. If one wants to store more than one function, then p and $|R|$ need to be stored only once since they are fixed for all functions from the hash class.

Slightly weaker than κ-wise independence is the assumption that the collision probability of at most κ elements from $(h_x)_{x\in U}$ is (almost) the same as for fully random elements.

Definition 13: ((ε,κ)-Universality)
A hash class $\mathcal{H}$ of functions from $\{h \mid h\colon U \to R\}$ is called (ε,κ)-*universal*, if for each sequence $x_0, x_1, \ldots, x_{\kappa-1}$ of pairwise distinct elements from U, and h chosen uniformly at random from $\mathcal{H}$ we have

$$\Pr\left(h(x_0) = h(x_1) = \ldots = h(x_{\kappa-1})\right) \leqslant (1+\varepsilon)\cdot|R|^{-\kappa+1}.$$

If $\varepsilon = 0$ then $\mathcal{H}$ is simply called κ-universal.

A hash class that is κ-wise independent is also κ-universal, but the inverse does not hold in general.

A common universal hash class that has small description size is the following. Let $U = [|U|]$ and $R = [|R|]$ with $|R| \leqslant |U|$. Furthermore let $b = 6 \cdot |R| \cdot \ln|U|/\varepsilon$, for $b \geqslant 2$, and let P be the set of primes between b and $2 \cdot b$. Then the hash class $\mathcal{H} = \{h_i\}_{i\in I}$ with $I = \{(a_0, a_1, p) \mid p \in P, a_0 \in [p], a_0 \neq 0, a_1 \in [p]\}$ of functions mapping from U to R via

$$h_{(a_0,a_1,p)}(x) := \big((a_0 + a_1 \cdot x) \bmod p\big) \bmod |R|,$$

is $(\varepsilon, 2)$-universal, see, e. g., [Sie04, Appendix 1, Facts 1 and 2] and [Pag09, variant of Theorem 14]. The space usage is $O(\log|R| + \log\log|U| + \log(1/\varepsilon))$, the evaluation time is constant. The smaller space usage, compared to the functions above, comes at the price that beside the coefficients a_0 and a_1 now also p must be randomly chosen from P. According to the inequality of Finsler, see, e. g., [Sie88, page 158], it holds that P has size at least $\frac{b}{3\cdot\ln(2\cdot b)}$.

A standard way for selecting p is to randomly choose several odd integers from the interval $[b, 2 \cdot b]$ and then repeatedly apply the Miller-Rabin primality test on each of

these samples, see, e. g., [CLRS01, pages 890–896]. There is a non-zero probability that the number that is returned by the algorithm is not a prime from P, and in this case h is not from $\mathcal{H}$. However, this probability can be made vanishingly small using an appropriate number of samples and repetition count. For example, if $b \leqslant 2^{n^{1/4}}$, then one can achieve a failure probability of at most $2^{-n^{1/4}}$ together with a running time of $O(n)$, see, e. g., [DHKP97, Lemma 2.9].

5.3.3. Sequences of Hash Values

Hashing-based data structures that apply the *multiple choice paradigm*, like cuckoo hashing and Bloomier filter, map each key to a sequence of hash values. For ease of discussion we only consider sequences of integers from $[m]$, i. e., mappings of the form

$$h\colon U \to [m]^d\,.$$

5.3.3.1. Duplicates Allowed

If duplicate hash values are allowed, then one can use d hash functions

$$h_0, h_1, \ldots, h_{d-1}\colon U \to [m]\,,$$

and define h via

$$h\colon x \mapsto \big(h_0(x), h_1(x), \ldots, h_{d-1}(x)\big)\,.$$

If the d hash functions h_i are fully random and independent, then h is also fully random.

5.3.3.2. Pairwise Distinct Elements

Sometimes the values of h are restricted to sequences with pairwise distinct elements from $[m]$. Using d hash functions

$$\tilde{h}_i\colon U \to [m-i] \text{ for all } i \in [d],$$

whose ranges successively decrease by one, h can be defined via the following standard approach, see, e. g., [Die07, Section 4.1]. The first element of $h(x)$ is defined as $\tilde{h}_0(x)$. If $\tilde{h}_0(x) \neq m-1$, then in the ranges of $\tilde{h}_1, \tilde{h}_2, \tilde{h}_3, \ldots, \tilde{h}_{d-1}$ the element $\tilde{h}_0(x)$ is substituted by $m-1$. The next element of $h(x)$ is then defined as $\tilde{h}_1(x)$. If $\tilde{h}_1(x) \neq m-2$, then in the ranges of $\tilde{h}_2, \tilde{h}_3, \ldots, \tilde{h}_{d-1}$ the element $\tilde{h}_1(x)$ is substituted by $m-2$. The third element of $h(x)$ is then $\tilde{h}_2(x)$ and so forth. More formally consider the recursive function

$$\iota(a, x, i) = \begin{cases} a\,, & \text{if } \hat{\jmath} \coloneqq \sup\{j \mid j \in [i], a = \tilde{h}_j(x)\} = -\infty \\ \iota(m-\hat{\jmath}-1, x, \hat{\jmath})\,, & \text{otherwise}\,. \end{cases}$$

Using $\iota(a, x, i)$ we define d functions $h_i(x), i \in [d]$, according to

$$h_i(x) \coloneqq \iota(\tilde{h}_i(x), x, i)\,,$$

which gives a mapping

$$h\colon x \mapsto \big(h_0(x), h_1(x), \ldots, h_{d-1}(x)\big)\,,$$

as required. If the d hash functions $\tilde{h}_i$ are fully random and independent, then h is also fully random. In order to efficiently realize h we use Algorithm 16, which on input x calculates $h(x)$ via evaluating each function $\tilde{h}_i, i \in [d]$, only once and tracing the substitution of the hash values using some dictionary $\mathfrak{D}$.

ALGORITHM 16: `pairwise_distinct_hash_values`

Input : Element x from U.
Output : Sequence $\mathbf{a}$ from $[m]^d$, where elements of $\mathbf{a}$ are pairwise distinct.
Require : Dictionary $\mathfrak{D}$ with $\texttt{lookup}(\texttt{insert}(\mathfrak{D}, a, w), a) = w$ and $\texttt{lookup}(\mathfrak{D}, a) = a$ if no key-value pair $(a, \star)$ was inserted. Hash functions $\tilde{h}_i\colon U \to [m - i], i \in [d]$.

```
function h(x)
    D ← empty();
    for i ← 0 to d − 1 do
        a ← h̃_i(x);                 b ← m − (i + 1);
        v ← lookup(D, a);           w ← lookup(D, b);
        D ← insert(D, a, w);        D ← insert(D, b, v);
    end
    return (lookup(D, m − 1), lookup(D, m − 2), ..., lookup(D, m − d));
end
```

If one uses for example a balanced binary search tree for the implementation of $\mathfrak{D}$, then in addition to space and time consumption of $\tilde{h}_i, i \in [d]$, the algorithm requires space $O\big(d \cdot (\log|U| + \log m)\big)$ and evaluation time $O(d \cdot \log d)$. Alternatively, one could use a hash table to implement $\mathfrak{D}$, for example chained hashing with a table of size n^δ, $\delta > 0$, and a random hash function h from a simple $(1, 2)$-universal hash class. With probability at least $1 - O(d^2/n^\delta)$ the function h is injective on the inputs that occur during the execution of Algorithm 16, see Section 5.4.1, which in this case leads to an additional evaluation time of $O(d)$, while the additional space usage is $O\big(n^\delta \cdot (\log|U| + \log m)\big)$.

5.4. Uniform Hashing in Close to Optimal Space

In this section we prove Theorem 5.1 — most of this proof appeared in [DR09]. We show how to construct a hash function $h\colon U \to R$ that, with high probability, is fully

random on an arbitrary key set $S \subseteq U$ with size at most n, where h uses space only by a factor $(1+\varepsilon)$ larger than the information theoretic optimum, and has evaluation time $O(\log^2(1/\varepsilon))$, for an arbitrary constant $\varepsilon > 0$. The construction is inspired by the construction of the basic retrieval data structure in [DP08b, Section 2.2]. But in contrast to [DP08b] where the set S is known, our approach must work for an arbitrary, unknown set S.

Let $S \subseteq U$ be fixed. If the universe has size super-polynomial in n, then we use a randomly chosen function h^{coll} from some hash class $\mathcal{H}^{\text{coll}}$ in order to reduce its size to n^κ, for some $\kappa \in \mathbb{N}$ determined later. We require that no pair of distinct keys from S collide under h^{coll}, since otherwise h would not be fully random on S.

Lemma 5.4.1 (Collapsing the Universe). Let U be of size super-polynomial in n. For arbitrary constant $\kappa \in \mathbb{N}$, with $\kappa \geqslant 3$, there is a hash class $\mathcal{H}^{\text{coll}}$ of functions mapping from U to $[n^\kappa]$, such that for each $S \subseteq U$ with $|S| = n$ and h randomly chosen from $\mathcal{H}^{\text{coll}}$ the following holds. There is a set $\mathcal{B}_S$ of "bad hash functions for S" such that:

(i) Under the condition that $h \notin \mathcal{B}_S$ the function h is injective on S.

(ii) The probability that $h \in \mathcal{B}_S$ is $O(n^{2-\kappa}) + 2^{-n^{1/4}}$.

(iii) The space usage for h is $O(\log n + \log\log|U|)$.

(iv) The evaluation time of h is $O(1)$.

(v) The construction time of h is $O(n)$, under the condition that $\log|U|$ has size $O(2^{n^{1/4}}/n^\kappa)$.

The proof is given in Section 5.4.1. From now on we assume that U has size polynomial in n. The next step is to split S into $t = n^{1-\delta}$, $\delta \in (0,1)$, disjoint subsets $S_0, S_1, \ldots, S_{t-1}$ of almost the same size without knowledge of S, i. e., without explicitly building these subsets. To achieve this, we randomly choose a hash function h^{split} from a certain hash class $\mathcal{H}^{\text{split}}$ and define

$$S_i = \{x \in S \mid h^{\text{split}}(x) = i\} \text{ for all } i \in [n^{1-\delta}].$$

The following lemma states properties of h^{split} and S_i.

Lemma 5.4.2 (Splitting the Key Set Evenly). Let U be of size polynomial in n. For arbitrary constant δ, α, λ, with $\delta \in (0,1)$, $\alpha \in (\delta, 1)$, $\lambda > 0$, as well as arbitrary constant $\kappa \in \mathbb{N}$, with $\kappa \geqslant 1$, there is a hash class $\mathcal{H}^{\text{split}}$ of functions mapping from U to $[n^{1-\delta}]$ such that for each $S \subseteq U$ with $|S| = n$ and h randomly chosen from $\mathcal{H}^{\text{split}}$ the following holds. There is a set $\mathcal{B}_S$ of "bad hash functions for S" such that:

(i) Under the condition that $h \notin \mathcal{B}_S$ all sets $S_i = \{x \in S \mid h(x) = i\}$, $i \in [n^{1-\delta}]$, have size at most $(1+\lambda) \cdot n^\delta$.

(ii) With probability at most

$$\left(\frac{e}{\kappa+1}\right)^{\kappa+1} \cdot n^{1-(\alpha-\delta)\cdot\kappa} + n^{1-\delta} \cdot \left(\frac{e^{\lambda}}{(1+\lambda)^{1+\lambda}}\right)^{n^{\delta}/\kappa}$$

we have that $h \in \mathcal{B}_S$.

(iii) The space usage for h is $O\big((n^{\alpha}+\kappa)\cdot\log n\big)$.

(iv) The evaluation time of h is $O(\kappa)$.

(v) The construction time of h is $O(n^{\alpha}+\kappa)$.

The proof of this lemma is given in Section 5.4.2. For the moment we assume all subsets S_i have size at most $(1+\lambda)\cdot n^{\delta}$. First we focus on one of these subsets S_i for fixed i. We want to obtain a hash function h^{map} that fully randomly maps the elements x of S_i to d-element subsets of $[m_i]$, for some $m_i \in \mathbb{N}$ determined later. It is easy to construct an appropriate hash class $\mathcal{H}^{\text{map}}$ if one allows spending many more random words than the size of S_i in order to store a function h^{map} from $\mathcal{H}^{\text{map}}$.

Lemma 5.4.3 (Fully Random Mapping on a Small Key Set). Let U be of size polynomial in n. For arbitrary constant δ, α, with $\delta \in (0,1)$, $\alpha \in (\delta, 1)$, as well as arbitrary constant $\kappa, d \in \mathbb{N}$, with $d \geqslant 1$ and $\kappa \geqslant 1$, there is a hash class $\mathcal{H}^{\text{map}}$ of hash functions mapping from U to $\binom{[m]}{d}$ such that for each $S \subseteq U$ with $|S| = 2\cdot n^{\delta}$ and h randomly chosen from $\mathcal{H}^{\text{map}}$ the following holds. There is a set $\mathcal{B}_S$ of "bad hash functions for S" such that:

(i) Under the condition that $h \notin \mathcal{B}_S$ the function h is fully random on S.

(ii) The probability that $h \in \mathcal{B}_S$ is at most $\left(\frac{2\cdot e}{\kappa+1}\right)^{\kappa+1} \cdot n^{\delta-(\alpha-\delta)\cdot\kappa}$.

(iii) The space usage for h is $O\big(d\cdot(\kappa\cdot n^{\alpha}\cdot\log n + \log m)\big)$.

(iv) The evaluation time of h is $O\big(d\cdot(\kappa+\log d)\big)$.

(v) The construction time of h is $O(d\cdot\kappa\cdot n^{\alpha})$.

The proof of this lemma is given in Section 5.4.3. Still focusing on S_i, we assume that h^{map} is fully random on this set. Let $|S_i| = n_i$ and for each $x \in S_i$ let $\mathbf{b}_x^{\intercal} \in \{0,1\}^{m_i}$ be the characteristic row vector of the set $h^{\text{map}}(x) \in \binom{[m_i]}{d}$. The next lemma states that if m_i and d are chosen appropriately, then the multiset $\bigcup_{x\in S_i}\{\mathbf{b}_x\}$ is linearly independent with high probability, i. e., the matrix $\mathbf{M}_i = (\mathbf{b}_x^{\intercal})_{x\in S_i} \in \{0,1\}^{n_i\times m_i}$ has *full row rank*. This is not surprising since $\mathbf{M}_i$ is identical to $\mathbf{M}^{d}_{n_i,m_i}$ type B and, according to Theorem 4.3, the matrix $\mathbf{M}_i$ has *asymptotically almost surely* full row rank, if $c_i = n_i/m_i < \hat{c}(d) - \varepsilon$ for arbitrary constant $\varepsilon > 0$. However, for our proof of Theorem 5.1 we need that $\mathbf{M}_i$ has full row rank *with high probability*. In order to obtain such a probability bound, we consider an upper bound $\hat{c}'(d)$ for the load factor c_i somewhat below the threshold $\hat{c}(d)$.

Lemma 5.4.4 (Lower Bounds on the Number of Vectors for Linear Dependence [Cal97, Theorems 1.1 and 1.2, with better error bound]). Let $B_d^m \subset \{0,1\}^m$ be the set of binary vectors of length m with exactly d ones and $m-d$ zeros. For any constant $d \geqslant 8$ there is a constant $\hat{c}'(d)$ such that the following holds for a sequence of $n = c \cdot m$ binary vectors $\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_{n-1}$ chosen uniformly at random from B_d^m:

(i) If $c < \hat{c}'(d)$, then the vectors $\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_{n-1}$ are linearly independent with probability $O\left(n^{1-(d+2)/9}\right)$.

(ii) The constant $\hat{c}'(d)$ can be chosen at least as large as $1 - 2 \cdot e^{-d}/\ln 2$.

This lemma is proved in Section 5.4.4. Using h^{map} and a fully random column vector $\mathbf{t}_i = (t_{i,0}, t_{i,1}, \ldots, t_{i,m-1})^\intercal \in R^m$, we construct a function $h_i \colon U \to R$ via

$$h_i(x) \coloneqq \mathbf{b}_x^\intercal \cdot \mathbf{t}_i, \text{ where } \mathbf{b}_x^\intercal \text{ is the characteristic row vector of } h^{\text{map}}(x).$$

This is equivalent to

$$h_i(x) = t_{i,a_0} \oplus t_{i,a_1} \oplus \ldots \oplus t_{i,a_{d-1}}, \text{ where } h^{\text{map}}(x) = \{a_0, \ldots, a_{d-1}\}.$$

Under the condition that $\mathbf{M}_i = (\mathbf{b}_x^\intercal)_{x \in S_i}$ has full row rank, it follows that $h_i(x)$ is fully random on S_i as stated in the following lemma.

Lemma 5.4.5 (Linear Independence Implies Stochastic Independence). Let $(R, \oplus)$ be an abelian group, and let $\mathbf{M} \in \{0,1\}^{n \times m}$ be a binary matrix with full row rank, as well as let $\mathbf{t}$ be a random column vector from R^m. Then the vector $\mathbf{M} \cdot \mathbf{t}$ is uniformly distributed in R^n.

The proof of this lemma is given in Section 5.4.5. Up to now, we have constructed a hash function h_i that is fully random on one S_i.

The last step is to *replicate* the construction described so far *for all* subsets S_i with $i \in [n^{1-\delta}]$. The crucial observation is that the function h^{map} can be *shared* among all subsets S_i, since its domain is U. The probability that h^{map} fails on at least one of these subsets is at most $n^{1-\delta}$ times the probability that it fails on a fixed subset. The vector $\mathbf{t}_i$ cannot be shared, and we need a separate random vector from R^m for each subset S_i, i. e., vectors $\mathbf{t}_0, \mathbf{t}_1, \ldots, \mathbf{t}_{n^{1-\delta}-1}$. Finally, the desired uniform hash function $h \colon U \to R$ is defined as

$$h(x) \coloneqq \bigoplus_{a \in h^{\text{map}}(x')} t_{h^{\text{split}}(x'),a} \text{ for all } x \in U, \text{ where}$$

$$x' = \begin{cases} h^{\text{coll}}(x), & \text{if domain reduction is applied} \\ x, & \text{otherwise} \end{cases}.$$

For ease of discussion let $\kappa = d/2$ for Lemma 5.4.1. Furthermore, let $\kappa = d$, $\alpha = 2/3$, and $\delta = 1/2$ for Lemmas 5.4.2 and 5.4.3. Moreover, let $\lambda = 1/\hat{c}'(d) - 1$ and let

$\varepsilon = 5 \cdot e^{-d}/\ln 2$, i. e., we have $1/\hat{c}'(d) \cdot (1+\lambda) < 1+\varepsilon$ and $d = O\big(\log(1/\varepsilon)\big)$. Then, for arbitrary but fixed $d \geqslant 8$, we obtain the following:

(i) By summing up all error probabilities, including the additional factor of $n^{1-\delta}$ for h^{map}, we get an overall error of $O(n^{1-(d+2)/9})$.

(ii) The space needs are $1/\hat{c}'(d) \cdot (1+\lambda) \cdot n$ elements from R for the vectors $\mathbf{t}_i$, $i \in [n^{1-\delta}]$, plus the space for h^{coll}, h^{split}, and h^{map}, which sums up to at most $(1+\varepsilon) \cdot n \cdot \log|R| + o(n) + O(\log\log|U|)$.

(iii) The evaluation time is at most $O(d^2)$, which is in $O\big(\log^2(1/\varepsilon)\big)$.

(iv) The construction time is dominated by the construction of the tables, which needs time $O(n)$, if $|R| \leqslant |U|$, as well as by the domain reduction which can also be done in time $O(n)$, if $\log|U|$ has size $O(2^{n^{1/4}}/n^{d/2})$.

To complete the proof of the Theorem 5.1, it remains to show Lemmas 5.4.1 to 5.4.5, which is done in the following five sections.

5.4.1. Collapsing the Universe

In this section we prove Lemma 5.4.1. Consider a universe U of size super-polynomial in $|S|$. A standard way to avoid problems with large keys is to apply an old technique [Sie95, Section 2], sometimes referred as *collapsing the universe* [DW03, Section 2.1] or *domain reduction* [PP08, Section 2], that comes at the cost of extra but negligible components in space usage and error probability.

The idea is to use a hash function that maps U to some smaller domain U', and then work with new keys $x' = h(x) \in U'$ of smaller description length. We require that $h\colon U \to U'$ is injective on S, but since in general S is an arbitrary unknown set, the usual way is to randomly choose h from some hash class $\mathcal{H}^{\text{coll}}$ and bound the probability that keys of S collide under h.

Let $|U'| = n^\kappa$, for constant $\kappa \geqslant 3$. If we define $\mathcal{H}^{\text{coll}}$ to be the $(\varepsilon, 2)$-universal hash class described in Section 5.3.2, then for any set $S \subseteq U$ of n elements, and constant ε, the probability that a randomly chosen h from $\mathcal{H}^{\text{coll}}$ is not one-to-one when restricted to S is

$$\Pr_{h \in \mathcal{H}^{\text{coll}}}\left(\exists\{x_0, x_1\} \in \binom{S}{2}\colon h(x_0) = h(x_1)\right) \leqslant \binom{n}{2} \cdot \frac{1+\varepsilon}{n^\kappa} = O(n^{2-\kappa})\,.$$

The function h can be evaluated in constant time and the additional space usage is $O(\log n + \log\log|U|)$, which is in $O(\log n)$ for all universes of reasonable size. Moreover, if $n^\kappa \cdot \log|U|$ is in $O(2^{n^{1/4}})$, then h can be constructed in time $O(n)$, at the price of an additive term of $2^{-n^{1/4}}$ to the probability that h is not one-to-one on S. This finishes the proof of the lemma. ■

5.4.2. Splitting the Key Set Evenly

In this section we prove Lemma 5.4.2. Let $\mathcal{H}^{\text{split}} = \mathcal{R}$, the hash class described in [DM90]. We will use $\mathcal{R}$ to *sharply split* a given key set $S, |S| = n$, into $n^{1-\delta}$ subsets $S_i, i \in [n^{1-\delta}]$, of size close to n^{δ}. A hash function h from $\mathcal{R}$ is built as follows:

$$h(x) = (g(x) + v_{f(x)}) \bmod n^{1-\delta} \text{ for all } x \in U,$$

where g and f are chosen uniformly at random from $(\kappa+1)$-wise independent classes with ranges $[n^{1-\delta}]$ and $[n^{\alpha}]$, where $\alpha \in (\delta, 1)$; and $\boldsymbol{v} = (v_j)_{j\in[n^{\alpha}]}$ is a vector of size n^{α} with fully random elements from $[n^{1-\delta}]$. For each h from $\mathcal{R}$ let $S_i = \{x \in S \mid h(x) = i\}$.

