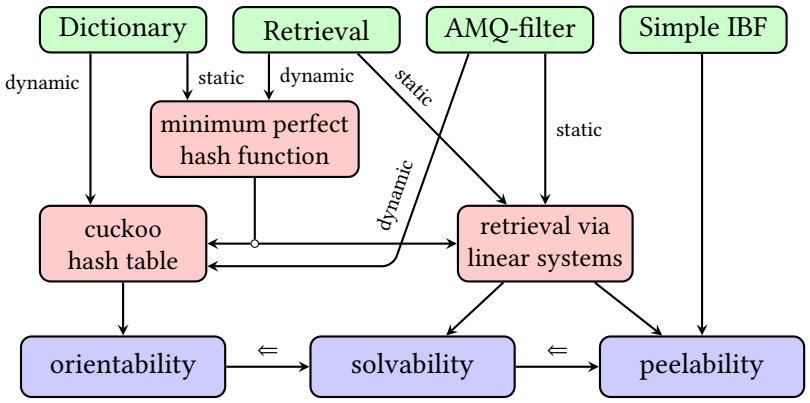
Random Hypergraphs
for Hashing-Based Data Structures

Dissertation zur Erlangung des akademischen Grades
Doctor rerum naturalium (Dr. rer. nat.)

vorgelegt der Fakultät für Informatik und Automatisierung
der Technischen Universität Ilmenau

von

Stefan Walzer



Betreuer: Univ.-Prof. Dr. Martin Dietzfelbinger, TU Ilmenau
2. Gutachter: Univ.-Prof. Dr. Konstantinos Panagiotou, LMU München
3. Gutachter: Prof. Michael Mitzenmacher, PhD., Harvard University

Tag der Einreichung: 1. Mai 2020
Tag der Verteidigung: 19. Oktober 2020

TECHNISCHE UNIVERSITÄT
ILMENAU

urn:nbn:de:gbv:ilm1-2020000498

Zusammenfassung

Diese Arbeit behandelt Wörterbücher und verwandte Datenstrukturen, die darauf aufbauen, mehrere zufällige Möglichkeiten zur Speicherung jedes Schlüssels vorzusehen.

Man stelle sich vor, Information über eine Menge S von $m = |S|$ Schlüsseln soll in n Speicherplätzen abgelegt werden, die durch $[n] = \{1, \dots, n\}$ indiziert sind. Jeder Schlüssel $x \in S$ bekommt eine kleine Menge $e(x) \subseteq [n]$ von Speicherplätzen durch eine zufällige Hashfunktion unabhängig von anderen Schlüsseln zugewiesen. Die Information über x darf nun ausschließlich in den Plätzen aus $e(x)$ untergebracht werden. Es kann hierbei passieren, dass zu viele Schlüssel um dieselben Speicherplätze konkurrieren, insbesondere bei hoher *Auslastung* $c = m/n$. Eine erfolgreiche Speicherung der Gesamtinformation ist dann eventuell unmöglich. Für die meisten Verteilungen von $e(x)$ lässt sich Erfolg oder Misserfolg allerdings sehr zuverlässig vorhersagen, da für Auslastung c unterhalb eines gewissen *Auslastungsschwellwertes* c^* die Erfolgswahrscheinlichkeit nahezu 1 ist und für c jenseits dieses Auslastungsschwellwertes nahezu 0 ist.
Hauptsächlich werden wir zwei Arten von Datenstrukturen betrachten:

- Eine *Kuckucks-Hashtabelle* ist eine Wörterbuchdatenstruktur, bei der jeder Schlüssel $x \in S$ zusammen mit einem assoziierten Wert $f(x)$ in einem der Speicherplätze mit Index aus $e(x)$ gespeichert wird. Die Verteilung von $e(x)$ wird hierbei vom *Hashing-Schema* festgelegt. Wir analysieren drei bekannte Hashing-Schemata und bestimmen erstmals deren exakte Auslastungsschwellwerte im obigen Sinne. Die Schemata sind *unausgerichtete Blöcke*, *Doppel-Hashing* sowie ein Schema für dynamisch wachsenden Schlüsselmengen.
- Auch eine *Retrieval-Datenstruktur* speichert einen Wert $f(x)$ für alle $x \in S$. Diesmal sollen die Werte in den Speicherplätzen aus $e(x)$ eine lineare Gleichung erfüllen, die den Wert $f(x)$ charakterisiert. Die entstehende Datenstruktur ist extrem platzsparend, aber ungewöhnlich: Sie ist ungeeignet um Fragen der Form „ist $y \in S$?“ zu beantworten. Bei Anfrage eines Schlüssels y wird ein Ergebnis z zurückgegeben. Falls $y \in S$ ist, so ist $z = f(y)$ garantiert, andernfalls darf z ein beliebiger Wert sein.
  Wir betrachten zwei neue Hashing-Schemata, bei denen die Elemente von $e(x)$ in einem oder in zwei zusammenhängenden Blöcken liegen. So werden gute Zugriffszeiten auf Word-RAMs und eine hohe Cache-Effizienz erzielt.

Eine wichtige Frage ist, ob Datenstrukturen obiger Art in Linearzeit konstruiert werden können. Die Erfolgswahrscheinlichkeit eines naheliegenden Greedy-Algorithmus weist abermals ein Schwellwertverhalten in Bezug auf die Auslastung c auf. Wir identifizieren ein Hashing-Schema, das diesbezüglich einen besonders hohen Schwellwert mit sich bringt.

In der mathematischen Modellierung werden die Speicherpositionen $[n]$ als Knoten und die Mengen $e(x)$ für $x \in S$ als Hyperkanten aufgefasst. Drei Eigenschaften der entstehenden Hypergraphen stellen sich dann als zentral heraus: *Schälbarkeit*, *Lösbarkeit* und *Orientierbarkeit*. Weite Teile dieser Arbeit beschäftigen sich daher mit den Wahrscheinlichkeiten für das Vorliegen dieser Eigenschaften abhängig von Hashing Schema und Auslastung, sowie mit entsprechenden Schwellwerten. Eine Rückübersetzung der Ergebnisse liefert dann Datenstrukturen mit geringen Anfragezeiten, hoher Speichereffizienz und geringen Konstruktionszeiten. Die theoretischen Überlegungen werden dabei durch experimentelle Ergebnisse ergänzt und gestützt.

Abstract

This thesis concerns dictionaries and related data structures that rely on providing several random possibilities for storing each key.

Imagine information on a set S of $m = |S|$ keys should be stored in n memory locations, indexed by $[n] = \{1, \dots, n\}$. Each object $x \in S$ is assigned a small set $e(x) \subseteq [n]$ of locations by a random hash function, independent of other objects. Information on x must then be stored in the locations from $e(x)$ only. It is possible that too many objects compete for the same locations, in particular if the *load* $c = m/n$ is high. Successfully storing all information may then be impossible. For most distributions of $e(x)$, however, success or failure can be predicted very reliably, since the success probability is close to 1 for loads c less than a certain *load threshold* c^* and close to 0 for loads greater than this load threshold.
We mainly consider two types of data structures:

- A *cuckoo hash table* is a dictionary data structure where each key $x \in S$ is stored together with an associated value $f(x)$ in one of the memory locations with an index from $e(x)$. The distribution of $e(x)$ is controlled by the *hashing scheme*. We analyse three known hashing schemes, and determine their exact load thresholds. The schemes are *unaligned blocks*, *double hashing* and a scheme for dynamically growing key sets.
- A *retrieval data structure* also stores a value $f(x)$ for each $x \in S$. This time, the values stored in the memory locations from $e(x)$ must satisfy a linear equation that characterises the value $f(x)$.
  The resulting data structure is extremely compact, but unusual. It cannot answer questions of the form "is $y \in S$?". Given a key y it returns a value z. If $y \in S$, then $z = f(y)$ is guaranteed, otherwise z may be an arbitrary value.
  We consider two new hashing schemes, where the elements of $e(x)$ are contained in one or two contiguous blocks. This yields good access times on a word RAM and high cache efficiency.