> Lemma 5.4.6 (Property of Hash Class $\mathcal{R}$ [DM90, part of Theorem 4.6]). Let $\lambda > 0$ be an arbitrary constant. If h is randomly chosen from $\mathcal{R}$, then it holds:
>
> $$\Pr(\exists i : S_i \geqslant (1+\lambda)\cdot n^{\delta}) = \left(\frac{e}{\kappa+1}\right)^{\kappa+1} \cdot n^{1-(\alpha-\delta)\cdot\kappa} + n^{1-\delta} \cdot \left(\frac{e^{\lambda}}{(1+\lambda)^{1+\lambda}}\right)^{n^{\delta}/\kappa}.$$

Proof. For all $i \in [n^{1-\delta}]$ and $j \in [n^{\alpha}]$ let

$$T_{i,j} = \{x \in S \mid g(x) = i \wedge f(x) = j\}.$$

Consider the events

$$\mathcal{T}_{i,j} := \{\text{The set } T_{i,j} \text{ has size larger than } \kappa.\},$$
$$\mathcal{T} := \{\text{It exists a set } T_{i,j} \text{ of size larger than } \kappa.\}.$$

Since $f(x)$ and $g(x)$ are $(\kappa+1)$-wise independent, the function $h'(x) = (g(x), f(x))$ is $(\kappa+1)$-wise independent too. It follows that

$$\Pr(\mathcal{T}_{i,j}) \leqslant \sum_{T\in\binom{S}{\kappa+1}} \Pr\left(\forall x \in T : h'(x) = (i,j)\right) = \binom{n}{\kappa+1} \cdot \left(\frac{1}{n^{\alpha+1-\delta}}\right)^{\kappa+1}.$$

Using the fact $\binom{n}{\kappa+1} \leqslant \left(\frac{e\cdot n}{\kappa+1}\right)^{\kappa+1}$, we get

$$\Pr(\mathcal{T}) \leqslant n^{\alpha} \cdot n^{1-\delta} \cdot \left(\frac{e\cdot n}{\kappa+1}\right)^{\kappa+1} \cdot \left(\frac{1}{n^{\alpha+1-\delta}}\right)^{\kappa+1} = \left(\frac{e}{\kappa+1}\right)^{\kappa+1} \cdot n^{1-(\alpha-\delta)\cdot\kappa}.$$

Hence, if $(\alpha-\delta)\cdot\kappa > 1$, then $\bar{\mathcal{T}}$ holds with high probability, i. e., h' is κ-*perfect* for S, which means that for each element $y \in [n^{1-\delta}] \times [n^{\alpha}]$ there are at most κ pairwise different elements x from S with $h'(x) = y$. Now, for all $i \in [n^{1-\delta}]$ and $j \in [n^{\alpha}]$ let $X_{i,j}$ be a random variable, where

$$X_{i,j} = \left|\{x \in S \mid f(x) = j \ \wedge \ g(x) + v_j \equiv i \bmod n^{1-\delta}\}\right|.$$

Using these random variables, the size of a set S_i is simply denoted as

$$|S_i| = \sum_{j \in [n^\alpha]} X_{i,j} \,,$$

and it follows that

$$\begin{aligned} \mathrm{Exp}(X_{i,j} \mid \bar{\mathcal{T}}) &= \sum_{a \in [n^{1-\delta}]} \left|\{x \in S \mid f(x) = j \ \wedge \ g(x) + a \equiv i \bmod n^{1-\delta}\}\right| \cdot \Pr(v_j = a \mid \bar{\mathcal{T}}) \\ &= \frac{|x \in S \mid f(x) = j|}{n^{1-\delta}} \,, \end{aligned}$$

as well as

$$\mathrm{Exp}(S_i \mid \bar{\mathcal{T}}) = \sum_{j \in [n^\alpha]} \frac{|x \in S \mid f(x) = j|}{n^{1-\delta}} = \frac{|S|}{n^{1-\delta}} = n^\delta \,.$$

Observe that all random variables $X_{i,j} \mid \bar{\mathcal{T}}$ are restricted to the range $\{0, 1, 2, \ldots, \kappa\}$ and therefore all random variables $Y_{i,j} \coloneqq (X_{i,j}/\kappa) \mid \bar{\mathcal{T}}$ are restricted to the range $[0, 1]$. Moreover the variables $Y_{i,0}, Y_{i,1}, \ldots, Y_{i,n^\alpha - 1}$ are independent. Therefore we can apply a standard a Chernoff–Hoeffding bound [Hoe63, by Theorem 1 with $\lambda \coloneqq t/\mu$] to limit the probability that a fixed S_i has size at least $(1+\lambda) \cdot n^\delta$, via

$$\Pr\left(|S_i| \geqslant (1+\lambda) \cdot n^\delta \mid \bar{\mathcal{T}}\right) = \Pr\left(\sum_{j \in [n^\alpha]} Y_{i,j} \geqslant (1+\lambda) \cdot \frac{n^\delta}{\kappa}\right) \leqslant \left(\frac{e^\lambda}{(1+\lambda)^{1+\lambda}}\right)^{n^\delta/\kappa} .$$

Finally, by the law of total probability, we get

$$\begin{aligned} &\Pr\left(\exists i \in [n^{1-\delta}] \colon |S_i| \geqslant (1+\lambda) \cdot n^\delta\right) \\ &\qquad \leqslant 1 \cdot \Pr(\mathcal{T}) + \Pr\left(\exists i \in [n^{1-\delta}] \colon |S_i| \geqslant (1+\lambda) \cdot n^\delta \mid \bar{\mathcal{T}}\right) \cdot 1 \\ &\qquad \leqslant \Pr(\mathcal{T}) + n^{1-\delta} \cdot \left(\frac{e^\lambda}{(1+\lambda)^{1+\lambda}}\right)^{n^\delta/\kappa} , \end{aligned}$$

which proves the lemma. ■

The functions f and g can be realized by polynomials of degree κ, see, e. g., Section 5.3.2. In this case the description size for h from $\mathcal{R} = \mathcal{H}^{\mathrm{split}}$ is $O\big((n^\alpha + \kappa) \cdot \log n\big)$ and the evaluation time is $O(\kappa)$. The time for the construction of the table is $O(n^\alpha)$ and the two polynomials can be build in time $O(\kappa)$. This finishes the proof of Lemma 5.4.2.■

5.4.3. Fully Random Mapping on a Small Key Set

The following proof of Lemma 5.4.3 is folklore. It is based on a standard two-level hashing approach, where one hash function splits the key set into not too large buckets and then for every bucket a separate hash function with appropriate random properties is used to map its keys to a common range.

REMARK. This idea lies at the core of the high performance hash class described in [DM90, DM92].

Let $|S| = 2 \cdot n^\delta$. Randomly choose a hash function h_s from a $(\kappa+1)$-wise independent hash class of functions mapping from U to $[n^\alpha]$. Define subsets

$$T_j := \{x \in S \mid j = h_s(x)\} \text{ for all } j \in [n^\alpha],$$

and consider the event

$$\mathcal{T} := \{\text{It exists a set } T_j \text{ of size larger than } \kappa.\}.$$

Analogously to the first part of the proof of Lemma 5.4.6, it follows that

$$\begin{aligned} \Pr(\mathcal{T}) &\leqslant n^\alpha \cdot \sum_{T \in \binom{S}{\kappa+1}} \Pr\left(\forall x \in T\colon h_s(x) = j\right) \\ &= n^\alpha \cdot \binom{2 \cdot n^\delta}{\kappa+1} \cdot \left(\frac{1}{n^\alpha}\right)^{\kappa+1} \leqslant \left(\frac{2 \cdot e}{\kappa+1}\right)^{\kappa+1} \cdot n^{\delta-(\alpha-\delta)\cdot\kappa}, \end{aligned}$$

that is h_s is κ-perfect for S with high probability for suitable parameters α, δ and κ. For each T_j, $j \in [n^\alpha]$, we randomly choose d hash functions $\tilde{h}_{0,j}, \tilde{h}_{1,j}, \ldots, \tilde{h}_{d-1,j}$ from different κ-wise independent hash classes, where $\tilde{h}_{i,j}$ has range $[m-i]$, for all $i \in [d]$. Using a standard approach, see Section 5.3.3.2, for each j one can define a mapping

$$h_j\colon U \to [m]^d,$$

on the basis of $\tilde{h}_{0,j}, \tilde{h}_{1,j}, \ldots, \tilde{h}_{d-1,j}$, such that, under the assumption of event $\overline{\mathcal{T}}$, for all $x \in T_j$ we have that A_x, the set of all elements of the vector $h_j(x)$, is a fully random element from $\binom{[m]}{d}$. Now the final hash function $h\colon U \to \binom{[m]}{d}$ is defined as

$$h(x) := A_x, \text{ where } A_x \text{ is the set of elements of } h_{h_s(x)}(x).$$

The space usage for the $1 + d \cdot n^\alpha$ hash functions h_s and $\tilde{h}_{i,j}, i \in [d], j \in [n^\alpha]$, is in $O\big(d \cdot (\kappa \cdot n^\alpha \cdot \log n + \log m)\big)$ if they are realized with polynomials, see Section 5.3.2. Additional space for $h_j, j \in [n^\alpha]$, is only needed during the evaluation and can be made as small as $O\big(d \cdot (\log|U| + \log m)\big)$. The evaluation time for h consists of the evaluation time of h_s and $\tilde{h}_{i,j}, i \in [d]$, as well as the additional time needed for h_j, which sums up to $O\big(d \cdot (\kappa + \log d)\big)$. Since the construction of h is done simply by choosing the coefficients of the polynomials, the construction time of h is $O(d \cdot \kappa \cdot n^\alpha)$. This ends the proof of the lemma. ■

5.4.4. Lower Bounds on the Number of Vectors for Linear Dependence

In this section we prove Lemma 5.4.4, which is a combination and strengthening of Theorems 1.1 and 1.2 of [Cal97]. In [Cal97] Calkin showed that if one randomly chooses

n binary weight-d vectors of length $m = n/c$, for constant $d \geqslant 5$, and c is below some threshold depending on d, then the vectors are linearly dependent with probability $O(n^{-1})$. Following Calkin's proof, we improve this result and obtain a polynomial error bound of $O\left(n^{1-(d+2)/9}\right)$, for $d \geqslant 8$.

! When we refer to [Cal97], be aware that we use n and m with reversed meaning.

5.4.4.1. Markov Chain and Probability Bound

We start by considering a sequence of random binary weight-d vectors $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \ldots$; we want to bound the probability that these vectors sum to $\mathbf{0}$, the zero vector. To this end, we define a discrete time Markov chain with $m+1$ states $\{0, 1, 2, \ldots, m\}$. At time $t = 0$ we are in state $X_0 = 0$, at time $t = 1$ we are in state $X_1 = \|\mathbf{b}_1\|_1 = d$, that is the state corresponding to the weight of $\mathbf{b}_1$; at time $t = 2$ we are in state $X_2 = \|\mathbf{b}_1 \oplus \mathbf{b}_2\|_1$, and so on ($\oplus$ denotes bitwise xor). In general we have

$$X_t = \left\| \bigoplus_{i=0}^{t} \mathbf{b}_i \right\|_1 \text{ for all } t \geqslant 0, \text{ where } \mathbf{b}_0 = \mathbf{0}.$$

Calkin showed [Cal97, page 267], that the probability that the process is in state 0 at time t, for $t \geqslant 0$, is exactly

$$\Pr(X_t = 0) = \sum_{i=0}^{m} \frac{1}{2^m} \cdot \lambda_i^t \cdot \binom{m}{i},$$

where the λ_i's are the eigenvalues of the transition matrix of the Markov chain. According to [Cal97, Theorem 2.1] these eigenvalues are given by

$$\lambda_i = \sum_{l=0}^{d} (-1)^l \cdot \frac{\binom{i}{l} \cdot \binom{m-i}{d-l}}{\binom{m}{d}}. \tag{5.1}$$

Note that it holds

$$\Pr(X_0 = 0) = \frac{1}{2^m} \cdot \sum_{i=0}^{m} \binom{m}{i} = \frac{1}{2^m} \cdot 2^m = 1,$$

and

$$\begin{aligned}
\Pr(X_1 = 0) &= \frac{1}{2^m} \cdot \sum_{i=0}^{m} \sum_{l=0}^{d} (-1)^l \cdot \frac{\binom{i}{l} \cdot \binom{m-i}{d-l} \cdot \binom{m}{i}}{\binom{m}{d}} \\
&= \frac{1}{2^m} \cdot \sum_{l=0}^{d} (-1)^l \cdot \binom{d}{l} \cdot \sum_{i=0}^{m} \binom{m-d}{i-l} \\
&= \frac{2^{m-d}}{2^m} \cdot \sum_{l=0}^{d} (-1)^l \cdot \binom{d}{l} = 0,
\end{aligned}$$

which naturally follows from the definition of the Markov process.

Now we come back to the original problem. Consider a multiset of n random binary weight-d vectors $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$. Let Y be a random variable counting the number of different (non-empty) sub-multisets whose vectors sum to $\mathbf{0}$. The probability that the n vectors are linearly dependent can be expressed as $\Pr(Y \geqslant 1)$. By applying Markov's inequality, we get the following upper bound

$$\Pr(Y \geqslant 1) \leqslant \operatorname{Exp}(Y) = \sum_{t=1}^{n} \binom{n}{t} \cdot \Pr(X_t = 0) = \sum_{t=1}^{n} \binom{n}{t} \cdot \sum_{i=0}^{m} \frac{1}{2^m} \cdot \lambda_i^t \cdot \binom{m}{i} . \quad (5.2)$$

Utilizing that $\Pr(X_0 = 0) = 1$, it follows that

$$\operatorname{Exp}(Y) = \sum_{t=0}^{n} \sum_{i=0}^{m} \frac{1}{2^m} \cdot \binom{m}{i} \cdot \binom{n}{t} \cdot \lambda_i^t - 1 = \sum_{i=0}^{m} \frac{1}{2^m} \cdot \binom{m}{i} \cdot (1 + \lambda_i)^n - 1 . \quad (5.3)$$

To bound $\Pr(Y \geqslant 1)$ from above we need asymptotics for the value of λ_i.

5.4.4.2. Asymptotics of λ_i — Revisited

In [Cal97, Section 3] Calkin determines asymptotics for the value of λ_i that relies on the expansion of the binomial coefficients. To get a more accurate value for λ_i, we have to extend this analysis.

Binomial Coefficient We start by expressing the binomial coefficient $\binom{m}{d}$ using the falling factorial $m^{\underline{d}} = m \cdot (m-1) \cdot (m-2) \cdot \ldots \cdot (m-(d-1))$ via

$$\binom{m}{d} = \frac{m^{\underline{d}}}{d!} .$$

The product $m^{\underline{d}} = \prod_{l=0}^{d-1} (m-l)$ is the factored form of a polynomial in m, that is

$$m^{\underline{d}} = s_d \cdot m^d + s_{d-1} \cdot m^{d-1} + \ldots + s_1 \cdot m^1 + s_0 \cdot m^0 ,$$

for integer coefficients $s_l = s(d,l)$, $0 \leqslant l \leqslant d$. These coefficients are the signed Stirling numbers of the first kind, see, e. g., [Knu97, page 67], sometimes denoted as

$$s(d,l) = (-1)^{d-l} \cdot \begin{bmatrix} d \\ l \end{bmatrix} ,$$

where $\left[\begin{smallmatrix} d \\ l \end{smallmatrix}\right]$ are the unsigned Stirling numbers of the first kind. Note that

$$\begin{aligned} s(d,d) &= \left[\begin{smallmatrix} d \\ d \end{smallmatrix}\right] = 1 , \\ s(d,0) &= \left[\begin{smallmatrix} d \\ 0 \end{smallmatrix}\right] = 0 , \text{ if } d > 0, \text{ and} \\ s(d+1,l) &= -d \cdot s(d,l) + s(d,l-1) , \text{ if } 1 \leqslant l \leqslant d . \end{aligned} \quad (5.4)$$

For example let $d = 4$, then we have

$$m^{\underline{4}} = m \cdot (m-1) \cdot (m-2) \cdot (m-3) = 1 \cdot m^4 - 6 \cdot m^3 + 11 \cdot m^2 - 6 \cdot m\,.$$

It follows that the binomial coefficient can be written as

$$\binom{m}{d} = \frac{\sum_{l=0}^{d} s(d,l) \cdot m^l}{d!} = \frac{\sum_{l=0}^{d} s(d,d-l) \cdot m^{d-l}}{d!} = \frac{m^d}{d!} \cdot \left(\sum_{l=0}^{d} \frac{s(d,d-l)}{m^l} \right). \tag{5.5}$$

Using (5.4) one can show via induction that for fixed l the signed Stirling number $s(d,d-l)$ can be seen as a polynomial in d of degree at most $2 \cdot l$. The following lemma characterizes this polynomial more precisely.

LEMMA 5.4.7 (FOLLOWS FROM [JOR79, PAGES 150–151][9]). For fixed l the signed Stirling number of the first kind $s(d,d-l)$ can be expressed as

$$s(d,d-l) = (-1)^l \cdot \left(0^l + \sum_{x=l+1}^{2 \cdot l} C(l,x) \cdot \binom{d}{x} \right),$$

with integer coefficients $C(l,x)$ recursively defined via

$$\begin{aligned} C(l,x) &= 0\ , \text{ if } x \leqslant l \text{ or } x \geqslant 2 \cdot l + 1\ , \\ C(1,2) &= 1\ , \text{ and} \\ C(l+1,x) &= (x-1) \cdot [C(l,x-1) + C(l,x-2)]\ . \end{aligned}$$

By applying Lemma 5.4.7 and the fact that $s(d,d) = 1$, we get

$$\binom{m}{d} = \frac{m^d}{d!} \cdot \left(1 + \sum_{l=1}^{d} \frac{(-1)^l \cdot \sum_{x=l+1}^{2l} C(l,x) \cdot \binom{d}{x}}{m^l} \right). \tag{5.6}$$

For example, it holds

$$\begin{aligned} C(1,2) &= 1 & C(2,3) &= 2 \cdot [C(1,2) + C(1,1)] = 2 \\ C(2,4) &= 3 \cdot [C(1,3) + C(1,2)] = 3 & C(3,4) &= 3 \cdot [C(2,3) + C(2,2)] = 6\,. \end{aligned}$$

Hence, $\binom{m}{4}$ can be expressed as

$$\begin{aligned} \binom{m}{4} &= \frac{m^4}{4!} \cdot \left(1 - \frac{C(1,2) \cdot \binom{4}{2}}{m} + \frac{C(2,3) \cdot \binom{4}{3} + C(2,4) \cdot \binom{4}{4}}{m^2} - \frac{C(3,4) \cdot \binom{4}{4}}{m^3} \right) \\ &= \frac{m^4}{4!} \cdot \left(1 - \frac{6}{m} + \frac{11}{m^2} - \frac{6}{m^3} \right) = \frac{m^{\underline{4}}}{4!}\,. \end{aligned}$$

[9] In [Jor79] the author writes S_n^{n-m} for our $s(n,n-m)$ and his coefficient $C_{m,\nu}$ is equivalent to our $(-1)^m \cdot C(m,2m-\nu)$.