An important question is whether these types of data structures can be constructed in linear time. The success probability of a natural linear time greedy algorithm exhibits, once again, threshold behaviour with respect to the load c. We identify a hashing scheme that leads to a particularly high threshold value in this regard.

In the mathematical model, the memory locations $[n]$ correspond to vertices, and the sets $e(x)$ for $x \in S$ correspond to hyperedges. Three properties of the resulting hypergraphs turn out to be important: *peelability*, *solvability* and *orientability*. Therefore, large parts of this thesis examine how hyperedge distribution and load affects the probabilities with which these properties hold and derive corresponding thresholds. Translated back into the world of data structures, we achieve low access times, high memory efficiency and low construction times. We complement and support the theoretical results by experiments.

Acknowledgements

Shortly after my master's, I find myself in a small unfamiliar town, about to see a professor I've never met before for a job interview. A few minutes later, I am sitting in the office of *Martin Dietzfelbinger* – he is explaining small world graphs to me – and I feel at home. It is his nature that drew me to Ilmenau: Kind, respectful, patient. His thinking seems remarkably structured, with every word, gesture or stroke of the pen deliberately crafted to expose an idea more clearly. In the following years, whenever I enter his office, he can spare a friendly smile and some time for his student. His warm-hearted nature holds the research group together. He is a role model, as a scientist and as a person. I cannot conceive of a better advisor and owe him my gratitude.

The following other people deserve my thanks. *Michael Rink* for providing me with the extensive bibliography file from his thesis [Rin15] and all cited papers. *Alexander Koch*, a good friend and frequent coauthor, for proofreading the introduction of this thesis. *Konstantinos Panagiotou* and *Michael Mitzenmacher* for encouraging words and taking the time to review this thesis. *Konstantinos Panagiotou* (again) and *Peter Sanders* for inviting me to Munich and Karlsruhe, respectively, to discuss our work. *Philipp Schlag* for his strong dedication to our joint teaching duties. And finally, thanks to everyone at the institute for theoretical computer science for a pleasant work atmosphere, stimulating discussions over lunch, exciting board game nights and – last but not least – designing my eminently beautiful doctoral hat.

Contributions by Others

This thesis is based on six papers with coauthors listed in Table 1.2. The following remarks are due: Michael Mitzenmacher gave the impulse for [MPW18] and Konstantinos Panagiotou discovered a Theorem by Lutz Warnke, which is a critical ingredient in the proof. Tobias Maier und Peter Sanders are the primary authors of [MSW19], I only contributed the derivation of the thresholds, as found in Chapter 11 of this thesis. A question by Seth Pettie inspired the work of [DW19c]. An email by Djamal Belazzougui provided a key reference that guided the development of [Wal20b] in the right direction.

The works with Martin Dietzfelbinger were written in close collaboration, and it is impossible to disentangle who contributed which ideas. Even in the papers where he is *not* officially a coauthor, he was always part of the discussion and his comments often greatly improved the presentation. This is doubly true for [Wal20b], which is a revision of our joint paper [DW19b].

This thesis is typeset with LaTeX using the Linux Libertine typeface. Many illustrations were created with Ti*k*Z. Some designs were adopted from the LIPIcs LaTeX-template by Dagstuhl Publishing.

Contents

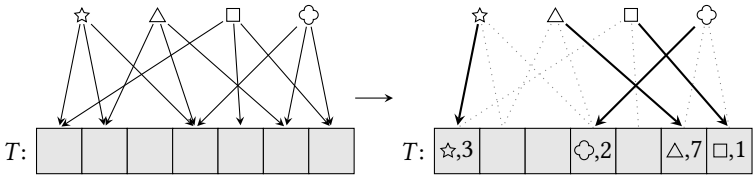
List of Figures

List of Tables

1. Introduction

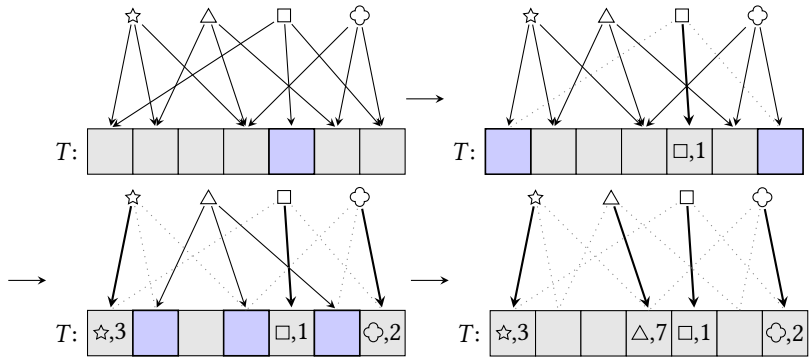
Before getting to the contributions of this thesis in Section 1.1, we review the notions of *peelability*, *solvability* and *orientability*, and how they relate to *cuckoo hash tables* and *retrieval data structures*. Except for the name "solvability" that we introduce, the terminology and general ideas are standard in the literature.

Imagine the mapping $f = \{☆ \mapsto 3, △ \mapsto 7, □ \mapsto 1, ◌ \mapsto 2\}$ should be stored in a data structure using a table T. To facilitate fast information retrieval while maintaining some flexibility, each *key* in $\{☆, △, □, ◌\}$ is randomly assigned several positions in T. A key's information must be stored in exactly one of these positions, and each position can hold at most one key/value pair. In the case shown below, placing all keys is possible, and we say the configuration on the left is **orientable**, with an *orientation* shown on the right.[1]



(1.1)

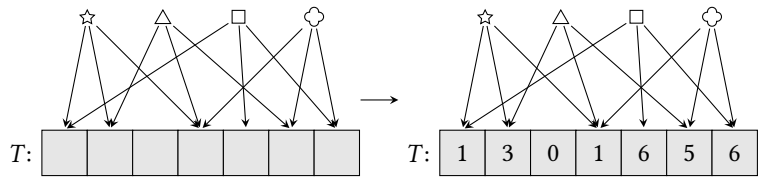
In the given case, a placement can even be found greedily, by identifying positions in T that are an option for only one (remaining) key. At first, only □ can be placed in this way into position 5, then ☆ and ◌ can be placed, and, finally, △ ends up with three positions to itself and can be placed in any one of them. When this greedy procedure works, we say a configuration is **peelable**.



(1.2)

[1] Note that the terminology of this introduction is less general than the terminology used in later chapters. For instance, what is now called an orientation, will later be called a 1-orientation, which is a special case of an ℓ-orientation.

Finally, consider a somewhat peculiar way of storing the information of f:



(1.3)

By taking the bit-wise exclusive or (XOR) of the numbers in the positions assigned to a key, we get the value assigned to the key. For ◌ this would be $1 \oplus 5 \oplus 6 = (001)_2 \oplus (101)_2 \oplus (110)_2 = (010)_2 = 2$ which is $f(◌)$. We say the configuration of keys and associated positions in T is **solvable** because the underlying equations can be satisfied for all keys simultaneously, regardless of the values assigned to the keys by f. We shall later see that *peelability* implies *solvability* which, in turn, implies *orientability*.

Roughly speaking, the goal of this thesis is to shed light on the following question.

Q: *If keys are assigned sets of positions independently at random, under what conditions can we expect the resulting configuration to be peelable, solvable or orientable?*

This question is motivated by hashing-based data structures. To establish the connection and construct working data structures from the ideas above, we have to provide a mechanism that assigns to each key a set of positions in T. This is done via a **hash function**, which is a simple algorithm with pseudo-random behaviour. For a given key (and possibly a *seed*) it computes a *hash value*. This hash value is then interpreted using a hashing scheme (see below) as a set of positions in T. We assume that, in this way, the keys are assigned sets of positions independently at random with identical distribution.[2] Note that repeated execution of the hash function with the same key always yields the same result.