Products of Binomial Coefficients With (5.5) the eigenvalue λ_i (5.1) can be written as

$$\lambda_i = \sum_{l=0}^{d} (-1)^l \cdot F_0 \cdot F_1 \cdot F_2 \,, \tag{5.7}$$

where

$$\begin{aligned} F_0 = \frac{i^l \cdot (m-i)^{d-l} \cdot d!}{l! \cdot (d-l)! \cdot m^d} &= \binom{d}{l} \cdot \frac{i^l}{m^l} \cdot \frac{(m-i)^{d-l}}{m^{d-l}} \\ &= \binom{d}{l} \cdot \left(\frac{i}{m}\right)^l \cdot \left(1 - \frac{i}{m}\right)^{d-l} \,, \end{aligned} \tag{5.8}$$

and

$$F_1 = \frac{m^d}{m^{\underline{d}}} \leqslant \left(\frac{m}{m-(d-1)}\right)^{d-1} \leqslant e^{(d-1)^2/(m-d+1)} \,, \tag{5.9}$$

as well as

$$F_2 = \left(\sum_{x=0}^{l} \frac{s(l, l-x)}{i^x}\right) \cdot \left(\sum_{y=0}^{d-l} \frac{s(d-l, d-l-y)}{(m-i)^y}\right) .$$

Using that $s(d,d) = 1$ and $s(d,l) = 0$ for $l > d$ and $l < 0$, the convolution of the two series of F_2 leads to

$$\begin{aligned} F_2 &= \left(\sum_{x=0}^{l} \frac{s(l, l-x)}{i^x}\right) \cdot \left(\sum_{y=0}^{d-l} \frac{s(d-l, d-l-y)}{(m-i)^y}\right) \\ &= 1 + \sum_{y=1}^{d} \sum_{x=0}^{y} \frac{s(l, l-x) \cdot s(d-l, (d-l)-(y-x))}{i^x \cdot (m-i)^{y-x}} \,. \end{aligned}$$

Applying Lemma 5.4.7 gives

$$\begin{aligned} F_2 = 1 + \sum_{y=1}^{d} \sum_{x=0}^{y} & \frac{(-1)^y \cdot \left(0^x + \sum_{a=x+1}^{2\cdot x} C(x,a) \cdot \binom{l}{a}\right)}{i^x \cdot (m-i)^{y-x}} \\ & \cdot \frac{\left(0^{y-x} + \sum_{b=y-x+1}^{2\cdot(y-x)} C(y-x,b) \cdot \binom{d-l}{b}\right)}{1} \,. \end{aligned}$$

Considering the inner sum for $x = y, x = 0$, and $1 \leqslant x \leqslant y - 1$, the last equation can be rearranged to

$$\begin{aligned} F_2 = 1 &+ \sum_{y=1}^{d} \sum_{a=y+1}^{2\cdot y} (-1)^y \cdot \frac{1}{i^y} \cdot C(y,a) \cdot \binom{l}{a} \\ &+ \sum_{y=1}^{d} \sum_{b=y+1}^{2\cdot y} (-1)^y \cdot \frac{1}{(m-i)^y} \cdot C(y,b) \cdot \binom{d-l}{b} \\ &+ \sum_{y=2}^{d} \sum_{x=1}^{y-1} \sum_{a=x+1}^{2\cdot x} \sum_{b=y-x+1}^{2\cdot(y-x)} (-1)^y \cdot \frac{1}{i^x \cdot (m-i)^{y-x}} \\ &\qquad \cdot C(x,a) \cdot C(y-x,b) \cdot \binom{l}{a} \cdot \binom{d-l}{b} . \end{aligned} \tag{5.10}$$

Recall that according to the definition of the coefficients C it holds

$$\begin{aligned} C(x,a) &= 0 \text{ , if } a \leqslant x \text{ or } a \geqslant 2\cdot x + 1 \\ C(y-x,b) &= 0 \text{ , if } b \leqslant y - x \text{ or } b \geqslant 2\cdot(y-x) + 1 . \end{aligned}$$

Binomial Identities The last ingredients are some identities of sums that involve products of binomial coefficients. Observe that

$$\begin{aligned} \binom{d}{l} \cdot \binom{l}{a} &= \binom{d-a}{l-a} \cdot \binom{d}{a} \\ \binom{d}{l} \cdot \binom{d-l}{b} &= \binom{d}{b} \cdot \binom{d-b}{l} \\ \binom{d}{l} \cdot \binom{l}{a} \cdot \binom{d-l}{b} &= \binom{d}{a} \cdot \binom{d-a}{b} \cdot \binom{d-b-a}{l-a} . \end{aligned}$$

Therefore, we get

$$\begin{aligned} \sum_{l=0}^{d} \binom{d}{l} \cdot \binom{l}{a} \cdot (-1)^l \cdot p^l \cdot q^{d-l} &= \binom{d}{a} \cdot \sum_{l=a}^{d} \binom{d-a}{l-a} \cdot (-p)^l \cdot q^{d-l} \\ &= \binom{d}{a} \cdot \sum_{l=0}^{d-a} \binom{d-a}{l} \cdot (-p)^{l+a} \cdot q^{d-l-a} \\ &= \binom{d}{a} \cdot (-p)^a \cdot (q-p)^{d-a} , \end{aligned} \tag{5.11}$$

and analogous

$$\sum_{l=0}^{d} \binom{d}{l} \cdot \binom{d-l}{b} \cdot (-1)^l \cdot p^l \cdot q^{d-l} = \binom{d}{b} \cdot q^b \cdot (q-p)^{d-b} , \tag{5.12}$$

as well as

$$\sum_{l=0}^{d} \binom{d}{l} \cdot \binom{l}{a} \cdot \binom{d-l}{b} \cdot (-1)^l \cdot p^l \cdot q^{d-l} = \binom{d}{a} \cdot \binom{d-a}{b} \cdot (-p)^a \cdot q^b \cdot (q-p)^{d-b-a} . \tag{5.13}$$

The last binomial identity that we need is

$$\sum_{l=0}^{d} (-1)^l \cdot \binom{m}{l} \cdot \binom{m}{d-l} = \begin{cases} \binom{m}{d/2} \cdot (-1)^{d/2}, & \text{if } d \text{ is even} \\ 0, & \text{otherwise} \end{cases} , \tag{5.14}$$

which can be found, e. g., in [Gou72, page 25, Equation 3.32]. The identity can be proved, e. g., by coefficient comparison of the expansion of $(1-x)^m \cdot (1-x)^m$, since

$$\begin{aligned} (1-x)^m \cdot (1+x)^m &= \left(\sum_{l=0}^{m} \binom{m}{l} \cdot (-x)^l \right) \cdot \left(\sum_{r=0}^{m} \binom{m}{r} \cdot x^r \right) \\ &= \sum_{r=0}^{2 \cdot m} \left[\sum_{l=0}^{r} (-1)^l \cdot \binom{m}{l} \cdot \binom{m}{r-l} \right] \cdot x^r , \end{aligned}$$

and

$$(1-x)^m \cdot (1+x)^m = (1-x^2)^m = \sum_{r=0}^{m} \binom{m}{r} \cdot (-x^2)^r = \sum_{r=0}^{m} \left[\binom{m}{r} \cdot (-1)^r \right] \cdot x^{2 \cdot r} .$$

Putting It All Together Note that F_1 (5.9) is independent of l. Substitution of F_0 (5.8) in (5.7) gives

$$\lambda_i = F_1 \cdot \sum_{l=0}^{d} (-1)^l \cdot \binom{d}{l} \cdot \left(\frac{i}{m} \right)^l \cdot \left(1 - \frac{i}{m} \right)^{d-l} \cdot F_2 .$$

Let $p = i/m$ and $q = 1-p$. Applying the binomial identities (5.11), (5.12) and (5.13) to the summands of $\sum_{l=0}^{d} (-1)^l \cdot \binom{d}{l} \cdot p^l \cdot q^{d-l} \cdot F_2$ (5.10) leads to

$$\lambda_i = F_1 \cdot (S_0 + S_1 + S_2 + S_3) , \tag{5.15}$$

where

$$S_0 = (1 - 2 \cdot p)^d , \tag{5.16}$$

and

$$S_1 = \sum_{y=1}^{d} \sum_{a=y+1}^{2 \cdot y} (-1)^y \cdot \frac{C(y,a)}{i^y} \cdot \binom{d}{a} \cdot (-p)^a \cdot (1 - 2 \cdot p)^{d-a} , \tag{5.17}$$

and

$$S_2 = \sum_{y=1}^{d} \sum_{b=y+1}^{2\cdot y} (-1)^y \cdot \frac{C(y,b)}{(m-i)^y} \cdot \binom{d}{b} \cdot (1-p)^b \cdot (1-2\cdot p)^{d-b}, \tag{5.18}$$

as well as

$$S_3 = \sum_{y=2}^{d} \sum_{x=1}^{y-1} \sum_{a=x+1}^{2\cdot x} \sum_{b=y-x+1}^{2\cdot(y-x)} (-1)^y \cdot \frac{C(x,a)\cdot C(y-x,b)}{i^x \cdot (m-i)^{y-x}} \cdot \binom{d}{a} \cdot \binom{d-a}{b} \cdot (-p)^a \cdot (1-p)^b \cdot (1-2\cdot p)^{d-b-a}. \tag{5.19}$$

5.4.4.3. The Middle Range

As in [Cal97, page 268], we derive a bound for λ_i, when i is close to $m/2$, more precisely, $i = (m - m^\theta)/2$, for $\theta \in [0,1)$. We treat the summands S_0, S_1, S_2, and S_3 separately. Consider S_0 (5.16):

$$S_0 = \left(1 - 2\cdot\frac{i}{m}\right)^d = \left(\frac{1}{m^{1-\theta}}\right)^d.$$

Consider S_1 (5.17): Shifting the index a by y gives

$$S_1 = \sum_{y=1}^{d} \sum_{a=1}^{y} \binom{d}{a+y} \cdot C(y, a+y) \cdot \left(\frac{1}{m}\right)^y \cdot \left(-\frac{m-m^\theta}{2\cdot m}\right)^a \cdot \left(\frac{1}{m^{1-\theta}}\right)^{d-a-y},$$

which can be bounded from above via

$$S_1 \leqslant \sum_{y=1}^{d} \sum_{a=1}^{y} \binom{d}{a+y} \cdot C(y, a+y) \cdot \left(\frac{1}{m^{1-\theta}}\right)^{d-a}.$$

From the ranges of the indices we have $a \leqslant y$ and $2 \leqslant a+y \leqslant 2\cdot d$. But since $\binom{d}{a+y} = 0$ for $a+y > d$, it follows that only terms with $a \leqslant d/2$ contribute to S_1 and therefore

$$S_1 = O\left(m^{-(1-\theta)\cdot d/2}\right).$$

Consider S_2 (5.18): Shifting the index b by y gives

$$S_2 = \sum_{y=1}^{d} \sum_{b=1}^{y} \binom{d}{b+y} \cdot C(y, b+y) \cdot \left(-\frac{1}{m}\right)^y \cdot \left(\frac{m+m^\theta}{2\cdot m}\right)^b \cdot \left(\frac{1}{m^{1-\theta}}\right)^{d-b-y},$$

which can be bounded from above via

$$S_2 \leqslant \sum_{y=1}^{d} \sum_{b=1}^{y} \binom{d}{b+y} \cdot C(y, b+y) \cdot \left(\frac{1}{m^{1-\theta}}\right)^{d-b}.$$

From the ranges of the indices we have $b \leqslant y$ and $2 \leqslant b+y \leqslant 2 \cdot d$. But since $\binom{d}{b+y} = 0$ for $b + y > d$, it follows that only terms with $b \leqslant d/2$ contribute to S_2 and therefore

$$S_2 = O\left(m^{-(1-\theta)\cdot d/2}\right).$$

Consider S_3 (5.19): Shifting the index a by x and shifting the index b by $y - x$ gives

$$\begin{aligned} S_3 = \sum_{y=2}^{d} \sum_{x=1}^{y-1} \sum_{a=1}^{x} \sum_{b=1}^{y-x} \binom{d}{a+x} \cdot \binom{d-a-x}{b+y-x} \cdot C(x, a+x) \cdot C(y-x, b+y-x) \\ \cdot (-1)^{y+x} \cdot \left(\frac{1}{m}\right)^{y} \cdot \left(-\frac{m-m^{\theta}}{2 \cdot m}\right)^{a} \cdot \left(\frac{m+m^{\theta}}{2 \cdot m}\right)^{b} \cdot \left(\frac{1}{m^{1-\theta}}\right)^{d-b-a-y}, \end{aligned}$$

which can be bounded from above via

$$\begin{aligned} S_3 \leqslant \sum_{y=2}^{d} \sum_{x=1}^{y-1} \sum_{a=1}^{x} \sum_{b=1}^{y-x} \binom{d}{a+x} \cdot \binom{d-a-x}{b+y-x} \cdot C(x, a+x) \cdot C(y-x, b+y-x) \\ \cdot \left(\frac{1}{m^{1-\theta}}\right)^{d-(a+b)}. \end{aligned}$$

From the ranges of the indices and the factors $\binom{d}{a+x}$ and $\binom{d-a-x}{b+y-x}$ it follows that for terms that contribute to the sum we have $a + x \leqslant d$ and $a \leqslant x$, which implies $a \leqslant d/2$, as well as $b + y - x \leqslant d - a - x$ and $b \leqslant y - x \leqslant d - a - x - b \leqslant d - a - 1 - b$, which implies $b \leqslant \frac{d-a-1}{2}$. Hence, we can bound the sum $a + b$ according to

$$a + b \leqslant a + \frac{d-a-1}{2} = \frac{a}{2} + \frac{d-1}{2} \leqslant \frac{d}{4} + \frac{d}{2} - \frac{1}{2} = \frac{3 \cdot d}{4} - \frac{1}{2}.$$

Therefore, we get for the summand S_3

$$S_3 = O\left(m^{-(1-\theta)\cdot(d/4+1/2)}\right).$$

Applying the results for S_0, S_1, S_2 and S_3 to (5.15) and utilizing that $F_1 = 1+o(1)$ (5.9), as well as $d/2 > d/4 + 1/2$ for $d \geqslant 3$, we obtain

$$\lambda_{\frac{m-m^{\theta}}{2}} = O\left(m^{-(1-\theta)\cdot(d/4+1/2)}\right), \text{ for } d \geqslant 3 \text{ and } \theta \in [0, 1). \tag{5.20}$$

Since $m^{\theta} \neq 0$, it remains to treat the eigenvalue $\lambda_{\frac{m}{2}}$ for m even. For this purpose we consider the term $\lambda_{\frac{m}{2}} \cdot \binom{m}{d}$, where according to (5.1) we have

$$\lambda_{\frac{m}{2}} \cdot \binom{m}{d} = \sum_{l=0}^{d} (-1)^{l} \cdot \binom{m/2}{l} \cdot \binom{m/2}{d-l}.$$

By (5.14) and $\left(\frac{m}{d}\right)^{d} \leqslant \binom{m}{d} \leqslant \frac{m^{d}}{d!}$, it follows that

$$\lambda_{\frac{m}{2}} \leqslant \frac{\binom{m/2}{d/2}}{\binom{m}{d}} = O\left(m^{-d/2}\right).$$

Therefore, the upper bound of summand S_1 also applies to $\lambda_{\frac{m}{2}}$.

5.4.4.4. Asymptotics of Exp(Y) — Revisited

Following [Cal97, Section 4] we consider the sum

$$S = \sum_{i=0}^{m} \frac{1}{2^m} \cdot \binom{m}{i} \cdot (1+\lambda_i)^n \,, \tag{5.21}$$

which equals $\mathrm{Exp}(Y) + 1$ (5.3), where $\mathrm{Exp}(Y)$ is an upper bound for $\Pr(Y \geqslant 1)$, the value we are looking for. We proceed as in [Cal97, Lemma 4.1] and show that S has the following properties.

- ▷ The tails of S, i. e., the partial sums for $i \in [0, \varepsilon \cdot m]$ and $i \in [(1-\varepsilon) \cdot m, m]$, are exponentially small.
- ▷ The middle part, i. e., the partial sum of S for $i \in [(1-\varepsilon) \cdot m/2, (1+\varepsilon) \cdot m/2]$, is bounded by $1 + O\left(m^{1-(d+2)/9}\right)$.
- ▷ The rest of the sum, i. e., for $i \in [\varepsilon \cdot m, (1-\varepsilon) \cdot m/2]$ and $i \in [(1+\varepsilon) \cdot m/2, (1-\varepsilon) \cdot m]$, is exponentially small, if $n/m < \hat{c}'(d)$ for some constant $\hat{c}'(d)$.

Let $\hat{\lambda}_i$ be an upper bound of $\max\{\lambda_i, \lambda_{m-i}\}$. Note that $\lambda_i = (-1)^d \cdot \lambda_{m-i}$ for $i > m/2$. We substitute λ_i by $\hat{\lambda}_i$ and consider the sums

$$S_t = \sum_{i=0}^{\varepsilon \cdot m} \frac{1}{2^m} \cdot \binom{m}{i} \cdot (1+\hat{\lambda}_i)^n \qquad S_r = \sum_{i=\varepsilon \cdot m}^{(1-\varepsilon) \cdot \frac{m}{2}} \frac{1}{2^m} \cdot \binom{m}{i} \cdot (1+\hat{\lambda}_i)^n$$

$$S_{n_1} = \sum_{i=(1-\varepsilon) \cdot \frac{m}{2}}^{\frac{m}{2} - \frac{m^{5/9}}{2}} \frac{1}{2^m} \cdot \binom{m}{i} \cdot (1+\hat{\lambda}_i)^n \quad S_{n_2} = \sum_{i=\frac{m}{2} - \frac{m^{5/9}}{2}}^{\frac{m}{2}} \frac{1}{2^m} \cdot \binom{m}{i} \cdot (1+\hat{\lambda}_i)^n \,.$$

Then, because of the symmetry $\binom{m}{i} = \binom{m}{m-i}$, it follows that

$$S \leqslant 2 \cdot (S_t + S_r + S_{n_1} + S_{n_2}) \,. \tag{5.22}$$

The parameter ε is a function of $c = n/m$ and the value $5/9$ is arbitrarily chosen from the interval $(1/2, 1)$. We let $c < 1$, $\varepsilon \leqslant 1/8$, and $d \geqslant 8$.

The Tails — "The Good" We consider S_t. Since $|\lambda_i| < 1$, we let $\hat{\lambda}_i = 1$. Furthermore, we use that

$$\binom{m}{i} \leqslant \binom{m}{\varepsilon \cdot m} \leqslant 2^{H_2(\varepsilon) \cdot m} \,,$$

where $H_2(\varepsilon) = -\varepsilon \cdot \log(\varepsilon) - (1-\varepsilon) \cdot \log(1-\varepsilon)$ is the binary entropy of a Bernoulli random variable with success probability ε. That gives

$$S_t \leqslant \varepsilon \cdot m \cdot 2^{H_2(\varepsilon) \cdot m} \cdot \frac{2^n}{2^m} \leqslant \frac{m}{8} \cdot \left(2^{H_2(\varepsilon) - (1-c)}\right)^m \,,$$

which is exponentially small for some suitable ε that makes $H_2(\varepsilon) - (1-c)$ a negative constant. Since $1-c$ is a constant from $(0,1)$ there is always such an ε.

THE MIDDLE — "THE BAD" First consider the sum S_{n_1}. According to [Cal97, Lemma 3.1 c], we can set $\hat{\lambda}_i = \left(1 - \frac{2 \cdot i}{m}\right)^d + O(m^{-2})$. Let $\rho = \left(1 - \frac{2 \cdot i}{m}\right)$, i. e., $i = \frac{m}{2} \cdot (1 - \rho)$ for $m^{-4/9} \leqslant \rho \leqslant \varepsilon$. Hence we get

$$S_{n_1} \leqslant \sum_{\rho \cdot m = m^{5/9}}^{\rho \cdot m = \varepsilon \cdot m} \underbrace{\binom{m}{\frac{m}{2} \cdot (1 - \rho)} \cdot 2^{-m}}_{B} \cdot \left(1 + \rho^d + O(m^{-2})\right)^{c \cdot m} .$$

The factor B can be bounded via

$$B = \binom{m}{x} \cdot 2^{-x} \cdot 2^{-(m-x)} = \Pr\left(X = x\right) \leqslant \Pr\left(X \leqslant x\right) ,$$

for $X = \mathrm{Bin}[m, 1/2]$ and $x = m/2 \cdot (1 - \rho)$. Since $X = \sum_{j=1}^{m} X_j$ for independent Bernoulli random variables X_j with success probability $1/2$, we can apply the following standard Chernoff–Hoeffding bound [MU05, Corollary 4.10] for symmetric random variables

$$\Pr\left(X \leqslant (1 - \rho) \cdot \mathrm{Exp}(X)\right) \leqslant e^{-\rho^2 \cdot \mathrm{Exp}(X)} .$$

With $\mathrm{Exp}(X) = m/2$ and $x = (1 - \rho) \cdot \mathrm{Exp}(X)$, we get

$$B \leqslant e^{-\rho^2 \cdot m/2} .$$

Using that $1 + y \leqslant e^y$, we conclude

$$S_{n_1} \leqslant \sum_{\rho \cdot m = m^{5/9}}^{\rho \cdot m = \varepsilon \cdot m} \left(e^{-\rho^2/2 + \rho^d + O(m^{-2})}\right)^m .$$

We want to bound $-\rho^2/2 + \rho^d$. Under the condition that $d \geqslant 3$, we get

$$-\frac{\rho^2}{2} + \rho^d \leqslant -\frac{\rho^2}{4} ,$$

if $\rho \leqslant \varepsilon \leqslant 1/4$. It follows that for constant $\rho \leqslant \varepsilon$ the corresponding summands of S_{n_1} are exponentially small. For non-constant $\rho \geqslant m^{-4/9}$ the contribution of the corresponding summands is at most

$$e^{-\frac{m^{1/9}}{4} + O(m^{-1})} ,$$

which is exponentially small. Hence S_{n_1} is exponentially small.

Second consider the sum S_{n_2}. Using the results from Section 5.4.4.3, we get

$$\hat{\lambda}_i = O\left(m^{-4/9 \cdot (d/4 + 1/2)}\right) .$$

It follows that

$$S_{n_2} \leqslant \left(1 + O\left(m^{-(d+2)/9}\right)\right)^{c \cdot m} \cdot \frac{1}{2^m} \cdot \sum_{i=0}^{\frac{m}{2}} \binom{m}{i} .$$

Using that $1 + x \leqslant e^x$, gives

$$S_{n_2} \leqslant e^{O\left(m^{1-(d+2)/9}\right)}.$$

The exponent is bounded by $o(1)$ for $d \geqslant 8$, and since for $x \leqslant 1$ it holds

$$e^x = \sum_{j=0}^{\infty} \frac{x^j}{j!} \leqslant 1 + x + x^2 \cdot \sum_{j=1}^{\infty} \frac{1}{2} = 1 + x + x^2 \leqslant 1 + 2 \cdot x,$$

we get

$$S_{n_2} \leqslant 1 + O\left(m^{1-(d+2)/9}\right).$$

THE REST — "AND THE UGLY" It remains to consider the sum S_r. As before, according to [Cal97, Lemma 3.1 c] we have $\hat{\lambda}_i = \left(1 - \frac{2 \cdot i}{m}\right)^d + O(m^{-2})$. Let $i = \alpha \cdot m$, for $\alpha \in [\varepsilon, (1-\varepsilon)/2]$, then we obtain

$$\begin{aligned} S_r &= \sum_{\alpha \cdot m = \varepsilon \cdot m}^{\alpha \cdot m = (1-\varepsilon) \cdot \frac{m}{2}} \frac{1}{2^m} \cdot \binom{m}{\alpha \cdot m} \cdot \left(1 + \hat{\lambda}_{\alpha \cdot m}\right)^n \\ &\leqslant \sum_{\alpha \cdot m = \varepsilon \cdot m}^{\alpha \cdot m = (1-\varepsilon) \cdot \frac{m}{2}} e^{f(\alpha, c, d) \cdot m + O(m^{-1})}, \end{aligned}$$

where

$$f(\alpha, c, d) = -\ln(2) - \alpha \cdot \ln(\alpha) - (1-\alpha) \cdot \ln(1-\alpha) + c \cdot \ln\left(1 + (1 - 2 \cdot \alpha)^d\right).$$

We are looking for some threshold $\hat{c}' = \hat{c}'(d)$, such that for all $c < \hat{c}'$ there is a constant $\gamma < 0$ with $f(\alpha, c, d) < \gamma$, for all $\alpha \in [\varepsilon, (1-\varepsilon)/2]$. For this, we consider local maximum points of f at which f evaluates to 0. Hence we want to solve the homogeneous system

$$\left\{ f(\alpha, c) = 0, \frac{\partial f(\alpha, c)}{\partial \alpha} = 0 \right\}. \tag{5.23}$$

Let $\tilde{\alpha} = e^{-d}$, and let

$$\tilde{c}(\alpha) = \frac{\ln(2) + \alpha \cdot \ln(\alpha) + (1-\alpha) \cdot \ln(1-\alpha)}{\ln\left(1 + (1 - 2 \cdot \alpha)^d\right)}.$$

It holds that $f(\alpha, \tilde{c}(\alpha)) = 0$, and in particular $f(\tilde{\alpha}, \tilde{c}(\tilde{\alpha})) = 0$. Using a computer algebra system one easily gets the asymptotic expansion of $\tilde{c}$, see, e. g., [Map09, page 233], which is

$$\begin{aligned} \tilde{c}(\alpha) = 1 &+ \frac{\ln(\alpha) - 1 + d}{\ln(2)} \cdot \alpha \\ &+ \frac{\ln(2) - d^2 \cdot \ln(2) + 2 \cdot d \cdot \ln(2) + 2 \cdot d \cdot \ln(\alpha) - 2 \cdot d + 2 \cdot d^2}{2 \cdot \ln(2)^2} \cdot \alpha^2 + O(\alpha^3). \end{aligned}$$

It follows that

$$\tilde{c}(\tilde{\alpha}) = 1 - \frac{e^{-d}}{\ln(2)} - \frac{e^{-2\cdot d}}{2\cdot\ln(2)} \cdot \left(-1 + d^2 - 2\cdot d + \frac{2\cdot d}{\ln(2)}\right) + O(e^{-3\cdot d}) .$$

Furthermore, one observes that

$$\lim_{d\to\infty} \frac{\partial f(\tilde{\alpha}, \tilde{c})}{\partial \alpha} = 0 ,$$

where $(\tilde{\alpha}, \tilde{c}(\tilde{\alpha}))$ is near a local maximum point. According to [Cal97, page 270] there are no other relevant solutions of (5.23). Hence, if we choose $c < \hat{c}'(d) \leqslant \tilde{c}(\tilde{\alpha}, d)$, then the sum S_r is exponentially small in m. For this it is sufficient to define

$$\hat{c}'(d) = 1 - \frac{2\cdot e^{-d}}{\ln(2)} .$$

The Complete Sum Recall that $\Pr(Y \geqslant 1)$ (5.2) is the probability that the $n = c\cdot m$ random binary weight-d vectors of length m are linearly dependent and it holds $\Pr(Y \geqslant 1) \leqslant \mathrm{Exp}(Y)$ (5.3), as well as $\mathrm{Exp}(Y) = S - 1$ (5.21). According to (5.22) it follows that for $d \geqslant 8$ and $c < \hat{c}'(d)$, we have

$$\Pr(Y \geqslant 1) \leqslant O\left(m^{1-(d+2)/9}\right) .$$

This finishes the proof of Lemma 5.4.4. ■

5.4.5. Linear Independence Implies Stochastic Independence

In this section we prove Lemma 5.4.5. We assume that $(R, \oplus)$ is an abelian group. Hence, the structures $(R^n, \oplus)$ and $(R^m, \oplus)$ with componentwise $\oplus$ are abelian groups too. Since $\boldsymbol{M} \in \{0,1\}^{n\times m}$ has *full row rank*, the mapping $\phi\colon R^m \to R^n$ via $\phi(\boldsymbol{v}) = \boldsymbol{M}\cdot\boldsymbol{v}$ is a *surjective* group homomorphism. Each set of preimages

$$\phi^{-1}(\boldsymbol{w}) = \{\boldsymbol{v} \in R^m \mid \phi(\boldsymbol{v}) = \boldsymbol{w}\}$$

of an element $\boldsymbol{w}$ from R^n is a coset of the kernel $\ker(\phi)$ of ϕ. All cosets of the kernel have the same cardinality $|\ker(\phi)|$. Therefore, if $\boldsymbol{v} \in R^m$ is fully random, then the image $\boldsymbol{M}\cdot\boldsymbol{v}$ is uniformly distributed in R^n. This finishes the proof of the lemma. ■

5.5. Conclusion

We have shown how to construct a data structure for representing a *hash function that is with high probability fully random* for an arbitrary unknown key set without excessive costs in space and time consumption. In fact our construction is near optimal

and has the fastest evaluation time among all data structures currently known that simulate full randomness with comparable space needs. This justifies the assumption that the hash functions used for the graph, hypergraph, and matrix structures analyzed in the previous chapters are fully random and have constant evaluation time, see Section 2.2.2.

6

Final Remarks

In Chapters 3 and 4 we considered random bipartite graphs with bounded left degree, n left nodes, and m right nodes. We determined left degree distributions for maximizing the ratio $c = n/m$, called load factor, up to which such graphs a. a. s. have an empty 2-core, i. e., admit an order generating matching, or a. a. s. have a 2-core of density at most 1, i. e., admit a general left-perfect matching. In essence, we showed that in the former case irregular distributions can be superior to regular ones, and that in the latter case regularity is optimal. Based on matchings in such graphs with optimized left degrees for large loads, we discussed how to efficiently solve four basic data structure problems known as dictionary, membership, retrieval, and perfect hashing. Since our graphs rely on fully random hash functions, in Chapter 5 we gave a randomized algorithm that w. h. p. determines such functions in linear time with failure probability arbitrary close to 0.

Open Problems We close with a list of tasks that appear interesting, but remained untouched or unsolved in the course of our study.

- ▷ Extend the proof of [Lel12a] in order to obtain orientation thresholds $\hat{c}_{k,\ell}(\mathbf{d}, \boldsymbol{\alpha})$ for graphs with irregular left degree distributions, see Section 3.1.1.5.
- ▷ Prove Conjecture 3.1.1, which says that the generalized selfless algorithm works a. a. s. up to the orientation threshold $\hat{c}_{k,\ell}(d)$ for all suitable parameters (d, k, ℓ); generalize this to $\hat{c}_{k,\ell}(\mathbf{d}, \boldsymbol{\alpha})$.
- ▷ Show that the fixed degree distribution is optimal to the left of the 2-core density threshold $\hat{c}(\bar{d})$, while the binomial degree distribution is optimal to the right, as stated in Conjecture 3.1.2.
- ▷ Obtain provable performance bounds for cuckoo hashing with pages.
- ▷ Extend Theorem 4.5 in order to obtain distributions $\boldsymbol{\alpha}^*$ that maximize the ℓ_+-core appearance threshold $\check{c}_{\ell_+}(\mathbf{d}, \boldsymbol{\alpha})$ for all suitable parameters $\ell_+ = \ell + 1$ and $\mathbf{d}$.
- ▷ Prove Conjecture 4.1.1, which states that the 2-core appearance threshold can be made arbitrary close to 1; maybe this could be used to improve the perfect hash function construction from Section 4.6.

- ▷ Obtain core appearance and density thresholds, or even general orientation thresholds, for random bipartite graphs with left degree $d > 2$ that are based on simple, non-ideal hash functions, like, e. g., in [DW03, PT12, ADW13] concerning the 2-core density threshold in the 2-left-regular case.

A

Appearance of 2-Cores

A.1. Asymptotic 2-Core Probability for Normal Graphs of Type A

As we have discussed in Section 4.1.1.3, for $d = 2$ the following statements are equivalent:

- ▷ $\mathbf{M}^d_{n,m}$ has full row rank,
- ▷ $G^d_{n,m}$ has an order generating matching,
- ▷ $H^d_{m,n}$ has an empty 2-core.

Depending on the actual type of the random graphs and matrices, these statements are true if and only if:

- ▷ type B and type C: Graph $H^d_{m,n}$ is acyclic.
- ▷ type A: Graph $H^d_{m,n}$ has no cycle of length at least two, and there is no simple path that starts and ends at two nodes that have a cycle of length one.

Let $c = n/m$ be a positive constant smaller than $1/2$, and let $P(c, n)$ be the probability that $H^d_{m,n}$ has an empty 2-core. It is long known that

$$\lim_{n\to\infty} P(c, n) = \exp(c) \cdot \sqrt{1 - 2 \cdot c} \text{ for type B,}$$
$$\lim_{n\to\infty} P(c, n) = \sqrt{1 - 4 \cdot c^2} \text{ for type C,}$$

see, e. g., [CHM92, Section 3]. In this section, we will derive an analogous result for type A.

So, consider a random graph $H^d_{m,n}$ of type A and the following events:

$$\mathcal{A} \coloneqq \{H^d_{m,n} \text{ has no cycle of length at least two.}\},$$
$$\mathcal{B} \coloneqq \{H^d_{m,n} \text{ has no component with two cycles.}\}.$$

According to the characterization above, it holds that

$$\Pr(\overline{\mathcal{A}}) < 1 - P(c, n) < \Pr(\overline{\mathcal{A}}) + \Pr(\overline{\mathcal{B}}).$$

For our parameters, the asymptotic probability of $\overline{\mathcal{B}}$, i. e., the existence of a bicyclic component, also known as complex component, is zero, see, e. g., [DK12, Theorem 4]. Therefore, we have

$$\lim_{n\to\infty} \Pr(\mathcal{A}) = \lim_{n\to\infty} P(c, n)\,.$$

Let X be the number of cycles of length 1 in $H^d_{m,n}$, which makes X a $\mathrm{Bin}\left(n, c/n\right)$-distributed random variable. By the law of total probability we have

$$\Pr(\mathcal{A}) = \sum_{i=0}^{\log(n)} \Pr(\mathcal{A} \mid X = i) \cdot \Pr(X = i) + \sum_{i=\log(n)+1}^{n} \Pr(\mathcal{A} \mid X = i) \cdot \Pr(X = i)\,.$$

The condition $X = i$ reduces the number of available edges for cycles of length larger than one to $n - i$. However, edges that form cycles of length one cannot be a part of longer cycles, i. e., there is no structural dependency. This allows us to derive the following upper bound

$$\begin{aligned}\Pr(\mathcal{A}) &\leqslant \Pr(\mathcal{A} \mid X = 0) \cdot \Pr(X \leqslant \log(n)) \\ &\quad + \Pr(\mathcal{A} \mid X = \log(n) + 1) \cdot \Pr(X \geqslant \log(n) + 1) \\ &\leqslant \Pr(\mathcal{A} \mid X = 0) + \Pr(X > \log(n))\,,\end{aligned}$$

as well as the following lower bound

$$\Pr(\mathcal{A}) \geqslant \Pr(\mathcal{A} \mid X = \log(n)) \cdot \Pr(X \leqslant \log(n))\,.$$

By Markov's inequality, we have $\Pr(X > \log(n)) < \frac{c}{\log(n)}$. Therefore, we get

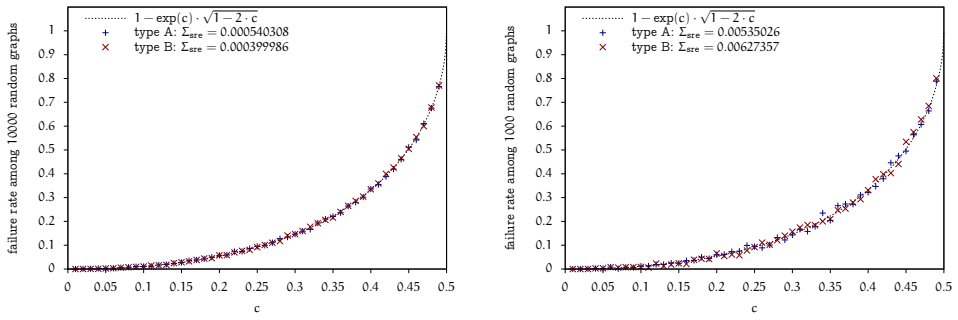
$$\lim_{n\to\infty} \Pr(X > \log(n)) = 0 \text{ and } \lim_{n\to\infty} \Pr(X \leqslant \log(n)) = 1\,.$$

Furthermore, it holds that the probability $\Pr(\mathcal{A} \mid X = 0)$ is equivalent to the probability that a random graph $H^d_{m,n}$ of type B is acyclic, see above. Moreover, the probabilities $\Pr(\mathcal{A} \mid X = 0)$ and $\Pr(\mathcal{A} \mid X = \log(n))$ are asymptotically equal. It follows that

$$\lim_{n\to\infty} P(c, n) = \lim_{n\to\infty} \Pr(\mathcal{A}) = \exp(c) \cdot \sqrt{1 - 2 \cdot c}\,,$$

which means that the asymptotic probability that $H^d_{m,n}$ has an empty 2-core is the same for type A and type B.

In order to back up this finding, we did an experiment for large pseudorandom graphs of type A and type B that were created using the Mersenne Twister, see Section 3.4.4. For fixed type from $\{A, B\}$ and fixed number of nodes m from $\{10^6, 10^7\}$, we generated $a = 10^{10}/m$ pseudorandom graphs $H^d_{m,n}$ with $n = c \cdot m$ edges for each $c = n/m \in \{0.01 + 0.01 \cdot i \mid i \in [49]\}$ and obtained the rate of graphs with non-empty 2-core (failure rate). The results are presented in Figure A.1.1. They show that for $m = 10^6$ and $m = 10^7$ the failure rates for type A and type B are both very close to the asymptotic failure probability $1 - \lim_{n\to\infty} P(c, n)$. The error Σ_{sre} for $m = 10^7$ is larger than for $m = 10^6$, which is possibly due to the smaller number of attempts a per load factor.



(a) $m = 10^6, a = 10^4$ (b) $m = 10^7, a = 10^3$

Figure A.1.1.: Rate of obtaining a non-empty 2-core in pseudorandom graphs of type A and type B for $c = n/m = 0.01, 0.02, \ldots, 0.49$. The value Σ_{sre} is the sum of squares of the residuals with respect to asymptotic failure probability (fit function) $1 - \exp(c) \cdot \sqrt{1 - 2 \cdot c} = 1 - \lim_{n\to\infty} P(c, n)$.

A.2. Maximum 2-Core Thresholds for Non-Uniform Hypergraphs of Type B

Figure A.2.2 shows optimal thresholds $\check{c}(\mathbf{d})$ for the appearance of a 2-core in hypergraphs $H^{\mathbf{d}}_{m,n,\boldsymbol{\alpha}}$ (type B) with $\mathbf{d} = (d_0, d_1)$, for edge sizes $3 \leqslant d_0 \leqslant 6$ and $d_0 \leqslant d_1 \leqslant 300$. The values were obtained with Algorithm 15 on input $\mathbf{d}$ and $\varepsilon = 10^{-10}$. In addition, for d_1 up to 50, Tables A.2.1 and A.2.2 list optimal points (z^*, α^*) of the transformed threshold function (4.1) and the corresponding average edge size $\bar{d} = \alpha^* \cdot d_0 + (1 - \alpha^*) \cdot d_1$, as well as the corresponding case of Theorem 4.5 that was applied.

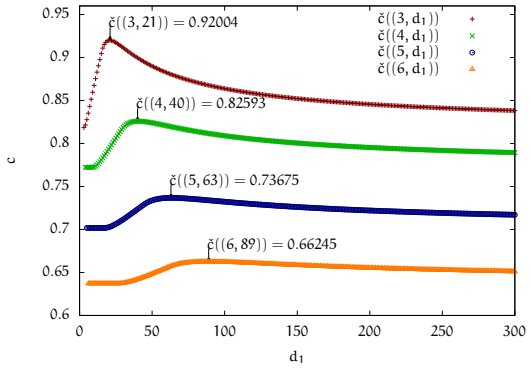


Figure A.2.2.: Optimal 2-core thresholds $\check{c}(\mathbf{d})$ for all $\mathbf{d} = (d_0, d_1) \in \mathbb{N}^2$ with $3 \leqslant d_0 \leqslant 6$ and $d_0 \leqslant d_1 \leqslant 300$.

d_1	z^*	α^*	$\bar{d}$	$\check{c}((3,d_1))$	case
3	0.71533	1.00000	3.00000	0.81847	1(i)
4	0.75000	0.83596	3.16404	0.82151	1(ii)
5	0.77460	0.84671	3.30658	0.82770	1(ii)
6	0.79370	0.85419	3.43744	0.83520	1(ii)
7	0.80911	0.86014	3.55944	0.84321	1(ii)
8	0.82188	0.86512	3.67439	0.85138	1(ii)
9	0.83268	0.86940	3.78359	0.85952	1(ii)
10	0.84198	0.87315	3.88795	0.86752	1(ii)
11	0.85009	0.87648	3.98818	0.87535	1(ii)
12	0.85724	0.87946	4.08482	0.88298	1(ii)
13	0.86361	0.88217	4.17830	0.89040	1(ii)
14	0.86932	0.88464	4.26898	0.89761	1(ii)
15	0.87449	0.88690	4.35715	0.90461	1(ii)
16	0.77926	0.88684	4.47102	0.91089	2(iii)
17	0.75795	0.88616	4.59372	0.91510	2(iii)
18	0.74529	0.88599	4.71015	0.91772	2(iii)
19	0.73694	0.88620	4.82077	0.91922	2(iii)
20	0.73116	0.88671	4.92601	0.91992	2(iii)
21	0.72704	0.88743	5.02626	0.92004	2(iii)
22	0.72404	0.88832	5.12190	0.91974	2(iii)
23	0.72185	0.88934	5.21328	0.91914	2(iii)
24	0.72022	0.89044	5.30070	0.91832	2(iii)
25	0.71900	0.89162	5.38446	0.91734	2(iii)
26	0.71809	0.89283	5.46482	0.91626	2(iii)
27	0.71741	0.89408	5.54202	0.91510	2(iii)
28	0.71689	0.89535	5.61628	0.91390	2(iii)
29	0.71651	0.89662	5.68780	0.91266	2(iii)
30	0.71621	0.89790	5.75675	0.91141	2(iii)
31	0.71599	0.89917	5.82330	0.91016	2(iii)
32	0.71583	0.90043	5.88761	0.90891	2(iii)
33	0.71570	0.90167	5.94981	0.90768	2(iii)
34	0.71561	0.90290	6.01003	0.90645	2(iii)
35	0.71554	0.90411	6.06839	0.90525	2(iii)
36	0.71549	0.90530	6.12498	0.90406	2(iii)
37	0.71545	0.90647	6.17992	0.90290	2(iii)
38	0.71542	0.90762	6.23328	0.90177	2(iii)
39	0.71540	0.90875	6.28516	0.90065	2(iii)
40	0.71538	0.90985	6.33562	0.89956	2(iii)
41	0.71537	0.91093	6.38475	0.89850	2(iii)
42	0.71536	0.91198	6.43260	0.89746	2(iii)
43	0.71535	0.91302	6.47925	0.89644	2(iii)
44	0.71535	0.91403	6.52474	0.89545	2(iii)
45	0.71534	0.91502	6.56913	0.89448	2(iii)
46	0.71534	0.91599	6.61246	0.89354	2(iii)
47	0.71534	0.91694	6.65480	0.89261	2(iii)
48	0.71534	0.91786	6.69617	0.89171	2(iii)
49	0.71534	0.91877	6.73662	0.89083	2(iii)
50	0.71533	0.91966	6.77619	0.88997	2(iii)

(a) Case $d_0 = 3$, the maximum threshold in this range is about 0.92004.

d_1	z^*	α^*	$\bar{d}$	$\check{c}((4,d_1))$	case
4	0.85100	1.00000	4.00000	0.77228	1(i)
5	0.85100	1.00000	4.00000	0.77228	1(i)
6	0.85100	1.00000	4.00000	0.77228	1(i)
7	0.85100	1.00000	4.00000	0.77228	1(i)
8	0.85100	1.00000	4.00000	0.77228	1(i)
9	0.85100	1.00000	4.00000	0.77228	1(i)
10	0.85837	0.98319	4.10087	0.77261	1(ii)
11	0.86544	0.97048	4.20664	0.77358	1(ii)
12	0.87169	0.96143	4.30855	0.77501	1(ii)
13	0.87725	0.95477	4.40707	0.77677	1(ii)
14	0.88225	0.94974	4.50259	0.77878	1(ii)
15	0.88678	0.94587	4.59540	0.78097	1(ii)
16	0.89090	0.94285	4.68579	0.78329	1(ii)
17	0.89467	0.94046	4.77397	0.78571	1(ii)
18	0.89814	0.93856	4.86013	0.78819	1(ii)
19	0.90134	0.93704	4.94444	0.79072	1(ii)
20	0.90430	0.93581	5.02703	0.79329	1(ii)
21	0.90706	0.93482	5.10804	0.79587	1(ii)
22	0.90964	0.93402	5.18757	0.79847	1(ii)
23	0.91205	0.93338	5.26572	0.80106	1(ii)
24	0.91431	0.93287	5.34257	0.80365	1(ii)
25	0.91643	0.93247	5.41822	0.80623	1(ii)
26	0.91844	0.93215	5.49272	0.80880	1(ii)
27	0.92033	0.93191	5.56615	0.81135	1(ii)
28	0.92212	0.93173	5.63855	0.81388	1(ii)
29	0.91076	0.93157	5.71069	0.81638	2(iii)
30	0.89400	0.93133	5.78542	0.81858	2(iii)
31	0.88530	0.93119	5.85792	0.82036	2(iii)
32	0.87918	0.93113	5.92826	0.82179	2(iii)
33	0.87453	0.93115	5.99653	0.82293	2(iii)
34	0.87084	0.93124	6.06282	0.82383	2(iii)
35	0.86785	0.93138	6.12721	0.82453	2(iii)
36	0.86539	0.93157	6.18979	0.82505	2(iii)
37	0.86333	0.93180	6.25063	0.82544	2(iii)
38	0.86161	0.93206	6.30980	0.82570	2(iii)
39	0.86015	0.93236	6.36738	0.82586	2(iii)
40	0.85891	0.93268	6.42343	0.82593	2(iii)
41	0.85784	0.93303	6.47802	0.82593	2(iii)
42	0.85693	0.93339	6.53122	0.82587	2(iii)
43	0.85615	0.93377	6.58308	0.82576	2(iii)
44	0.85548	0.93416	6.63365	0.82560	2(iii)
45	0.85489	0.93456	6.68299	0.82540	2(iii)
46	0.85439	0.93497	6.73115	0.82518	2(iii)
47	0.85395	0.93539	6.77818	0.82492	2(iii)
48	0.85357	0.93582	6.82413	0.82465	2(iii)
49	0.85324	0.93624	6.86903	0.82436	2(iii)
50	0.85295	0.93668	6.91294	0.82405	2(iii)

(b) Case $d_0 = 4$, the maximum threshold in this range is about 0.82593.