A dictionary is an abstract data type for storing a function f. A **cuckoo hash table** [PR04; Fot+05; DW07; CSW07; FR07; LP09; DMR11; PS12] implements a dictionary and is given by a hash function and a table T populated as shown in **(1.1)**. To *lookup* the key △, the positions of T indicated by the hash function, here $\{2, 4, 6\}$, are probed and $T[6]$ reveals that $f(△) = 7$. A lookup of an absent key ◇ is carried out similarly by evaluating the hash function and probing positions. After failing to find ◇ in these positions, we know that ◇ is not in the domain of f.

A **retrieval data structure** [BKZ05; DP08; ADR09; Por09; DR09; Por09; BPZ13; GOV16; GOV20; EGV20] is a *partial* implementation of the dictionary data type. It allows to evaluate the stored function f, but may yield an arbitrary value if the requested key is not in the domain of f. Typically, a retrieval data structure is given by a hash function and a table T populated as shown in **(1.3)**. For a given key △ in the domain of f, we can evaluate the hash function to obtain positions in T, here $\{2, 4, 6\}$, and by taking the bit-wise exclusive or (XOR) of the stored numbers, here $3 \oplus 1 \oplus 5 = 7$, we obtain $f(△)$. However, for an absent key ◇, we can go through the same motions and obtain a value with no indication that ◇ is not contained in the domain of f. Strictly speaking, a retrieval data structure therefore only provides *conditional* information of the form "*if* △ is contained in the domain of f, *then* $f(△) = 7$". The upside is that keys need not be stored in T, which saves a significant

[2] Whether or not this "full randomness assumption" is problematic is discussed in Section 4.3.

amount of memory, especially if the number of bits required to store a value, here 3, is far smaller than the number of bits required to store a key. In the example, $7 \cdot 3 = 21$ bits suffice to store T.

As a motivation for retrieval data structures, we briefly outline their role in a construction of **approximate membership query (AMQ) filters** [Blo70; BM03; CC08; DP08; FAK13; GL19]. Assume we have a set M, say $M = \{\star, \triangle, \square, \diamondsuit\}$, and care about membership queries, i.e. questions of the form "is $x \in M$?". We choose a range $R = \{0, \dots, 2^r - 1\}$, say with $r = 3$, and a hash function f with range R, often called a *fingerprint function*. We store the assignment $\{x \mapsto f(x) \mid x \in M\}$ in a retrieval data structure as shown in **(1.3)** but *also* keep f as a hash function. For a given key, we now have two ways of computing a value from R, firstly by evaluating f directly, and secondly by consulting the retrieval data structure. For keys from M we get the same result in both cases, e.g. 3 for $\star$. For a key $\diamond \notin M$, we obtain matching results only with probability $1/|R| = 2^{-r}$ since $f(\diamond)$ had no role in the construction of the retrieval data structure. To a question "is $x \in M$?" we can therefore answer "$x \notin M$, surely" on a mismatch and "$x \in M$, probably" on a match. Even though the positive answers are unreliable, such AMQ-filters are useful since they are small and fast. If the reliable negative answers are sufficiently common, they can significantly disburden large background data structures that can answer "is $x \in M$?" perfectly but slowly.

To summarise, orientability from **(1.1)** allows for cuckoo hash tables, and solvability from **(1.3)** allows for retrieval data structures. It is not hard to see that, in both cases, peelability as in **(1.2)** guarantees that a simple greedy algorithm, called a *peeling algorithm*, can be used to populate the table T. We can therefore rephrase our earlier question **Q** from a data structure perspective:

Q′: *Assuming fully random hash functions are available, under what conditions can an attempt at constructing a cuckoo hash table or a retrieval data structure like above be expected to succeed? Under what conditions can a peeling algorithm carry out the construction?*

Let us break this down some more. By "conditions" we mean two things. Firstly, we mean the ratio $c = \frac{m}{n} \in [0, 1]$ between the number m of keys and the size n of the table T. We call c **load** or **density**. Secondly, we mean the **hashing scheme**. It determines how a key's hash value, which is typically a long bit string, is interpreted as a set of positions in T. The hashing scheme therefore dictates the structure of the set $e(x) \subseteq \{1, \dots, n\}$ of table positions assigned to each key x. Some previously studied schemes are:

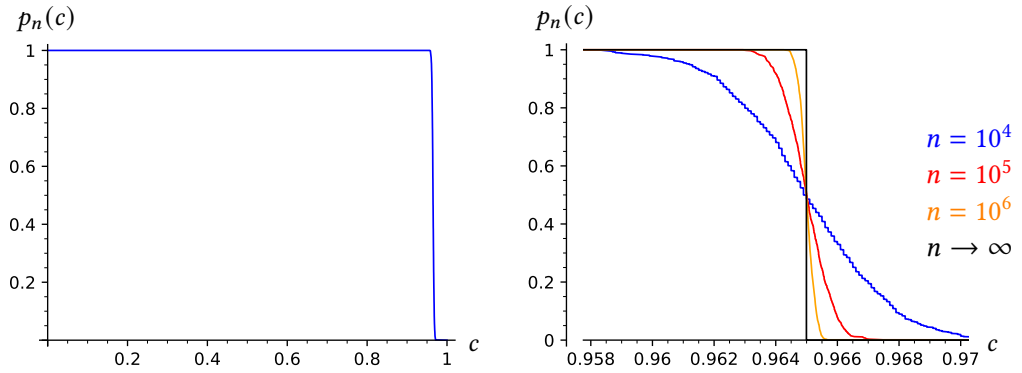
Fully Random. For a given $k \in \mathbb{N}$, The set $e(x)$ contains k positions selected independently and uniformly at random. The example was like this with $k = 3$.

Double Hashing. For a given $k \in \mathbb{N}$, the set $e(x)$ is a random k-term arithmetic progression, i.e. is of the form $\{a, a + b, a + 2b, \dots, a + (k - 1)b\}$, with each term taken modulo n, for random a and b. The motivation is to disburden the hash function, which has to produce only 2 random values instead of k.

(Un-)Aligned Blocks. For given $k, \ell \in \mathbb{N}$, the set $e(x)$ consists of k randomly chosen *blocks* of ℓ consecutive positions each, for a total of $k\ell$ positions. This is motivated by faster access times to contiguous memory positions due to processor caches. In one variant, blocks must be *aligned* to multiples of ℓ. In that case, ℓ must divide the size of T. In the *unaligned* variant, blocks may lie arbitrarily.

The phrase "can it be *expected* to succeed" in **Q′** indicates that we deal with uncertainty. Indeed, due to the randomness in the hash function, it is possible that too many keys compete for the same positions in T and a construction attempt fails (though we may retry with a new hash function or a new seed). But as soon as we consider large data structures with millions of elements, the uncertainty aspect fades into the background since the probabilities for success are often extremely close to 0 or extremely close to 1.

To illustrate this general and well-known effect, consider the probability p for orientability when using the hashing scheme with $k = 2$ unaligned blocks of size $\ell = 2$. In the following picture on the left, we plot (experimental approximations of) $p = p_n(c)$ for varying density $c \in [0, 1]$ and fixed table size $n = 10^4$.



As we would expect, when we increase c and therefore the number $m = cn$ of keys, the probability for orientability decreases. What is *not* obvious is that the transition from $p_n(c) \approx 1$ to $p_n(c) \approx 0$ happens within a tiny interval of values around $c = 0.96$. On the right, we zoom in and also plot the function for $n = 10^5$ and $n = 10^6$. This shows that the transition becomes even steeper for larger n and starts to resemble a step function. Indeed, as we shall prove in this thesis, there is a sharp **density threshold** $c^* \approx 0.96499$ such that for $c < c^*$ we have $\lim_{n\to\infty} p_n(c) = 1$ and for $c > c^*$ we have $\lim_{n\to\infty} p_n(c) = 0$.

In large data structures, the density thresholds for orientability, solvability and peelability reveal the memory efficiency that a cuckoo hash table or a retrieval data structure using a certain hashing scheme can reliably achieve and when a (linear time) peeling algorithm can reliably carry out the construction. For this reason, the proofs in this thesis frequently revolve around the following, more concrete question:

Q″: *Given various hashing schemes, what are their thresholds for peelability, solvability and orientability?*

We are now ready to delineate our main results.