Table A.2.1.: Optimal 2-core thresholds $\check{c}(\mathbf{d})$ and corresponding optimal points (z^*, α^*) for all $\mathbf{d} = (d_0, d_1) \in \mathbb{N}^2$ with $3 \leqslant d_0 \leqslant 4$ and $d_0 \leqslant d_1 \leqslant 50$.

d_1	z^*	α^*	$\bar{d}$	$\check{c}((5,d_1))$	case
5	0.90335	1.00000	5.00000	0.70178	1(i)
6	0.90335	1.00000	5.00000	0.70178	1(i)
7	0.90335	1.00000	5.00000	0.70178	1(i)
8	0.90335	1.00000	5.00000	0.70178	1(i)
9	0.90335	1.00000	5.00000	0.70178	1(i)
10	0.90335	1.00000	5.00000	0.70178	1(i)
11	0.90335	1.00000	5.00000	0.70178	1(i)
12	0.90335	1.00000	5.00000	0.70178	1(i)
13	0.90335	1.00000	5.00000	0.70178	1(i)
14	0.90335	1.00000	5.00000	0.70178	1(i)
15	0.90335	1.00000	5.00000	0.70178	1(i)
16	0.90335	1.00000	5.00000	0.70178	1(i)
17	0.90335	1.00000	5.00000	0.70178	1(i)
18	0.90617	0.99375	5.08121	0.70187	1(ii)
19	0.90905	0.98793	5.16898	0.70215	1(ii)
20	0.91172	0.98300	5.25495	0.70258	1(ii)
21	0.91421	0.97880	5.33924	0.70315	1(ii)
22	0.91654	0.97518	5.42198	0.70383	1(ii)
23	0.91871	0.97204	5.50326	0.70460	1(ii)
24	0.92076	0.96931	5.58318	0.70545	1(ii)
25	0.92268	0.96691	5.66183	0.70637	1(ii)
26	0.92450	0.96480	5.73927	0.70734	1(ii)
27	0.92621	0.96293	5.81557	0.70836	1(ii)
28	0.92783	0.96127	5.89081	0.70942	1(ii)
29	0.92937	0.95979	5.96502	0.71051	1(ii)
30	0.93084	0.95847	6.03827	0.71163	1(ii)
31	0.93223	0.95728	6.11061	0.71278	1(ii)
32	0.93356	0.95622	6.18206	0.71394	1(ii)
33	0.93483	0.95526	6.25268	0.71512	1(ii)
34	0.93604	0.95440	6.32250	0.71631	1(ii)
35	0.93720	0.95361	6.39156	0.71752	1(ii)
36	0.93831	0.95291	6.45989	0.71873	1(ii)
37	0.93937	0.95227	6.52751	0.71995	1(ii)
38	0.94039	0.95168	6.59446	0.72117	1(ii)
39	0.94137	0.95115	6.66075	0.72240	1(ii)
40	0.94232	0.95067	6.72643	0.72362	1(ii)
41	0.94323	0.95024	6.79150	0.72485	1(ii)
42	0.94410	0.94984	6.85599	0.72608	1(ii)
43	0.94495	0.94948	6.91991	0.72731	1(ii)
44	0.94576	0.94915	6.98330	0.72853	1(ii)
45	0.93857	0.94897	7.04133	0.72973	2(iii)
46	0.93262	0.94889	7.09535	0.73078	2(iii)
47	0.92856	0.94885	7.14826	0.73171	2(iii)
48	0.92543	0.94883	7.20010	0.73252	2(iii)
49	0.92287	0.94884	7.25089	0.73322	2(iii)
50	0.92073	0.94887	7.30068	0.73384	2(iii)

(a) Case $d_0 = 5$, the maximum threshold in this range is about 0.73384.

d_1	z^*	α^*	$\bar{d}$	$\check{c}((6,d_1))$	case
6	0.93008	1.00000	6.00000	0.63708	1(i)
7	0.93008	1.00000	6.00000	0.63708	1(i)
8	0.93008	1.00000	6.00000	0.63708	1(i)
9	0.93008	1.00000	6.00000	0.63708	1(i)
10	0.93008	1.00000	6.00000	0.63708	1(i)
11	0.93008	1.00000	6.00000	0.63708	1(i)
12	0.93008	1.00000	6.00000	0.63708	1(i)
13	0.93008	1.00000	6.00000	0.63708	1(i)
14	0.93008	1.00000	6.00000	0.63708	1(i)
15	0.93008	1.00000	6.00000	0.63708	1(i)
16	0.93008	1.00000	6.00000	0.63708	1(i)
17	0.93008	1.00000	6.00000	0.63708	1(i)
18	0.93008	1.00000	6.00000	0.63708	1(i)
19	0.93008	1.00000	6.00000	0.63708	1(i)
20	0.93008	1.00000	6.00000	0.63708	1(i)
21	0.93008	1.00000	6.00000	0.63708	1(i)
22	0.93008	1.00000	6.00000	0.63708	1(i)
23	0.93008	1.00000	6.00000	0.63708	1(i)
24	0.93008	1.00000	6.00000	0.63708	1(i)
25	0.93008	1.00000	6.00000	0.63708	1(i)
26	0.93008	1.00000	6.00000	0.63708	1(i)
27	0.93088	0.99807	6.04054	0.63709	1(ii)
28	0.93237	0.99463	6.11825	0.63717	1(ii)
29	0.93379	0.99153	6.19490	0.63732	1(ii)
30	0.93514	0.98873	6.27054	0.63754	1(ii)
31	0.93642	0.98619	6.34522	0.63781	1(ii)
32	0.93765	0.98388	6.41900	0.63812	1(ii)
33	0.93881	0.98178	6.49189	0.63849	1(ii)
34	0.93993	0.97986	6.56396	0.63889	1(ii)
35	0.94100	0.97810	6.63523	0.63932	1(ii)
36	0.94202	0.97648	6.70573	0.63979	1(ii)
37	0.94301	0.97498	6.77551	0.64028	1(ii)
38	0.94395	0.97361	6.84458	0.64080	1(ii)
39	0.94486	0.97233	6.91297	0.64134	1(ii)
40	0.94573	0.97116	6.98071	0.64190	1(ii)
41	0.94657	0.97006	7.04783	0.64248	1(ii)
42	0.94738	0.96905	7.11433	0.64308	1(ii)
43	0.94816	0.96810	7.18026	0.64369	1(ii)
44	0.94892	0.96722	7.24562	0.64431	1(ii)
45	0.94965	0.96640	7.31044	0.64494	1(ii)
46	0.95035	0.96563	7.37472	0.64558	1(ii)
47	0.95103	0.96491	7.43850	0.64624	1(ii)
48	0.95170	0.96424	7.50178	0.64690	1(ii)
49	0.95233	0.96361	7.56458	0.64756	1(ii)
50	0.95295	0.96302	7.62692	0.64823	1(ii)

(b) Case $d_0 = 6$, the maximum threshold in this range is about 0.64823.

Table A.2.2.: Optimal 2-core thresholds $\check{c}(\mathbf{d})$ and corresponding optimal points (z^*, α^*) for all $\mathbf{d} = (d_0, d_1) \in \mathbb{N}^2$ with $5 \leqslant d_0 \leqslant 6$ and $d_0 \leqslant d_1 \leqslant 50$.

Nomenclature

A	set of addresses
a	address; also arbitrary variable
$\mathcal{A}$	event
a	number of attempts
α	fraction of left nodes of a certain degree in a bipartite graph
b	number of blocked nodes; also arbitrary variable
$\mathcal{B}$	event
β	bias parameter for random walk, influences how often insertion is tried on primary and backup pages; also fraction of nodes that belong to the set of blocked nodes, $\beta = b/m$
Bin	Binomial distribution / random variable
$\perp$	special value from V, if associated with a key $x \in U$ means that $x \notin S$
c	load factor, $c = n/m$
$\mathcal{C}$	event
$\check{c}$	threshold for the existence of an order generating matching, $\check{c}$ equals $\check{c}_{\ell_+}$ for $\ell_+ = 2$
$\check{c}_{\ell_+}$	threshold for the appearance of an ℓ_+-core
c_{end}	end of load factor interval
$\hat{c}$	threshold for the existence of a left-perfect matching, $\hat{c}$ equals $\hat{c}_{k,\ell}$ for $\ell = k = 1$
$\hat{c}^{d_b}_{\check{m}}$	(hypothetical) threshold for the existence of a left-perfect matching in a scenario with pages of size $\check{m}$ and d_b backup options
$\hat{c}_{k,\ell}$	threshold for the existence of a (k,ℓ)-orientation
$\circ$	concatenation operator

c_p	load factor of primary keys, $c_p = n_p/m$
c_{start}	start of load factor interval
d	number of choices, node degree, hyperedge size, matrix row weight
D	random degree
$\mathfrak{D}$	(abstract) data structure
d_b	number of choices on backup page
$\bar{d}$	average mean degree over all $x \in S$, $\bar{d} = \frac{1}{n} \cdot \sum_{x \in S} \Delta_x$
$\check{d}$	minimum degree
deg	degree of node
δ	small constant; also parameter of fit function σ
Δ	expected degree, $\Delta = \mathrm{Exp}(D)$
dens	density function
$\hat{d}$	maximum degree
d_p	number of choices on primary page
E	edge set or edge multiset
$\mathcal{E}$	event
E_{pot}	set of unsettled edges incident to a node (set of potential edges)
ε	small constant
F	set of free nodes
f	function
Fail	failure function, $\mathrm{Fail} = \mathrm{Fail}(d_y, d_z)$
fail	failure function, $\mathrm{fail} = \mathrm{fail}(d_y, d_z, b, r, i_1, \ldots, i_r)$
$\mathbb{F}$	field
$F_\bullet$	set of half-free nodes
G	graph
g	function

γ	small constant; also parameter of fit function σ
$G^{\acute{d}}_{n,\acute{m}}$	random bipartite $\acute{d}$-left-regular graph, in general different from $G^{d}_{n,m}$
$G^{\bar{d}}_{n,m}$	random bipartite left-irregular graph with expected left degree $\bar{d}$
$G^{d}_{n,m}$	random bipartite d-left-regular graph
$G^{\mathbf{d}}_{n,m,\alpha}$	random bipartite left-irregular graph
$G^{d_p,d_b}_{n,(m,\check{m})}$	random bipartite graph "with pages"
H	hypergraph
h	(hash) function
H_2	entropy to base 2
h_b	hash function that is used to determine backup pages
${}^{\text{bin}}H^{\mathbf{d}}_{m,\mathbf{p}}$	random non-uniform binomial hypergraph
$\mathcal{H}$	hash class
$\mathcal{H}_0$	null hypothesis
$\mathcal{H}^{\text{coll}}$	class of "collapse" hash functions
$\mathcal{H}^{\text{map}}$	class of "mapping" hash functions
${}^{\text{sim}}\mathcal{H}^{\mathbf{d}}_{m,n,\alpha}$	class of non-uniform hypergraphs without multiple edges
$\mathcal{H}^{\text{split}}$	class of "splitting" hash functions
h^{col}	hash function for determining a column index of a matrix
h^{coll}	hash function for collapsing the universe, $h^{\text{coll}} \in \mathcal{H}^{\text{coll}}$
$H^{\acute{d}}_{\acute{m},n}$	random $\acute{d}$-uniform hypergraph (almost uniform when of type A), in general different from $H^{d}_{m,n}$
$H^{\bar{d}}_{m,n}$	random non-uniform hypergraph with expected edge size $\bar{d}$
$H^{d}_{m,n}$	random d-uniform hypergraph (almost uniform when of type A)
$H^{\mathbf{d}}_{m,n,\alpha}$	random non-uniform hypergraph
$H^{d_p,d_b}_{(m,\check{m}),n}$	random hypergraph "with pages"
H_e	entropy to base e

h^{map}	hash function for mapping step, $h^{map} \in \mathcal{H}^{map}$
${}^{mul}H^{\mathbf{d}}_{m,n,\alpha}$	random non-uniform hypergraph with multiple edges
h_p	hash function that is used to determine primary pages
${}^{poi}H^{\mathbf{d}}_{m,\Lambda}$	random non-uniform Poisson cloning hypergraph
h^{row}	hash function for determining a row index of a matrix
h	mapping that is build upon d hash functions
${}^{sim}H^{\mathbf{d}}_{m,n,\alpha}$	random non-uniform hypergraph without multiple edges
h^{split}	hash function for splitting a key set, $h^{split} \in \mathcal{H}^{split}$
I	index set
i	number of iterations
inc	number of edges directed towards a node
ι	fraction of nodes that belong to an independent set, $\iota_j = i_j/m$
J	index set
k	integer constant, usually $l \leqslant k$; often degree
K	constant
κ	degree of independence; also configuration, i. e., sequence of parameters
key	key function
k	required number of orientations of hyperedge
l	integer constant, usually $l \leqslant k$; often degree
L	left node set of bipartite graph $G = (L \cup R, E)$
Λ	expected degree of random hypergraph node (with respect to edges of a certain size)
λ	mean of Poisson distribution
λ^{den}_{ℓ}	λ^{den}_{ℓ} is the solution of $dens(\lambda) = \ell$
λ^{key}_{c}	λ^{key}_{c} is the solution of $key(\lambda) = c$
$\mathfrak{L}$	(unordered) list

ℓ	maximum allowed indegree for orientation, related to core parameter ℓ_+ via $\ell = \ell_+ - 1$
ln	logarithm to base e
log	logarithm to base 2
ℓ_+	core parameter, related to orientation parameter ℓ via $\ell_+ = \ell + 1$
$\mathbf{M}$	matrix
M	matching, $M \subseteq E$
m	number of table cells, right nodes, hypergraph nodes, matrix columns; size of range of hash function
$\acute{m}$	same as m, but in general with different value
$\mathcal{M}$	event that a matching exists
$\check{m}$	page size
$\mathbf{M}^{\bar{d}}_{n,m}$	random binary matrix with expected row weight $\bar{d}$
$\mathbf{M}^{d}_{n,m}$	random binary matrix with d-uniform row weight (almost uniform when of type A)
$\mathbf{M}^{\mathbf{d}}_{n,m,\alpha}$	random binary matrix with non-uniform row weight
$\mathbf{M}^{d_p,d_b}_{n,(m,\check{m})}$	random binary matrix "with pages"
N	neighborhood, set of nodes
n	number of (relevant) keys, left nodes, hyperedges, matrix rows
N_b	neighborhood consisting of backup nodes only
n_b	number of keys that are assigned to their backup page (backup keys)
$n_{b\|i}$	number of keys that have primary page number i but are inserted on their backup page
$\mathbb{N}$	the set of natural numbers
N_p	neighborhood consisting of primary nodes only
n_p	number of keys that are assigned to their primary page (primary keys)
ω	weight function $\omega\colon E \to \mathbb{R}$

ω_{cf}	cost function $\omega_{cf}\colon E \to \mathbb{R}$
ω^+	weight of edge set, $\omega^+(E) = \sum_{e\in E} \omega(e)$
$\omega^{\pm}_M$	incremental weight of edge set with respect to matching M
$\oplus$	addition in abelian group, often xor; also symmetric difference
out-deg	outdegree of node
P	path; also product; also probability
p	probability; also prime number
$\mathcal{P}$	property of hypergraphs; also power set
$\mathfrak{p}$	number of pages
$\mathfrak{r}$	average number of page requests of random walk
ϕ	homomorphism
φ	Boolean formula assignment
π	permutation, ordering
Po	Poisson distribution / random variable
prio	non-negative priority (smallest$\hat{=}$highest)
Pr'	probability measure in modified probability space
q	probability
$\mathfrak{Q}$	(priority) queue
R	right node set of bipartite graph $G = (L \cup R, E)$; also range of hash function
r	length of vector; also number of independent sets
$\mathbb{R}$	the set of real numbers
$\mathcal{R}$	hash class with strong randomness properties
req	number of nodes to which an edge must be directed
ρ	probability mass function
r_p	ratio of primary keys, $r_p = n_p/n$
S	key set, $S \subseteq U$; also intermediate result or sum

s	length of vector
$\check{S}$	subset of $\check{U}$
$\mathcal{S}$	space consumption
$\mathfrak{s}$	average number of insertion steps of random walk
$\mathfrak{s}_{\text{key}}$	average number of insertion steps of random walk per key
σ	fit function
Σ_{sre}	sum of squares of residuals
$\star$	stands for sequence of symbols that has finite length
$\mathbf{t}$	table (vector)
T	set; also intermediate result or sum
$\mathcal{T}$	time consumption
$\mathcal{T}_c$	construction time
$\mathcal{T}_e$	evaluation time
$\mathbf{T}^{\text{dec}}$	lookup matrix for decoding (decompression)
$\mathbf{t}^{\text{enc}}$	encoded (compressed) vector
θ	parameter for random walk, influences total number of trials; also real value from $[0, 1)$
U	universe
u	size of universe, $u = \lvert U \rvert$
$\check{U}$	non-standard or extended universe
V	set of values, e. g., $f\colon U \to V$; also node set, e. g., $G = (V, E)$; also set of group elements
v	function value; also (hyper-)graph node
V_{pot}	set of unsaturated nodes to which an edge can be directed (set of potential nodes)
w	lower bound for the smallest increment of augmenting paths
X	discrete random variable, realization is denoted by x

x	key; also realization of X
$\check{x}$	element from $\check{U}$
ξ	failure rate
Y	discrete random variable, realization is denoted by y
y	key; also realization of Y
Z	discrete random variable, realization is denoted by z
$\mathbb{Z}$	the set of integers
$\check{z}_0$	smaller root of $g(z)$, local maximum point of $h(z)$
$\hat{z}_0$	larger root of $g(z)$, local minimum point of $h(z)$
z_{le}	left interval border for relevant z points
z_{lo}	used to determine bound for search interval
z_{l0}	point between z_{le} and $\check{z}_0$ with $h(z_{l0}) = h(\hat{z}_0)$
z_{in}	indicator point
z_{ri}	right interval border for relevant z points
z_{up}	used to determine bound for search interval
z_{0r}	point between $\hat{z}_0$ and z_{ri} with $h(z_{0r}) = h(\check{z}_0)$
ζ	scale factor that depends on some compression scheme

List of Figures

List of Tables

List of Algorithms

Bibliography

[AV88] Alok Aggarwal and Jeffrey Scott Vitter. The Input/Output Complexity of Sorting and Related Problems. *Commun. ACM*, 31(9):1116–1127, 1988. ISSN 0001-0782. January 17, 2015.
http://doi.acm.org/10.1145/48529.48535
(Cited on page 38.)

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Upper Saddle River, NJ, USA, 1993. ISBN 0-13-617549-X. August 7, 2013.
http://dl.acm.org/citation.cfm?id=137406
(Cited on pages 46, 71, and 72.)

[AZ10] Martin Aigner and Günter M. Ziegler. *Proofs from THE BOOK*. Springer, 4th edition, 2010. ISBN 978-3-642-00855-9. August 7, 2013.
http://dx.doi.org/10.1007/978-3-642-00856-6
(Cited on page 192.)

[ASA+09] Dan A. Alcantara, Andrei Sharf, Fatemeh Abbasinejad, Shubhabrata Sengupta, Michael Mitzenmacher, John D. Owens, and Nina Amenta. Real-Time Parallel Hashing on the GPU. *ACM Trans. Graph.*, 28(5): 154:1–154:9, 2009. ISSN 0730-0301. July 29, 2013.
http://dx.doi.org/10.1145/1618452.1618500
(Cited on pages 2 and 31.)

[AP11] Rasmus Resen Amossen and Rasmus Pagh. A New Data Layout for Set Intersection on GPUs. In *Proc. 25th IPDPS*, pages 698–708. IEEE, 2011. ISBN 978-1-61284-372-8. December 6, 2014.
http://dx.doi.org/10.1109/IPDPS.2011.71
(Cited on page 23.)

[ANS09] Yuriy Arbitman, Moni Naor, and Gil Segev. De-amortized Cuckoo Hashing: Provable Worst-Case Performance and Experimental Results. In *Proc. 36th ICALP (1)*, volume 5555 of *LNCS*, pages 107–118. Springer, 2009. ISBN 978-3-642-02926-4. July 23, 2013.
http://dx.doi.org/10.1007/978-3-642-02927-1_11
(Cited on page 38.)

[ANS10] Yuriy Arbitman, Moni Naor, and Gil Segev. Backyard Cuckoo Hashing: Constant Worst-Case Operations with a Succinct Representation. In

Proc. 51th FOCS, pages 787–796. IEEE Computer Society, 2010. ISBN 978-0-7695-4244-7. ISSN 0272-5428. July 23, 2013.
http://dx.doi.org/10.1109/FOCS.2010.80
(Cited on page 38.)

[Aum10] Martin Aumüller. An Alternative Analysis of Cuckoo Hashing with a Stash and Realistic Hash Functions. Diplomarbeit, Technische Universität Ilmenau, 2010. July 22, 2013.
http://gso.gbv.de/DB=2.1/PPNSET?PPN=622447068
(Cited on pages 38, 188, and 189.)

[ADR09] Martin Aumüller, Martin Dietzfelbinger, and Michael Rink. Experimental Variations of a Theoretically Good Retrieval Data Structure. In *Proc. 17th ESA*, volume 5757 of *LNCS*, pages 742–751. Springer, 2009. ISBN 978-3-642-04127-3. July 25, 2013.
http://dx.doi.org/10.1007/978-3-642-04128-0_66
(Cited on pages xiii, 101, 102, and 115.)

[ADW12] Martin Aumüller, Martin Dietzfelbinger, and Philipp Woelfel. Explicit and Efficient Hash Families Suffice for Cuckoo Hashing with a Stash. In *Proc. 20th ESA*, volume 7501 of *LNCS*, pages 108–120. Springer, 2012. ISBN 978-3-642-33089-6. July 22, 2013.
http://dx.doi.org/10.1007/978-3-642-33090-2_11
(Cited on pages 7, 38, 184, 185, and 189.)

[ADW13] Martin Aumüller, Martin Dietzfelbinger, and Philipp Woelfel. Explicit and Efficient Hash Families Suffice for Cuckoo Hashing with a Stash. *Algorithmica*, pages 1–29, 2013. ISSN 0178-4617. January 11, 2014.
http://dx.doi.org/10.1007/s00453-013-9840-x
(Cited on pages 38 and 216.)

[AFPR59] T. L. Austin, R. E. Fagen, W. F. Penney, and John Riordan. The Number of Components in Random Linear Graphs. *The Annals of Mathematical Statistics*, 30(3):747–754, 1959. August 30, 2013.
http://dx.doi.org/10.1214/aoms/1177706204
(Cited on page 121.)

[ABKU94] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced Allocations (Extended abstract). In *Proc 26th. STOC*, pages 593–602. ACM, 1994. ISBN 0-89791-663-8. July 23, 2013.
http://dx.doi.org/10.1145/195058.195412
(Cited on page 36.)

[ABKU99] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced Allocations. *SIAM J. Comput.*, 29(1):180–200, 1999. July 31, 2013. http://dx.doi.org/10.1137/S0097539795288490
(Cited on pages 36 and 37.)

[BMST04] Holger Bast, Kurt Mehlhorn, Guido Schäfer, and Hisao Tamaki. Matching Algorithms Are Fast in Sparse Random Graphs. In *Proc. 21st STACS*, volume 2996 of *LNCS*, pages 81–92. Springer, 2004. ISBN 3-540-21236-1. July 22, 2013. http://dx.doi.org/10.1007/978-3-540-24749-4_8
(Cited on page 72.)

[BMST06] Holger Bast, Kurt Mehlhorn, Guido Schäfer, and Hisao Tamaki. Matching Algorithms Are Fast in Sparse Random Graphs. *Theory Comput. Syst.*, 39(1):3–14, 2006. July 22, 2013. http://dx.doi.org/10.1007/s00224-005-1254-y
(Cited on page 72.)

[BBPV09] Djamal Belazzougui, Paolo Boldi, Rasmus Pagh, and Sebastiano Vigna. Monotone Minimal Perfect Hashing: Searching a Sorted Table with O(1) Accesses. In *Proc. 20th SODA*, pages 785–794. SIAM, 2009. July 24, 2013. http://doi.acm.org/10.1145/1496770.1496856
(Cited on page 110.)

[BBD09] Djamal Belazzougui, Fabiano Cupertino Botelho, and Martin Dietzfelbinger. Hash, Displace, and Compress. In *Proc. 17th ESA*, volume 5757 of *LNCS*, pages 682–693. Springer, 2009. ISBN 978-3-642-04127-3. July 23, 2013. http://dx.doi.org/10.1007/978-3-642-04128-0_61
(Cited on pages 112, 114, and 189.)

[Ber57] Claude Berge. Two Theorems in Graph Theory. *Proceedings of the National Academy of Sciences*, 43(9):842–844, 1957. July 29, 2013. http://www.ncbi.nlm.nih.gov/pmc/articles/PMC534337/
(Cited on page 42.)

[Bey12] Stephan Beyer. Analysis of the Linear Probing Variant of Cuckoo Hashing. Diplomarbeit, Technische Universität Ilmenau, 2012. July 22, 2013. http://gso.gbv.de/DB=2.1/PPNSET?PPN=685166759
(Cited on pages 33 and 189.)

[Blo70] Burton H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM*, 13(7):422–426, 1970. ISSN 0001-0782. July 31, 2013. http://doi.acm.org/10.1145/362686.362692
(Cited on pages 5, 11, 34, 83, and 115.)

[BK06] Tom Bohman and Jeong Han Kim. A Phase Transition for Avoiding a Giant Component. *Random Struct. Algorithms*, 28(2):195–214, 2006. ISSN 1042-9832. August 7, 2013.
http://dx.doi.org/10.1002/rsa.20085
(Cited on page 16.)

[Bot08] Fabiano Cupertino Botelho. *Near-Optimal Space Perfect Hashing Algorithms.* PhD thesis, Federal University of Minas Gerais, 2008. July 22, 2013.
http://homepages.dcc.ufmg.br/~fbotelho/en/pub/thesis.pdf
(Cited on pages 111 and 116.)

[BKZ05] Fabiano Cupertino Botelho, Yoshiharu Kohayakawa, and Nivio Ziviani. A Practical Minimal Perfect Hashing Method. In *Proc. 4th WEA*, volume 3503 of *LNCS*, pages 488–500. Springer, 2005. ISBN 3-540-25920-1. August 11, 2013. http://dx.doi.org/10.1007/11427186_42
(Cited on page 114.)

[BPZ07] Fabiano Cupertino Botelho, Rasmus Pagh, and Nivio Ziviani. Simple and Space-Efficient Minimal Perfect Hash Functions. In *Proc. 10th WADS*, volume 4619 of *LNCS*, pages 139–150. Springer, 2007. ISBN 978-3-540-73948-7. July 23, 2013.
http://dx.doi.org/10.1007/978-3-540-73951-7_13
(Cited on pages 5, 106, 113, and 189.)

[BPZ13] Fabiano Cupertino Botelho, Rasmus Pagh, and Nivio Ziviani. Practical Perfect Hashing in Nearly Optimal Space. *Inf. Syst.*, 38(1):108–131, 2013. ISSN 0306-4379. July 23, 2013.
http://dx.doi.org/10.1016/j.is.2012.06.002
(Cited on pages 5, 106, 113, 114, and 116.)

[BWZ12] Fabiano Cupertino Botelho, Nicholas C. Wormald, and Nivio Ziviani. Cores of random r-partite hypergraphs. *Inf. Process. Lett.*, 112(8-9): 314–319, 2012. ISSN 0020-0190. July 29, 2013.
http://dx.doi.org/10.1016/j.ipl.2011.10.017
(Cited on page 16.)

[BM01] Andrei Z. Broder and Michael Mitzenmacher. Using Multiple Hash Functions to Improve IP Lookups. In *Proc. 20th INFOCOM*, pages 1454–1463. IEEE, 2001. ISBN 0-7803-7016-3. July 23, 2013.
http://dx.doi.org/10.1109/INFCOM.2001.916641
(Cited on pages 36 and 37.)

[BM03] Andrei Z. Broder and Michael Mitzenmacher. Network Applications of Bloom Filters: A Survey. *Internet Mathematics*, 1(4):485–509, 2003.

August 2, 2013.
http://dx.doi.org/10.1080/15427951.2004.10129096
(Cited on pages 115 and 116.)

[CSW07] Julie Anne Cain, Peter Sanders, and Nicholas C. Wormald. The Random Graph Threshold for k-orientiability and a Fast Algorithm for Optimal Multiple-Choice Allocation. In *Proc. 18th SODA*, pages 469–476. SIAM, 2007. ISBN 978-0-898716-24-5. July 22, 2013.
http://dl.acm.org/citation.cfm?id=1283383.1283433
(Cited on pages 4, 23, 24, and 189.)

[Cal97] Neil J. Calkin. Dependent Sets of Constant Weight Binary Vectors. *Combinatorics, Probability and Computing*, 6(3):263–271, 1997. ISSN 0963-5483. August 8, 2013.
http://journals.cambridge.org/article_S0963548397003040
(Cited on pages 113, 197, 201, 202, 203, 208, 210, 211, 212, and 213.)

[CW77] Larry Carter and Mark N. Wegman. Universal Classes of Hash Functions (Extended Abstract). In *Proc. 9th STOC*, pages 106–112. ACM, 1977. July 25, 2013. http://dx.doi.org/10.1145/800105.803400
(Cited on pages 187 and 188.)

[CW79] Larry Carter and Mark N. Wegman. Universal Classes of Hash Functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979. ISSN 0022-0000. August 6, 2013. http://dx.doi.org/10.1016/0022-0000(79)90044-8
(Cited on pages 186, 187, and 190.)

[CC08a] Denis Xavier Charles and Kumar Chellapilla. Bloomier Filters: A second look. *CoRR*, abs/0807.0928, 2008. July 31, 2013.
http://arxiv.org/abs/0807.0928
(Cited on page 113.)

[CC08b] Denis Xavier Charles and Kumar Chellapilla. Bloomier Filters: A Second Look. In *Proc. 16th ESA*, volume 5193 of *LNCS*, pages 259–270. Springer, 2008. ISBN 978-3-540-87743-1. July 29, 2013.
http://dx.doi.org/10.1007/978-3-540-87744-8_22
(Cited on pages 113 and 115.)

[CKRT04] Bernard Chazelle, Joe Kilian, Ronitt Rubinfeld, and Ayellet Tal. The Bloomier Filter: An Efficient Data Structure for Static Support Lookup Tables. In *Proc. 15th SODA*, pages 30–39. SIAM, 2004. ISBN 0-89871-558-X. July 23, 2013.
http://dl.acm.org/citation.cfm?id=982792.982797
(Cited on pages 5, 95, 97, 101, 106, 113, 114, and 189.)

[CG85] Benny Chor and Oded Goldreich. Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity (Extended Abstract). In *Proc. 26th FOCS*, pages 429–442. IEEE Computer Society, 1985. ISSN 0272-5428. August 7, 2013. http://dx.doi.org/10.1109/SFCS.1985.62
(Cited on page 186.)

[CG88] Benny Chor and Oded Goldreich. Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity. *SIAM J. Comput.*, 17(2):230–261, 1988. ISSN 0097-5397. August 7, 2013. http://dx.doi.org/10.1137/0217015
(Cited on page 186.)

[CV08] Kai-Min Chung and Salil P. Vadhan. Tight Bounds for Hashing Block Sources. In *Proc. 11th APPROX / 12th RANDOM*, volume 5171 of *LNCS*, pages 357–370. Springer, 2008. ISBN 978-3-540-85362-6. July 22, 2013. http://dx.doi.org/10.1007/978-3-540-85363-3_29
(Cited on pages 186 and 187.)

[CK09] Jeffery S. Cohen and Daniel M. Kane. Bounds on the Independence Required for Cuckoo Hashing, 2009. submitted to ACM TALG; August 14, 2012. http://math.stanford.edu/~dankane/cuchkoohashing.pdf
(Cited on page 188.)

[Coo04] Colin Cooper. The Cores of Random Hypergraphs with a Given Degree Sequence. *Random Struct. Algorithms*, 25(4):353–375, 2004. ISSN 1098-2418. July 29, 2013. http://dx.doi.org/10.1002/rsa.20040
(Cited on page 16.)

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2nd edition, 2001. ISBN 0-262-03293-7. August 7, 2013. http://dl.acm.org/citation.cfm?id=500824
(Cited on pages 35, 189, and 193.)

[CT06] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 2nd edition, 2006. ISBN 978-0-471-24195-9. August 8, 2013. http://gso.gbv.de/DB=2.1/PPNSET?PPN=485617870
(Cited on page 190.)

[CHM92] Zbigniew J. Czech, George Havas, and Bohdan S. Majewski. An optimal algorithm for generating minimal perfect hash functions. *Inf. Process. Lett.*, 43(5):257–264, 1992. July 22, 2013. http://dx.doi.org/10.1016/0020-0190(92)90220-P
(Cited on pages 95, 99, 113, 114, and 217.)

[CHM97] Zbigniew J. Czech, George Havas, and Bohdan S. Majewski. Fundamental Study Perfect hashing. *Theor. Comput. Sci.*, 182(1-2):1–143, 1997. July 22, 2013. http://itee.uq.edu.au/~havas/chm.pdf
(Cited on pages 99 and 111.)

[CRS03] Artur Czumaj, Chris Riley, and Christian Scheideler. Perfectly Balanced Allocation. In *Proc. 6th APPROX - 7th RANDOM*, volume 2764 of *LNCS*, pages 240–251. Springer, 2003. ISBN 3-540-40770-7. July 23, 2013. http://dx.doi.org/10.1007/978-3-540-45198-3_21
(Cited on page 37.)

[CS97] Artur Czumaj and Volker Stemann. Randomized Allocation Processes (Extended Abstract). In *Proc. 38th FOCS*, pages 194–203. IEEE Computer Society, 1997. July 24, 2013. http://dl.acm.org/citation.cfm?id=795663.796355
(Cited on page 20.)

[CS01] Artur Czumaj and Volker Stemann. Randomized allocation processes. *Random Struct. Algorithms*, 18(4):297–331, 2001. ISSN 1098-2418. July 24, 2013. http://dx.doi.org/10.1002/rsa.1011
(Cited on page 20.)

[DMPP06] Erik D. Demaine, Friedhelm Meyer auf der Heide, Rasmus Pagh, and Mihai Pătraşcu. De Dictionariis Dynamicis Pauco Spatio Utentibus (*lat.* On Dynamic Dictionaries Using Little Space). In *LATIN*, volume 3887 of *LNCS*, pages 349–361. Springer, 2006. ISBN 3-540-32755-X. July 23, 2013. http://dx.doi.org/10.1007/11682462_34
(Cited on page 1.)

[DM03] Luc Devroye and Pat Morin. Cuckoo hashing: Further analysis. *Inf. Process. Lett.*, 86(4):215–219, 2003. ISSN 0020-0190. July 29, 2013. http://dx.doi.org/10.1016/S0020-0190(02)00500-8
(Cited on pages 20 and 37.)

[Die07] Martin Dietzfelbinger. Design Strategies for Minimal Perfect Hash Functions. In *Proc. 4th SAGA*, volume 4665 of *LNCS*, pages 2–17. Springer, 2007. ISBN 978-3-540-74870-0. July 25, 2013. http://dx.doi.org/10.1007/978-3-540-74871-7_2
(Cited on pages 111, 185, 189, and 193.)

[DGM+09] Martin Dietzfelbinger, Andreas Goerdt, Michael Mitzenmacher, Andrea Montanari, Rasmus Pagh, and Michael Rink. Tight Thresholds for Cuckoo Hashing via XORSAT. *CoRR*, abs/0912.0287, 2009. October 10, 2012. http://arxiv.org/abs/0912.0287
(Cited on pages xiii, 28, 29, 46, 54, 100, and 130.)

[DGM+10] Martin Dietzfelbinger, Andreas Goerdt, Michael Mitzenmacher, Andrea Montanari, Rasmus Pagh, and Michael Rink. Tight Thresholds for Cuckoo Hashing via XORSAT. In *Proc. 37th ICALP (1)*, volume 6198 of *LNCS*, pages 213–225. Springer, 2010. ISBN 978-3-642-14164-5. July 25, 2013. http://dx.doi.org/10.1007/978-3-642-14165-2_19
(Cited on pages xiii, 3, 21, 23, 28, 29, 54, 67, 100, 103, 104, 113, 129, 130, 180, and 189.)

[DH01] Martin Dietzfelbinger and Torben Hagerup. Simple Minimal Perfect Hashing in Less Space. In *Proc. 9th ESA*, volume 2161 of *LNCS*, pages 109–120. Springer, 2001. ISBN 3-540-42493-8. July 21, 2013. http://dx.doi.org/10.1007/3-540-44676-1_9
(Cited on pages 112 and 114.)

[DHKP97] Martin Dietzfelbinger, Torben Hagerup, Jyrki Katajainen, and Martti Penttonen. A Reliable Randomized Algorithm for the Closest-Pair Problem. *J. Algorithms*, 25(1):19–51, 1997. ISSN 0196-6774. July 29, 2013. http://dx.doi.org/10.1006/jagm.1997.0873
(Cited on page 193.)

[DKM+88] Martin Dietzfelbinger, Anna R. Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Heide, Hans Rohnert, and Robert Endre Tarjan. Dynamic Perfect Hashing: Upper and Lower Bounds. In *Proc. 29th FOCS*, pages 524–531. IEEE Computer Society, 1988. ISBN 0-8186-0877-3. July 22, 2013. http://doi.ieeecomputersociety.org/10.1109/SFCS.1988.21968
(Cited on pages 2 and 110.)

[DKM+94] Martin Dietzfelbinger, Anna R. Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Heide, Hans Rohnert, and Robert Endre Tarjan. Dynamic Perfect Hashing: Upper and Lower Bounds. *SIAM J. Comput.*, 23(4):738–761, 1994. ISSN 0097-5397. July 29, 2013. http://dx.doi.org/10.1137/S0097539791194094
(Cited on pages 2 and 110.)

[DM90] Martin Dietzfelbinger and Friedhelm Meyer auf der Heide. A new universal class of hash functions and dynamic hashing in real time. In *Proc. 17th ICALP*, volume 443 of *LNCS*, pages 6–19. Springer, 1990. ISBN 3-540-52826-1. July 25, 2013. http://dx.doi.org/10.1007/BFb0032018
(Cited on pages 184, 189, 199, and 201.)

[DM92] Martin Dietzfelbinger and Friedhelm Meyer auf der Heide. Dynamic Hashing in Real Time. In Johannes Buchmann, Harald Ganzinger, and Wolfgang J. Paul, editors, *Informatik. Festschrift zum 60. Geburtstag von Günter Hotz*, volume 1 of *TEUBNER-TEXTE zur Informatik*,

pages 95–119. Vieweg+Teubner Verlag, 1992. ISBN 978-3-8154-2033-1. November 6, 2014.
http://dx.doi.org/10.1007/978-3-322-95233-2_7
(Cited on pages 184 and 201.)

[DMR11a] Martin Dietzfelbinger, Michael Mitzenmacher, and Michael Rink. Cuckoo Hashing with Pages. In *Proc. 19th ESA*, volume 6942 of *LNCS*, pages 615–627. Springer, 2011. ISBN 978-3-642-23718-8. July 23, 2013.
http://dx.doi.org/10.1007/978-3-642-23719-5_52
(Cited on pages xiii, 33, and 69.)

[DMR11b] Martin Dietzfelbinger, Michael Mitzenmacher, and Michael Rink. Cuckoo Hashing with Pages. *CoRR*, abs/1104.5111, 2011. October 10, 2012.
http://arxiv.org/abs/1104.5111
(Cited on pages xiii and 69.)

[DP08a] Martin Dietzfelbinger and Rasmus Pagh. Succinct Data Structures for Retrieval and Approximate Membership. *CoRR*, abs/0803.3693, 2008. April 10, 2013. http://arxiv.org/abs/0803.3693
(Cited on pages 101, 113, and 115.)

[DP08b] Martin Dietzfelbinger and Rasmus Pagh. Succinct Data Structures for Retrieval and Approximate Membership (Extended Abstract). In *Proc. 35th ICALP (1)*, volume 5125 of *LNCS*, pages 385–396. Springer, 2008. ISBN 978-3-540-70574-1. July 25, 2013.
http://dx.doi.org/10.1007/978-3-540-70575-8_32
(Cited on pages 101, 113, 115, 189, and 195.)

[DR09] Martin Dietzfelbinger and Michael Rink. Applications of a Splitting Trick. In *Proc. 36th ICALP (1)*, volume 5555 of *LNCS*, pages 354–365. Springer, 2009. ISBN 978-3-642-02926-4. July 25, 2013.
http://dx.doi.org/10.1007/978-3-642-02927-1_30
(Cited on pages xiii, 185, 189, and 194.)

[DR12a] Martin Dietzfelbinger and Michael Rink. Towards Optimal Degree-Distributions for Left-Perfect Matchings in Random Bipartite Graphs. In *Proc. 7th CSR*, volume 7353 of *LNCS*, pages 99–111. Springer, 2012. ISBN 978-3-642-30641-9. July 25, 2013.
http://dx.doi.org/10.1007/978-3-642-30642-6_11
(Cited on pages xiii, 29, 30, 53, and 54.)

[DR12b] Martin Dietzfelbinger and Michael Rink. Towards Optimal Degree-distributions for Left-perfect Matchings in Random Bipartite

Graphs. *CoRR*, abs/1203.1506, 2012. October 10, 2012.
http://arxiv.org/abs/1203.1506
(Cited on page xiii.)

[DS09] Martin Dietzfelbinger and Ulf Schellbach. On Risks of Using Cuckoo Hashing with Simple Universal Hash Classes. In *Proc. 20th SODA*, pages 795–804. SIAM, 2009. July 25, 2013.
http://dl.acm.org/citation.cfm?id=1496770.1496857
(Cited on page 188.)

[DW05] Martin Dietzfelbinger and Christoph Weidling. Balanced Allocation and Dictionaries with Tightly Packed Constant Size Bins. In *Proc. 32nd ICALP*, volume 3580 of *LNCS*, pages 166–178. Springer, 2005. ISBN 3-540-27580-0. July 25, 2013.
http://dx.doi.org/10.1007/11523468_14
(Cited on page 32.)

[DW07] Martin Dietzfelbinger and Christoph Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. *Theor. Comput. Sci.*, 380(1-2):47–68, 2007. July 25, 2013.
http://dx.doi.org/10.1016/j.tcs.2007.02.054
(Cited on pages 32, 185, and 189.)

[DW03] Martin Dietzfelbinger and Philipp Woelfel. Almost Random Graphs with Simple Hash Functions. In *Proc. 35th STOC*, pages 629–638. ACM, 2003. ISBN 1-58113-674-9. July 22, 2013.
http://dx.doi.org/10.1145/780542.780634
(Cited on pages 184, 188, 189, 198, and 216.)

[DPT10] Yevgeniy Dodis, Mihai Pătraşcu, and Mikkel Thorup. Changing Base without Losing Space. In *Proc. 42nd STOC*, pages 593–602. ACM, 2010. ISBN 978-1-4503-0050-6. July 22, 2013.
http://dx.doi.org/10.1145/1806689.1806771
(Cited on page 173.)

[DK12] Michael Drmota and Reinhard Kutzelnigg. A Precise Analysis of Cuckoo Hashing. *ACM Transactions on Algorithms*, 8(2):11:1–11:36, 2012. ISSN 1549-6325. July 29, 2013.
http://doi.acm.org/10.1145/2151171.2151174
(Cited on pages 3, 20, 37, and 218.)

[DP09] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1st edition, 2009. ISBN

978-0-521-88427-3. August 7, 2013.
http://dl.acm.org/citation.cfm?id=1568639
(Cited on page 69.)

[DM02] Olivier Dubois and Jacques Mandler. The 3-XORSAT Threshold. In *Proc. 43rd FOCS*, pages 769–778. IEEE Computer Society, 2002. ISBN 0-7695-1822-2. July 25, 2013.
http://dx.doi.org/10.1109/SFCS.2002.1182002
(Cited on pages 100 and 113.)