1.1. Contributions

This thesis contains six main results, denoted by the letters A–F. For each result, there is a core mathematical theorem, denoted by the same letter, and more "computer sciency" theorems that transfer the result to data structure applications, denoted by, A1, A2, etc. An overview is given in Table 1.1.

Thm.	Subject	Notes (may sound cryptic on a first read)
A,A'	peelability	spatial coupling raises thresholds to orientability threshold
A1	dictionary	$O(m)$ construction
A2	retrieval	high space efficiency, $O(m)$ construction
A3	perfect hashing	simple $O(m)$ construction, $\approx 2.2m$ bits
A4	invertible Bloom filter	improved space efficiency
B	solvability	$m \times m$ matrix, $\Pr[\text{regular}] = \Theta(1)$, two $\log(m)$-sized blocks per row
B1	retrieval	optimal space efficiency, $\widetilde{O}(m^2)$ construction
B2	retrieval	almost optimal space efficiency, $\widetilde{O}(m^{1+\alpha})$ construction
C	solvability	$m \times (1+\varepsilon)m$ matrix, rank m whp, one $\log(m)/\varepsilon$-sized block per row
C1	retrieval	1 cache miss, space efficiency $\frac{1}{1+\varepsilon}$, $O(m/\varepsilon^2)$ construction, $O(1/\varepsilon)$ eval
C2	retrieval	theoretical improvement of C1, shaving $1/\varepsilon$ factor from running times
D	orientability	unaligned blocks yield higher threshold
D1	dictionary	unaligned blocks improves load threshold in cuckoo hashing
E	orientability	double hashing does not affect the threshold
E1	dictionary	double hashing does not affect the load threshold in cuckoo hashing
F	orientability	threshold analysis of certain unbalanced hypergraphs
F1	dictionary	small sacrifices in load allow dynamically growing cuckoo tables

Table 1.1 List of all theorems.

Results D–F derive orientability thresholds for previously studied cuckoo hashing schemes in cases where no complete analysis was known.

Results A–C are different in that they concern *novel* hashing schemes. The schemes from B and C are meant for retrieval data structures, the scheme from A has much wider applicability. Apart from analysing their thresholds, we shall argue for their overall advantages in applications, both in theory and in experiments.

We now briefly outline each individual result.

Result A. We consider a new hashing scheme called *spatial coupling* inspired from work in coding theory, parametrised by $k \in \mathbb{N}$ and $\varepsilon \in (0, 1]$. In a table of size n, the k table positions of each key are obtained by first choosing an interval of εn consecutive positions at random and then independently selecting k positions from that interval.[3] The peelability threshold of this scheme is remarkably high. For small ε it is almost the orientability threshold of the fully random scheme with the same parameter k. This yields, for instance, retrieval data structures with excellent space efficiency and a construction time of $O(n)$.

Result B. We consider a new hashing scheme with *two aligned block subsets*. The scheme resembles aligned blocks with $k = 2$, but is the more natural variant when solvability rather than orientability is concerned. The block length $L = \Theta(\log n)$ depends on the table size, and a key is assigned a uniformly random *subset* of the $2L$ positions in the two blocks.[4]

[3] In Definition 2.4 the scheme is actually parametrised by $z = \frac{1}{\varepsilon} - 1 \in \mathbb{R}^+$ for technical reasons.

[4] Note that in Definition 2.5 this will be phrased differently, in terms of matrices that contain 2 blocks of L

We prove that the solvability threshold of this scheme is $c^* = 1$. This yields a retrieval data structure with (almost) perfect space efficiency. However, a good bit of engineering is required to achieve reasonable construction times.

Result C. We consider a new hashing scheme similar to the one from B, but with *one* (rather than two) *unaligned* (rather than aligned) block subset. For $\varepsilon > 0$ and a block size of $L = \frac{\log n}{\varepsilon}$, we find a solvability threshold of $c^* \geq 1 - O(\varepsilon)$. This yields a retrieval data structure that achieves an excellent trade-off between important performance metrics in our experiments.