[ER60] Paul Erdős and Alfréd Rényi. On the Evolution of Random Graphs. *Publ. Math. Inst. Hung. Acad. Sci.*, 5:17–61, 1960. July 29, 2013.
http://www.renyi.hu/~p_erdos/1961-15.pdf
(Cited on pages 99, 121, and 122.)

[ER61] Paul Erdős and Alfréd Rényi. On a Classical Problem of Probability Theory. *Publ. Math. Inst. Hung. Acad. Sci., Ser. A 6*, pages 215–220, 1961. July 29, 2013. http://www.renyi.hu/~p_erdos/Erdos.html
(Cited on page 20.)

[EMM06] Úlfar Erlingsson, Mark Manasse, and Frank Mcsherry. A cool and practical alternative to traditional hash tables. In *Proc. 7th WDAS*, 2006. July 25, 2013. http://www.ru.is/faculty/ulfar/CuckooHash.pdf
(Cited on page 2.)

[FCAB98] Li Fan, Pei Cao, Jussara M. Almeida, and Andrei Z. Broder. Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. In *Proc. SIGCOMM '98*, volume 28, pages 254–265. ACM, 1998. ISBN 1-58113-003-1. ISSN 0146-4833. July 25, 2013.
http://doi.acm.org/10.1145/285243.285287
(Cited on page 89.)

[FCAB00] Li Fan, Pei Cao, Jussara M. Almeida, and Andrei Z. Broder. Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol. *IEEE/ACM TON*, 8(3):281–293, 2000. ISSN 1063-6692. July 25, 2013.
http://dx.doi.org/10.1109/90.851975
(Cited on page 89.)

[FR07] Daniel Fernholz and Vijaya Ramachandran. The k-orientability Thresholds for $G_{n,p}$. In *Proc. 18th SODA*, pages 459–468. SIAM, 2007. ISBN 978-0-898716-24-5. July 22, 2013.
http://dl.acm.org/citation.cfm?id=1283383.1283432
(Cited on pages 23 and 189.)

[FPSS03] Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul G. Spirakis. Space Efficient Hash Tables with Worst Case Constant Access Time. In *Proc. 20th STACS*, volume 2607 of *LNCS*, pages 271–282. Springer, 2003. ISBN 3-540-00623-0. July 25, 2013. http://dx.doi.org/10.1007/3-540-36494-3_25
(Cited on pages 2 and 38.)

[FPSS05] Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul G. Spirakis. Space Efficient Hash Tables with Worst Case Constant Access Time. *Theory Comput. Syst.*, 38(2):229–248, 2005. July 25, 2013. http://dx.doi.org/10.1007/s00224-004-1195-x
(Cited on pages 2, 4, 5, 17, 20, 22, 38, 74, 107, 185, and 189.)

[FKP11] Nikolaos Fountoulakis, Megha Khosla, and Konstantinos Panagiotou. The Multiple-Orientability Thresholds for Random Hypergraphs. In *Proc. 22nd SODA*, pages 1222–1236. SIAM, 2011. July 22, 2013. http://www.siam.org/proceedings/soda/2011/SODA11_092_fountoulakisn.pdf
(Cited on pages 24 and 189.)

[FP10] Nikolaos Fountoulakis and Konstantinos Panagiotou. Orientability of Random Hypergraphs and the Power of Multiple Choices. In *Proc. 37th ICALP (1)*, volume 6198 of *LNCS*, pages 348–359. Springer, 2010. ISBN 978-3-642-14164-5. July 22, 2013. http://dx.doi.org/10.1007/978-3-642-14165-2_30
(Cited on pages 21, 128, and 189.)

[FP12] Nikolaos Fountoulakis and Konstantinos Panagiotou. Sharp Load Thresholds for Cuckoo Hashing. *Random Struct. Algorithms*, 41(3): 306–333, 2012. ISSN 1042-9832. July 31, 2013. http://dx.doi.org/10.1002/rsa.20426
(Cited on pages 3, 20, 21, and 23.)

[FPS10] Nikolaos Fountoulakis, Konstantinos Panagiotou, and Angelika Steger. On the Insertion Time of Cuckoo Hashing. *CoRR*, abs/1006.1231, 2010. March 13, 2013. http://arxiv.org/abs/1006.1231
(Cited on page 34.)

[FCDH90] Edward A. Fox, Qi Fan Chen, Amjad M. Daoud, and Lenwood S. Heath. Order Preserving Minimal Perfect Hash Functions and Information Retrieval. In *Proc. 13th SIGIR*, pages 279–311. ACM, 1990. ISBN 0-89791-408-2. July 22, 2013. http://dx.doi.org/10.1145/96749.98233
(Cited on pages 110 and 114.)

[FCDH91] Edward A. Fox, Qi Fan Chen, Amjad M. Daoud, and Lenwood S. Heath. Order-preserving Minimal Perfect Hash Functions and Information Retrieval. *ACM Trans. Inf. Syst.*, 9(3):281–308, 1991. ISSN 1046-8188. July 22, 2013. http://dx.doi.org/10.1145/125187.125200
(Cited on page 114.)

[FCHD89] Edward A. Fox, Qi Fan Chen, Lenwood S. Heath, and S. Datta. A More Cost Effective Algorithm for Finding Perfect Hash Functions. In *Proc. 17th CSC*, pages 114–122. ACM, 1989. July 22, 2013. http://dx.doi.org/10.1145/75427.75440
(Cited on page 114.)

[FHCD92] Edward A. Fox, Lenwood S. Heath, Qi Fan Chen, and Amjad M. Daoud. Practical minimal perfect hash functions for large databases. *Commun. ACM*, 35(1):105–121, 1992. July 22, 2013. http://dx.doi.org/10.1145/129617.129623
(Cited on page 114.)

[FK84] Michael L. Fredman and János Komlós. On the Size of Separating Systems and Families of Perfect Hash Functions. *SIAM Journal on Algebraic Discrete Methods*, 5(1):61–68, 1984. ISSN 0196-5212. July 23, 2013. http://dx.doi.org/10.1137/0605009
(Cited on pages 109 and 116.)

[FKS82] Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with O(1) worst case access time. In *Proc. 23rd FOCS*, pages 165–169. IEEE Computer Society, 1982. July 22, 2013. http://doi.ieeecomputersociety.org/10.1109/SFCS.1982.39
(Cited on page 2.)

[FKS84] Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a Sparse Table with O(1) Worst Case Access Time. *J. ACM*, 31(3): 538–544, 1984. August 7, 2013. http://dx.doi.org/10.1145/828.1884
(Cited on pages 2, 110, and 184.)

[FM09] Alan M. Frieze and Páll Melsted. Maximum Matchings in Random Bipartite Graphs and the Space Utilization of Cuckoo Hashtables. *CoRR*, abs/0910.5535, 2009. October 11, 2012. http://arxiv.org/abs/0910.5535
(Cited on pages 21 and 189.)

[FM12] Alan M. Frieze and Páll Melsted. Maximum Matchings in Random Bipartite Graphs and the Space Utilization of Cuckoo Hash Tables.

Random Struct. Algorithms, 41(3):334–364, 2012. ISSN 1042-9832. July 29, 2013. http://dx.doi.org/10.1002/rsa.20427
(Cited on pages 3, 21, and 23.)

[FMM11] Alan M. Frieze, Páll Melsted, and Michael Mitzenmacher. An Analysis of Random-Walk Cuckoo Hashing. *SIAM J. Comput.*, 40(2):291–308, 2011. July 29, 2013. http://dx.doi.org/10.1137/090770928
(Cited on page 34.)

[GDT+11] Mark Galassi, Jim Davies, James Theiler, Brian Gough, Gerard Jungman, Patrick Alken, Michael Booth, and Fabrice Rossi. *GNU Scientific Library Reference Manual*. Free Software Foundation, 1.15 edition, 2011. April 23, 2013. http://www.gnu.org/software/gsl/manual/
(Cited on page 50.)

[GW10] Pu Gao and Nicholas C. Wormald. Load Balancing and Orientability Thresholds for Random Hypergraphs. In *Proc. 42nd STOC*, pages 97–104. ACM, 2010. ISBN 978-1-4503-0050-6. July 22, 2013. http://dx.doi.org/10.1145/1806689.1806705
(Cited on pages 24 and 189.)

[Gen03] James E. Gentle. *Random Number Generation and Monte Carlo Methods*. Statistics and Computing. Springer, 2nd edition, 2003. ISBN 0-387-00178-6. August 7, 2013. http://dx.doi.org/10.1007/b97336
(Cited on page 27.)

[Gil59] Edgar Nelson Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, 1959. August 31, 2013. http://dx.doi.org/10.1214/aoms/1177706098
(Cited on page 122.)

[Gon81] Gaston H. Gonnet. Expected Length of the Longest Probe Sequence in Hash Code Searching. *J. ACM*, 28(2):289–304, 1981. ISSN 0004-5411. December 12, 2014. http://doi.acm.org/10.1145/322248.322254
(Cited on page 36.)

[GL88] Gaston H. Gonnet and Per-Åke Larson. External Hashing with Limited Internal Storage. *J. ACM*, 35(1):161–184, 1988. ISSN 0004-5411. January 13, 2015. http://doi.acm.org/10.1145/42267.42274
(Cited on page 39.)

[GM79] Gaston H. Gonnet and J. Ian Munro. Efficient Ordering of Hash Tables. *SIAM J. Comput.*, 8(3):463–478, 1979. July 29, 2013. http://dx.doi.org/10.1137/0208038
(Cited on page 37.)

[GM11] Michael T. Goodrich and Michael Mitzenmacher. Invertible Bloom Lookup Tables. In *Proc. 49th Communication, Control, and Computing (Allerton)*, pages 792–799. IEEE, 2011. ISBN 978-1-4577-1817-5. July 23, 2013. http://dx.doi.org/10.1109/Allerton.2011.6120248
(Cited on pages 104 and 189.)

[GS89] Marco Gori and Giovanni Soda. An Algebraic Approach to Cichelli's Perfect Hashing. *BIT*, 29(1):2–13, 1989. ISSN 0006-3835. July 23, 2013. http://dx.doi.org/10.1007/BF01932700
(Cited on page 113.)

[Gou72] Henry W. Gould. *Combinatorial Identities: a standardized set of tables listing 500 binomial coefficient summations*. Morgantown, W. Va., 1972. August 24, 2013. http://gso.gbv.de/DB=2.1/PPNSET?PPN=025916785
(Cited on page 207.)

[Hag98] Torben Hagerup. Sorting and Searching on the Word RAM. In *Proc. 15th STACS*, volume 1373 of *LNCS*, pages 366–398. Springer, 1998. ISBN 3-540-64230-7. July 22, 2013. http://dx.doi.org/10.1007/BFb0028575
(Cited on page 10.)

[HT01] Torben Hagerup and Torsten Tholey. Efficient Minimal Perfect Hashing in Nearly Minimal Space. In *Proc. 18th STACS*, volume 2010 of *LNCS*, pages 317–326. Springer, 2001. ISBN 3-540-41695-1. July 22, 2013. http://dx.doi.org/10.1007/3-540-44693-1_28
(Cited on pages 111, 112, and 185.)

[Hal35] Philip Hall. On Representatives of Subsets. *Journal of the London Mathematical Society*, 10:26–30, 1935. August 7, 2013. http://dx.doi.org/10.1112/jlms/s1-10.37.26
(Cited on page 18.)

[HMWC93] George Havas, Bohdan S. Majewski, Nicholas C. Wormald, and Zbigniew J. Czech. Graphs, hypergraphs and hashing. In *Proc. 19th WG*, volume 790 of *LNCS*, pages 153–165. Springer, 1993. ISBN 3-540-57899-4. July 22, 2013. http://dx.doi.org/10.1007/3-540-57899-4_49
(Cited on page 113.)

[Hoe63] Wassily Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *J. Amer. Statist. Assoc.*, 58(301):13–30, 1963. ISSN 0162-1459. July 29, 2013. http://www.jstor.org/stable/2282952
(Cited on page 200.)

[HK71] John E. Hopcroft and Richard M. Karp. A $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. In *Proc. 12th SWAT (FOCS)*, pages 122–125. IEEE Computer Society, 1971. ISSN 0272-4847. July 31, 2013. http://dx.doi.org/10.1109/SWAT.1971.1
(Cited on page 34.)

[HK73] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM J. Comput.*, 2(4): 225–231, 1973. July 31, 2013. http://dx.doi.org/10.1137/0202019
(Cited on pages 5 and 34.)

[HE05] Bradford Hovinen and Wayne Eberly. A Reliable Block Lanczos Algorithm over Small Finite Fields. In *Proc. ISSAC 2005*, pages 177–184. ACM, 2005. ISBN 1-59593-095-7. July 22, 2013. http://dx.doi.org/10.1145/1073884.1073910
(Cited on page 101.)

[JvEB86] Christiaan T. M. Jacobs and Peter van Emde Boas. Two Results on Tables. *Inf. Process. Lett.*, 22(1):43–48, 1986. ISSN 0020-0190. August 6, 2013. http://dx.doi.org/10.1016/0020-0190(86)90041-4
(Cited on page 110.)

[JŁR00] Svante Janson, Tomas Łuczak, and Andrzej Ruciński. *Random Graphs*. Wiley-Interscience, 2000. ISBN 978-0-471-17541-4. August 24, 2013. http://gso.gbv.de/DB=2.1/PPNSET?PPN=308697065
(Cited on pages 126 and 127.)

[JP08] Morten Skaarup Jensen and Rasmus Pagh. Optimality in External Memory Hashing. *Algorithmica*, 52(3):403–411, 2008. ISSN 1432-0541. January 13, 2015. http://dx.doi.org/10.1007/s00453-007-9155-x
(Cited on page 39.)

[Jor79] Charles Jordan. *Calculus of Finite Differences*. Chelsea Publishing Company, New York, USA, 3rd edition, 1979. ISBN 0-8284-0033-4. August 8, 2013. http://gso.gbv.de/DB=2.1/PPNSET?PPN=267909322
(Cited on page 204.)

[Kho13] Megha Khosla. Balls into Bins Made Faster. In *Proc. 21st ESA*, volume 8125 of *LNCS*, pages 601–612. Springer, 2013. ISBN 978-3-642-40449-8. September 15, 2014. http://dx.doi.org/10.1007/978-3-642-40450-4_51
(Cited on pages 4, 23, and 172.)

[Kim06] Jeong Han Kim. Poisson cloning model for random graphs. In *Proc. ICM Madrid 2006 Vol. III*, pages 873–898. EMS Ph, 2006. July 25, 2013. http://www.mathunion.org/ICM/ICM2006.3/
(Cited on pages 16, 104, 122, 124, 145, and 180.)

[Kim08] Jeong Han Kim. Poisson Cloning Model for Random Graphs. *CoRR*, abs/0805.4133v1, 2008. August 21, 2013. http://arxiv.org/abs/0805.4133v1
(Cited on pages 122, 124, and 145.)

[KM07] Adam Kirsch and Michael Mitzenmacher. Using a Queue to De-amortize Cuckoo Hashing in Hardware. In *Proc. 45th Communication, Control, and Computing (Allerton)*, pages 751–758. Curran Associates, 2007. ISBN 978-1-60560-086-4. July 23, 2013. http://www.eecs.harvard.edu/~michaelm/postscripts/aller2007.pdf
(Cited on page 38.)

[KM08] Adam Kirsch and Michael Mitzenmacher. On the Performance of Multiple Choice Hash Tables with Moves on Deletes and Inserts. In *Proc. 46th Communication, Control, and Computing (Allerton)*, pages 1284–1290. IEEE, 2008. ISBN 978-1-4244-2925-7. July 23, 2013. http://dx.doi.org/10.1109/ALLERTON.2008.4797708
(Cited on page 36.)

[KMW08] Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More Robust Hashing: Cuckoo Hashing with a Stash. In *Proc. 16th ESA*, volume 5193 of *LNCS*, pages 611–622. Springer, 2008. ISBN 978-3-540-87743-1. July 23, 2013. http://dx.doi.org/10.1007/978-3-540-87744-8_51
(Cited on page 38.)

[KMW09] Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More Robust Hashing: Cuckoo Hashing with a Stash. *SIAM J. Comput.*, 39(4): 1543–1561, 2009. July 31, 2013. http://dx.doi.org/10.1137/080728743
(Cited on page 38.)

[KK10] Zoltán Király and Péter Kovács. An Experimental Study of Minimum Cost Flow Algorithms. In *Proc. 8th ICAI*, volume 2, pages 227–235. Eger, Hungary, 2010. July 22, 2013. http://icai.ektf.hu/pdf/ICAI2010-vol2-pp227-235.pdf
(Cited on page 74.)

[Knu63] Donald Ervin Knuth. Notes on "open" addressing. unpublished, 1963.

August 14, 2012. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.56.4899
(Cited on page 188.)

[Knu97] Donald Ervin Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, 3rd edition, 1997. ISBN 0-201-89683-4. August 8, 2013. http://dl.acm.org/citation.cfm?id=260999
(Cited on page 203.)

[Knu98] Donald Ervin Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, 2nd edition, 1998. ISBN 0-201-89685-0. August 8, 2013. http://dl.acm.org/citation.cfm?id=280635
(Cited on pages 2, 35, 38, 39, and 188.)

[Koz91] Dexter C. Kozen. *The Design and Analysis of Algorithms*. Springer, 1991. ISBN 0-387-97687-6. August 7, 2013. http://dx.doi.org/10.1007/978-1-4612-4400-4
(Cited on page 43.)

[Kut10] Reinhard Kutzelnigg. A further analysis of Cuckoo Hashing with a Stash and Random Graphs of Excess r. *Discrete Mathematics & Theoretical Computer Science*, 12(3):81–102, 2010. ISSN 1365-8050. December 16, 2014. http://www.dmtcs.org/dmtcs-ojs/index.php/dmtcs/article/view/1345
(Cited on page 38.)

[Lar88] Per-Åke Larson. Linear Hashing with Separators — A Dynamic Hashing Scheme Achieving One-Access Retrieval. *ACM Trans. Database Syst.*, 13(3):366–388, 1988. ISSN 0362-5915. January 13, 2015. http://doi.acm.org/10.1145/44498.44500
(Cited on page 39.)

[LP09] Eric Lehman and Rina Panigrahy. 3.5-Way Cuckoo Hashing for the Price of 2-and-a-Bit. In *Proc. 17th ESA*, volume 5757 of *LNCS*, pages 671–681. Springer, 2009. ISBN 978-3-642-04127-3. July 23, 2013. http://dx.doi.org/10.1007/978-3-642-04128-0_60
(Cited on pages 33 and 189.)

[Lel12a] Marc Lelarge. A New Approach to the Orientation of Random Hypergraphs. In *Proc. 23rd SODA*, pages 251–264. SIAM, 2012. January 10, 2014. http://dl.acm.org/citation.cfm?id=2095139
(Cited on pages 4, 24, 47, 189, and 215.)

[Lel12b] Marc Lelarge. A new approach to the orientation of random hypergraphs. *CoRR*, abs/1201.5335, 2012. March 11, 2013. http://arxiv.org/abs/1201.5335
(Cited on pages 4, 24, and 47.)

[LEM11] LEMON Graph Library, version 1.2.3. online, 2011. launched by the Egerváry Research Group on Combinatorial Optimization, March 25, 2013. http://lemon.cs.elte.hu/trac/lemon/
(Cited on page 74.)

[LP12] Po-Shen Loh and Rasmus Pagh. Thresholds for Extreme Orientability. *CoRR*, abs/1202.1111, 2012. March 18, 2013. http://arxiv.org/abs/1202.1111
(Cited on pages 24 and 46.)

[Lub02] Michael Luby. LT Codes. In *Proc. 43rd FOCS*, pages 271–280. IEEE Computer Society, 2002. ISBN 0-7695-1822-2. ISSN 0272-5428. July 25, 2013. http://dx.doi.org/10.1109/SFCS.2002.1181950
(Cited on pages 6 and 104.)

[LMSS01] Michael Luby, Michael Mitzenmacher, Mohammad Amin Shokrollahi, and Daniel A. Spielman. Efficient Erasure Correcting Codes. *IEEE Transactions on Information Theory*, 47(2):569–584, 2001. ISSN 0018-9448. July 29, 2013. http://dx.doi.org/10.1109/18.910575
(Cited on pages 6, 102, and 104.)

[LMS+97] Michael Luby, Michael Mitzenmacher, Mohammad Amin Shokrollahi, Daniel A. Spielman, and Volker Stemann. Practical Loss-Resilient Codes. In *Proc. 29th STOC*, pages 150–159. ACM, 1997. ISBN 0-89791-888-6. July 25, 2013. http://dx.doi.org/10.1145/258533.258573
(Cited on page 104.)

[Mai83] Harry G. Mairson. The Program Complexity of Searching a Table. In *Proc. 24th FOCS*, pages 40–47. IEEE Computer Society, 1983. ISBN 0-8186-0508-1. January 18, 2015. http://dx.doi.org/10.1109/SFCS.1983.76
(Cited on page 39.)

[Mai92] Harry G. Mairson. The Effect of Table Expansion on the Program Complexity of Perfect Hash Funtions. *BIT*, 32(3):430–440, 1992. ISSN 0006-3835. January 18, 2015. http://dx.doi.org/10.1007/BF02074879
(Cited on page 39.)

[MWHC96] Bohdan S. Majewski, Nicholas C. Wormald, George Havas, and Zbigniew J. Czech. A Family of Perfect Hashing Methods. *Comput. J.*,

39(6):547–554, 1996. July 22, 2013.
http://dx.doi.org/10.1093/comjnl/39.6.547
(Cited on pages 5, 6, 95, 100, 106, 113, and 115.)

[Map09] Maplesoft, Waterloo Maple Inc. *Maple User Manual*, Maple 13, 2009. August 7, 2012. http://www.maplesoft.com/products/maple/history/documentation.aspx
(Cited on page 212.)

[May02] Petar Maymounkov. Online codes (Extended Abstract). Technical Report TR2002-833, Courant Institute, 2002. July 22, 2013.
http://cs.nyu.edu/web/Research/TechReports/reports.html
(Cited on page 104.)

[Meh82] Kurt Mehlhorn. On the program size of perfect and universal hash functions (extended abstract). In *Proc. 23rd FOCS*, pages 170–175. IEEE Computer Society, 1982. July 22, 2013.
http://dx.doi.org/10.1145/96749.98233
(Cited on pages 109, 110, and 116.)

[Meh84] Kurt Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*. EATCS Monographs on Theoretical Computer Science. Springer, 1984. ISBN 978-3-540-13302-5. July 3, 2013.
http://www.mpi-inf.mpg.de/~mehlhorn/DatAlgbooks.html
(Cited on pages 110, 116, and 117.)

[Mit09] Michael Mitzenmacher. Some Open Questions Related to Cuckoo Hashing. In *Proc. 17th ESA*, volume 5757 of *LNCS*, pages 1–10. Springer, 2009. ISBN 978-3-642-04127-3. July 23, 2013.
http://dx.doi.org/10.1007/978-3-642-04128-0_1
(Cited on pages 2 and 36.)

[MT12] Michael Mitzenmacher and Justin Thaler. Peeling Arguments and Double Hashing. In *Proc. 50th Communication, Control, and Computing (Allerton)*, pages 1118–1125. IEEE, 2012. ISBN 978-1-4673-4537-8. July 23, 2013. http://dx.doi.org/10.1109/Allerton.2012.6483344
(Cited on page 189.)

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005. ISBN 0-521-83540-2. August 7, 2013.
http://dl.acm.org/citation.cfm?id=1076315
(Cited on pages 138 and 211.)

[MV08] Michael Mitzenmacher and Salil P. Vadhan. Why Simple Hash Functions Work: Exploiting the Entropy in a Data Stream. In *Proc. 19th SODA*, pages 746–755. SIAM, 2008. July 23, 2013.
http://dl.acm.org/citation.cfm?id=1347082.1347164
(Cited on pages 186 and 187.)

[MV12] Michael Mitzenmacher and George Varghese. Biff (Bloom Filter) Codes: Fast Error Correction for Large Data Sets. In *Proc. ISIT 2012*, pages 483–487. IEEE, 2012. ISBN 978-1-4673-2580-6. July 25, 2013.
http://dx.doi.org/10.1109/ISIT.2012.6284236
(Cited on page 189.)

[MV99] Michael Mitzenmacher and Berhold Vöcking. The Asymptotics of Selecting the Shortest of Two, Improved. Technical Report TR-08-99, Computer Science Group, 1999. July 25, 2013.
ftp://ftp.deas.harvard.edu/techreports/tr-1999.html
(Cited on page 37.)

[MV02] Michael Mitzenmacher and Berhold Vöcking. Selecting the Shortest of Two Queues, Improved. In Yu. M. Suhov, editor, *Analytic Methods in Applied Probability. In Memory of Fridrikh Karpelevich*, volume 207 of *American Mathematical Society Translations: Series 2*, pages 165–175. AMS, 2002. ISBN 0-8218-3306-5. November 6, 2014.
https://zbmath.org/?q=an:1027.60096
(Cited on page 37.)

[Mit91] Michael David Mitzenmacher. *The Power of Two Choices in Randomized Load Balancing*. PhD thesis, Harvard University, 1991. July 22, 2013. http://www.eecs.harvard.edu/~michaelm/postscripts/mythesis.pdf
(Cited on page 36.)

[Mol04] Michael Molloy. The pure literal rule threshold and cores in random hypergraphs. In *Proc. 15th SODA*, pages 672–681. SIAM, 2004. ISBN 0-89871-558-X. July 25, 2013.
http://dl.acm.org/citation.cfm?id=982792.982896
(Cited on pages 6, 16, 103, 104, 130, 131, 132, 135, 138, 141, 145, and 180.)

[ÖP03] Anna Östlin and Rasmus Pagh. Uniform Hashing in Constant Time and Linear Space. In *Proc. 35th STOC*, pages 622–628. ACM, 2003. ISBN 1-58113-674-9. July 25, 2013.
http://dx.doi.org/10.1145/780542.780633
(Cited on pages 184 and 189.)

[PP08] Anna Pagh and Rasmus Pagh. Uniform Hashing in Constant Time and Optimal Space. *SIAM J. Comput.*, 38(1):85–96, 2008. July 29, 2013. http://dx.doi.org/10.1137/060658400
(Cited on pages 7, 184, 185, 189, and 198.)

[PPR07] Anna Pagh, Rasmus Pagh, and Milan Ružić. Linear Probing with Constant Independence. In *Proc. 39th STOC*, pages 318–327. ACM, 2007. ISBN 978-1-59593-631-8. July 24, 2013. http://dx.doi.org/10.1145/1250790.1250839
(Cited on page 188.)

[PPR09] Anna Pagh, Rasmus Pagh, and Milan Ružić. Linear Probing with Constant Independence. *SIAM J. Comput.*, 39(3):1107–1120, 2009. July 24, 2013. http://dx.doi.org/10.1137/070702278
(Cited on page 188.)

[Pag99] Rasmus Pagh. Hash and Displace: Efficient Evaluation of Minimal Perfect Hash Functions. In *Proc. 6th WADS*, volume 1663 of *LNCS*, pages 49–54. Springer, 1999. ISBN 3-540-66279-0. July 23, 2013. http://dx.doi.org/10.1007/3-540-48447-7_5
(Cited on pages 111, 112, and 114.)

[Pag01a] Rasmus Pagh. Low Redundancy in Static Dictionaries with Constant Query Time. *SIAM J. Comput.*, 31(2):353–363, 2001. July 29, 2013. http://dx.doi.org/10.1137/S0097539700369909
(Cited on page 184.)

[Pag01b] Rasmus Pagh. On the Cell Probe Complexity of Membership and Perfect Hashing. In *Proc. 33rd STOC*, pages 425–432. ACM, 2001. ISBN 1-58113-349-9. July 24, 2013. http://dx.doi.org/10.1145/380752.380836
(Cited on pages 1, 2, 4, 20, 22, 37, and 188.)

[Pag03] Rasmus Pagh. Basic External Memory Data Structures. In *Algorithms for Memory Hierarchies, Advanced Lectures*, volume 2625 of *LNCS*, pages 14–35. Springer, 2003. ISBN 3-540-00883-7. January 13, 2015. http://dx.doi.org/10.1007/3-540-36574-5_2
(Cited on page 39.)

[Pag09] Rasmus Pagh. Dispersing Hash Functions. *Random Struct. Algorithms*, 35(1):70–82, 2009. July 23, 2013. http://dx.doi.org/10.1002/rsa.20257
(Cited on page 192.)

[PR01] Rasmus Pagh and Flemming Friche Rodler. Cuckoo Hashing. In *Proc. 9th ESA*, volume 2161 of *LNCS*, pages 121–133. Springer, 2001. ISBN 3-540-42493-8. July 25, 2013.
http://dx.doi.org/10.1007/3-540-44676-1_10
(Cited on pages 2, 36, and 37.)

[PR04] Rasmus Pagh and Flemming Friche Rodler. Cuckoo Hashing. *J. Algorithms*, 51(2):122–144, 2004. ISSN 0196-6774. August 6, 2013.
http://dx.doi.org/10.1016/j.jalgor.2003.12.002
(Cited on pages 2, 23, 37, and 188.)

[PWYZ14] Rasmus Pagh, Zhewei Wei, Ke Yi, and Qin Zhang. Cache-Oblivious Hashing. *Algorithmica*, 69(4):864–883, 2014. ISSN 0178-4617. January 17, 2015. http://dx.doi.org/10.1007/s00453-013-9763-6
(Cited on page 39.)

[Pan12] Konstantinos Panagiotou. personal communication (communicated by Martin Dietzfelbinger), 2012
(Cited on page 105.)

[Pan05] Rina Panigrahy. Efficient Hashing with Lookups in two Memory Accesses. In *Proc. 16th SODA*, pages 830–839. SIAM, 2005. ISBN 0-89871-585-7. July 23, 2013.
http://dl.acm.org/citation.cfm?id=1070432.1070549
(Cited on page 32.)

[PS12] Boris Pittel and Gregory B. Sorkin. The Satisfiability Threshold for k-XORSAT. *CoRR*, abs/1212.1905, 2012. April 8, 2013.
http://arxiv.org/abs/1212.1905
(Cited on pages 100 and 113.)

[PM89] Patricio V. Poblete and J. Ian Munro. Last-Come-First-Served Hashing. *J. Algorithms*, 10(2):228–248, 1989. ISSN 0196-6774. July 29, 2013.
http://dx.doi.org/10.1016/0196-6774(89)90014-X
(Cited on page 37.)

[Por08] Ely Porat. An Optimal Bloom Filter Replacement Based on Matrix Solving. *CoRR*, abs/0804.1845, 2008. July 31, 2013.
http://arxiv.org/abs/0804.1845
(Cited on page 113.)

[Por09] Ely Porat. An Optimal Bloom Filter Replacement Based on Matrix Solving. In *Proc. 4th CSR*, volume 5675 of *LNCS*, pages 263–273.

Springer, 2009. ISBN 978-3-642-03350-6. July 26, 2013. http://dx.doi.org/10.1007/978-3-642-03351-3_25 (Cited on pages 113 and 115.)

[PS12] Ely Porat and Bar Shalem. A Cuckoo Hashing Variant with Improved Memory Utilization and Insertion Time. In *Proc. 22nd DCC*, pages 347–356. IEEE Computer Society, 2012. ISBN 978-0-7695-4656-8. July 25, 2013. http://dx.doi.org/10.1109/DCC.2012.41 (Cited on pages 32, 33, and 189.)

[PS98] Helmut Prodinger and Wojciech Szpankowski. Preface. *Algorithmica*, 22 (4):363–365, 1998. ISSN 0178-4617. August 6, 2013. http://dx.doi.org/10.1007/PL00009229 (Cited on page 188.)

[Pă08] Mihai Pătraşcu. Succincter. In *Proc. 49th FOCS*, pages 305–313. IEEE Computer Society, 2008. July 22, 2013. http://doi.ieeecomputersociety.org/10.1109/FOCS.2008.83 (Cited on page 173.)

[PT10] Mihai Pătraşcu and Mikkel Thorup. On the k-Independence Required by Linear Probing and Minwise Independence. In *Proc. 37th ICALP (1)*, volume 6198 of *LNCS*, pages 715–726. Springer, 2010. ISBN 978-3-642-14164-5. July 25, 2013. http://dx.doi.org/10.1007/978-3-642-14165-2_60 (Cited on page 188.)

[PT11] Mihai Pătraşcu and Mikkel Thorup. The Power of Simple Tabulation Hashing. In *Proc. 43rd STOC*, pages 1–10. ACM, 2011. ISBN 978-1-4503-0691-1. July 25, 2013. http://doi.acm.org/10.1145/1993636.1993638 (Cited on page 188.)

[PT12] Mihai Pătraşcu and Mikkel Thorup. The Power of Simple Tabulation Hashing. *J. ACM*, 59(3):14:1–14:50, 2012. ISSN 0004-5411. July 25, 2013. http://dx.doi.org/10.1145/2220357.2220361 (Cited on pages 188, 189, and 216.)

[PT13] Mihai Pătraşcu and Mikkel Thorup. Twisted Tabulation Hashing. In *Proc. 24th SODA*, pages 209–228. SIAM, 2013. January 6, 2014. http://knowledgecenter.siam.org/0236-000005 (Cited on page 189.)

[QM98] Hongbin Qi and Charles U. Martel. Design and Analysis of Hashing Algorithms with Cache Effects. Technical report, UC Davis, 1998.

January 17, 2015. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.55.664
(Cited on page 39.)

[RS98] Martin Raab and Angelika Steger. "Balls into Bins" - A Simple and Tight Analysis. In *Proc. 2nd RANDOM*, volume 1518 of *LNCS*, pages 159–170. Springer, 1998. ISBN 3-540-65142-X. July 23, 2013. http://dx.doi.org/10.1007/3-540-49543-6_13
(Cited on page 36.)

[Rin07] Michael Rink. Untersuchungen zu neueren Bloom-Filter-Varianten und d-left-Hashing. Diplomarbeit, Technische Universität Ilmenau, 2007. August 2, 2013. http://gso.gbv.de/DB=2.1/PPNSET?PPN=550169911
(Cited on page 116.)

[Rin12] Michael Rink. On Thresholds for the Appearance of 2-cores in Mixed Hypergraphs. *CoRR*, abs/1204.2131, 2012. October 10, 2012. http://arxiv.org/abs/1204.2131
(Cited on pages xiii, 146, and 170.)

[Rin13] Michael Rink. Mixed Hypergraphs for Linear-Time Construction of Denser Hashing-Based Data Structures. In *Proc. 39th SOFSEM*, volume 7741 of *LNCS*, pages 356–368. Springer, 2013. ISBN 978-3-642-35842-5. July 25, 2013. http://dx.doi.org/10.1007/978-3-642-35843-2_31
(Cited on pages xiii, 146, and 170.)

[Riv78] Ronald L. Rivest. Optimal Arrangement of Keys in a Hash Table. *J. ACM*, 25(2):200–209, 1978. ISSN 0004-5411. July 31, 2013. http://doi.acm.org/10.1145/322063.322065
(Cited on pages 20 and 37.)

[Ros07] Kenneth A. Ross. Efficient Hash Probes on Modern Processors. In *Proc. 23rd ICDE*, pages 1297–1301. IEEE, 2007. July 22, 2013. http://doi.ieeecomputersociety.org/10.1109/ICDE.2007.368997
(Cited on page 2.)

[Sag85] Thomas J. Sager. A Polynomial Time Generator for Minimal Perfect Hash Functions. *Commun. ACM*, 28(5):523–532, 1985. ISSN 0001-0782. July 22, 2013. http://doi.acm.org/10.1145/3532.3538
(Cited on page 114.)

[San04] Peter Sanders. Algorithms for Scalable Storage Servers. In *Proc. 30th SOFSEM*, volume 2932 of *LNCS*, pages 82–101. Springer, 2004. ISBN

3-540-20779-1. July 22, 2013.
http://dx.doi.org/10.1007/978-3-540-24618-3_8
(Cited on pages 4, 23, and 24.)

[SEK00] Peter Sanders, Sebastian Egner, and Jan H. M. Korst. Fast Concurrent Access to Parallel Disks. In *Proc. 11th SODA*, pages 849–858. ACM/SIAM, 2000. ISBN 0-89871-453-2. July 29, 2013.
http://dl.acm.org/citation.cfm?id=338219.338649
(Cited on page 37.)

[SEK03] Peter Sanders, Sebastian Egner, and Jan H. M. Korst. Fast Concurrent Access to Parallel Disks. *Algorithmica*, 35(1):21–55, 2003. July 29, 2013.
http://dx.doi.org/10.1007/s00453-002-0987-0
(Cited on page 37.)

[Sch09] Ulf Schellbach. *On Risks of Using a High Performance Hashing Scheme With Common Universal Classes*. Dissertation, Technische Universität Ilmenau, 2009. July 22, 2013.
http://gso.gbv.de/DB=2.1/PPNSET?PPN=604739664
(Cited on page 188.)

[SS88] Jeanette P. Schmidt and Alan Siegel. The spatial complexity of oblivious k-probe hash functions. Technical Report Ultracomputer Note #142, Courant Institute, 1988. July 25, 2013.
ftp://cs.nyu.edu/pub/local/ultra/ucn/101-150/ucn142.ps.Z
(Cited on page 110.)

[SS90] Jeanette P. Schmidt and Alan Siegel. The Spatial Complexity of Oblivious k-Probe Hash Functions. *SIAM J. Comput.*, 19(5):775–786, 1990. July 24, 2013. http://dx.doi.org/10.1137/0219054
(Cited on page 110.)

[SH94] Steven S. Seiden and Daniel S. Hirschberg. Finding succinct ordered minimal perfect hash functions. *Inf. Process. Lett.*, 51(6):283–288, 1994. ISSN 0020-0190. July 23, 2013.
http://dx.doi.org/10.1016/0020-0190(94)00108-1
(Cited on pages 113 and 115.)

[Sho06] Amin Shokrollahi. Raptor Codes. *IEEE Transactions on Information Theory*, 52(6):2551–2567, 2006. ISSN 0018-9448. July 25, 2013.
http://dx.doi.org/10.1109/TIT.2006.874390
(Cited on page 104.)

[Sie89] Alan Siegel. On Universal Classes of Fast High Performance Hash Functions, Their Time-Space Tradeoff, and Their Applications (Extended

Abstract). In *Proc. 30th FOCS*, pages 20–25. IEEE Computer Society, 1989. ISBN 0-8186-1982-1. July 25, 2013.
http://dx.doi.org/10.1109/SFCS.1989.63450
(Cited on page 188.)

[Sie95] Alan Siegel. On universal classes of extremely random constant-time hash functions and their time-space tradeoff. Technical Report TR1995-684, Courant Institute, 1995. July 22, 2013.
http://cs.nyu.edu/web/Research/TechReports/reports.html
(Cited on page 198.)

[Sie04] Alan Siegel. On Universal Classes of Extremely Random Constant-Time Hash Functions. *SIAM J. Comput.*, 33(3):505–543, 2004. July 29, 2013.
http://dx.doi.org/10.1137/S0097539701386216
(Cited on pages 97, 184, 188, and 192.)

[Sie88] Wacław Sierpiński. *Elementary Theory of Numbers*. Elsevier, 2nd edition, 1988. ISBN 978-0-444-86662-2. August 24, 2013.
http://gso.gbv.de/DB=2.1/PPNSET?PPN=232354383
(Cited on page 192.)

[TY79] Robert Endre Tarjan and Andrew Chi-Chih Yao. Storing a Sparse Table. *Commun. ACM*, 22(11):606–611, 1979. ISSN 0001-0782. July 23, 2013.
http://dx.doi.org/10.1145/359168.359175
(Cited on page 111.)

[TZ04] Mikkel Thorup and Yin Zhang. Tabulation Based 4-Universal Hashing with Applications to Second Moment Estimation. In *Proc. 15th SODA*, pages 615–624. SIAM, 2004. ISBN 0-89871-558-X. January 6, 2014.
http://dl.acm.org/citation.cfm?id=982792
(Cited on page 189.)

[TZ10] Mikkel Thorup and Yin Zhang. Tabulation Based 5-Universal Hashing and Linear Probing. In *Proc. 12th ALENEX*, pages 62–76. SIAM, 2010. ISBN 978-0-898719-31-4. January 6, 2014. http://www.siam.org/proceedings/alenex/2010/alx10_007_thorupm.pdf
(Cited on page 189.)

[TZ12] Mikkel Thorup and Yin Zhang. Tabulation-Based 5-Independent Hashing with Applications to Linear Probing and Second Moment Estimation. *SIAM J. Comput.*, 41(2):293–331, 2012. January 6, 2014.
http://dx.doi.org/10.1137/100800774
(Cited on page 189.)

[Vö99] Berthold Vöcking. How Asymmetry Helps Load Balancing. In *Proc. 40th FOCS*, pages 131–141. IEEE Computer Society, 1999. ISBN 0-7695-0409-4. July 23, 2013. http://dx.doi.org/10.1109/SFFCS.1999.814585
(Cited on page 36.)

[Vö03] Berthold Vöcking. How Asymmetry Helps Load Balancing. *J. ACM*, 50 (4):568–589, 2003. ISSN 0004-5411. July 31, 2013. http://doi.acm.org/10.1145/792538.792546
(Cited on pages 36 and 37.)

[WC79] Mark N. Wegman and Larry Carter. New Classes and Applications of Hash Functions. In *Proc. 20th FOCS*, pages 175–182. IEEE Computer Society, 1979. ISSN 0272-5428. July 25, 2013. http://doi.ieeecomputersociety.org/10.1109/SFCS.1979.26
(Cited on page 187.)

[WC81] Mark N. Wegman and Larry Carter. New Hash Functions and Their Use in Authentication and Set Equality. *J. Comput. Syst. Sci.*, 22(3): 265–279, 1981. ISSN 0022-0000. August 6, 2013. http://dx.doi.org/10.1016/0022-0000(81)90033-7
(Cited on pages 187 and 191.)

[Wei04] Christoph Weidling. *Platzeffiziente Hashverfahren mit garantierter konstanter Zugriffszeit*. Dissertation, Technische Universität Ilmenau, 2004. July 22, 2013. http://www.weidlings.de/christoph/papers/dissertation.pdf
(Cited on pages 33 and 114.)

[Wie86] Douglas H. Wiedemann. Solving Sparse Linear Equations Over Finite Fields. *IEEE Transactions on Information Theory*, 32(1):54–62, 1986. ISSN 0018-9448. July 25, 2013. http://dx.doi.org/10.1109/TIT.1986.1057137
(Cited on pages 101 and 113.)

[Wik12] Wikipedia. DBpedia 3.8: Dataset Titles and Dataset Categories (Labels), 2012. June 4, 2013. http://wiki.dbpedia.org/Downloads38
(Cited on page 174.)

[WKL+12] Thomas Williams, Colin Kelley, Russell Lang, Dave Kotz, John Campbell, Gershon Elber, Alexander Woo, and many others. gnuplot, version 4.6. online, 2012. May 1, 2013. http://www.gnuplot.info/
(Cited on pages 50 and 177.)

[Woe06] Philipp Woelfel. Maintaining External Memory Efficient Hash Tables (Extended Abstract). In *Proc. 9th APPROX - 10th RANDOM*, volume 4110 of *LNCS*, pages 508–519. Springer, 2006. ISBN 3-540-38044-2. July 21, 2013. http://dx.doi.org/10.1007/11830924_46
(Cited on page 112.)

Erklärung

Ich versichere, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Weitere Personen waren an der inhaltlich-materiellen Erstellung der vorliegenden Arbeit nicht beteiligt. Insbesondere habe ich hierfür nicht die entgeltliche Hilfe von Vermittlungs- bzw. Beratungsdiensten (Promotionsberater oder anderer Personen) in Anspruch genommen. Niemand hat von mir unmittelbar oder mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form einer Prüfungsbehörde vorgelegt.

Ich bin darauf hingewiesen worden, dass die Unrichtigkeit der vorstehenden Erklärung als Täuschungsversuch bewertet wird und gemäß § 7 Abs. 10 der Promotionsordnung den Abbruch des Promotionsverfahrens zur Folge hat.

Ilmenau, den 7. April 2015

Michael Rink