Result D. Among the hashing schemes with blocks, the ones without alignment achieve higher density thresholds for orientability compared to those with alignment. There has been some work on this [DW07; LP09], but we are the first to determine the thresholds for unaligned blocks exactly.

Result E. We prove that the hashing schemes "Fully Random" and "Double Hashing" have the same orientability threshold. This was conjectured in [MT12] and *partially* proven in [Lec13].

Result F. We derive orientability thresholds for a specialised hashing scheme from [MS17], which is designed for dynamically growing tables.

A comparison to previous work will be carried out after the technical details of the results are fully stated and all important performance characteristics of the data structures have been explained, see Sections 2.2.2, 2.3.3, 2.4.4 and 3.4.3.

1.2. Techniques

From the pictures in **(1.1)–(1.3)** it should be intuitively clear how a configuration of keys and table positions can be modelled by a bipartite graph. An equivalent and occasionally more convenient model uses **hypergraphs**, with each table position corresponding to a vertex and each key corresponding to the set of positions assigned to it, hence a hyperedge. When solvability is concerned, we usually consider the incidence matrix of the hypergraph because solvability simply means that the rows of this matrix are linearly independent. Therefore, much of this thesis uses corresponding language from graph theory and linear algebra. Apart from this, we routinely use standard tools from probability theory such as first moment arguments and standard concentration bounds.

For Result B, this is all that is needed. The other results, however, require more fancy equipment. Three theories stand out, the first two of which are particularly far-reaching and carry the main burden of the proofs in which they are deployed.

The Objective Method and Local Weak Convergence [AS04; BS11; Lel12; LLM13; Bor16]. Given a graph G, its *local* characteristics can be captured by sampling a vertex $\circ$ of G uniformly at random and then considering, for a small constant $t \in \mathbb{N}$, the subgraph $G(\circ)_t$ of G induced by vertices with distance at most t from $\circ$. When comparing two graphs G and H through this lense only, i.e. by comparing the distributions of $G(\circ)_t$ and $H(\circ)_t$, the graphs may appear quite similar even when there are major global differences. Imagine, for instance, that G and H are roadmaps of two countries

random bits per row.

of different sizes and with different shapes on the map. The graphs may nevertheless be nearly indistinguishable in their local characteristics, with $G(\circ)_t$ and $H(\circ)_t$ both concentrated on non-descript small planar graphs with many nodes of degree 3 and 4. For the same reason, it is possible that a sequence $(G_n)_{n\in\mathbb{N}}$ of graphs, where G_n contains n vertices, converges with respect to such local characteristics. For the (hyper-)graphs considered in this thesis, this is predominantly the case. In the spirit of the *objective method* by Aldous and Steele [AS04], we work with a corresponding infinite limiting object instead of with the infinite sequence of finite graphs in asymptotic considerations. We can then recruit a theorem by Lelarge [Lel12], which characterises orientability thresholds in terms of properties of such a limiting object. The theorem is used to prove Results D and F, but also underlies a theorem by Leconte, which we strengthen to obtain Result E. Moreover, it plays a supporting role in proving Result A.

Threshold Saturation via Spatial Coupling [KRU10; KRU11; KRU15; TTK12]. We borrow an idea from the seemingly distant field of *coding theory*. A central question in coding theory concerns the amount of redundancy that must be contained in a message to allow decoding the message even when a certain amount of information was lost during transmission. The class of *low density parity check (LDPC) codes* can be modelled by hypergraphs, much like the ones we consider in this thesis, and similar thresholds arise. The threshold up to which a simple *belief propagation* decoder is likely to succeed is often lower than the threshold up to which an ideal *maximum aposteriori probability* (MAP) decoder with unrestricted running time is likely to succeed.

In this context, the technique of *spatial coupling* superimposes a linear geometry on codes, thereby making the constructions less symmetric. In some cases, this raises the belief propagation threshold $c^\triangle$ to the MAP threshold c^*. Translated to our setting, it allows us to raise a peelability threshold $c^\triangle$ to an orientability threshold c^*, which is what Result A is aiming for.

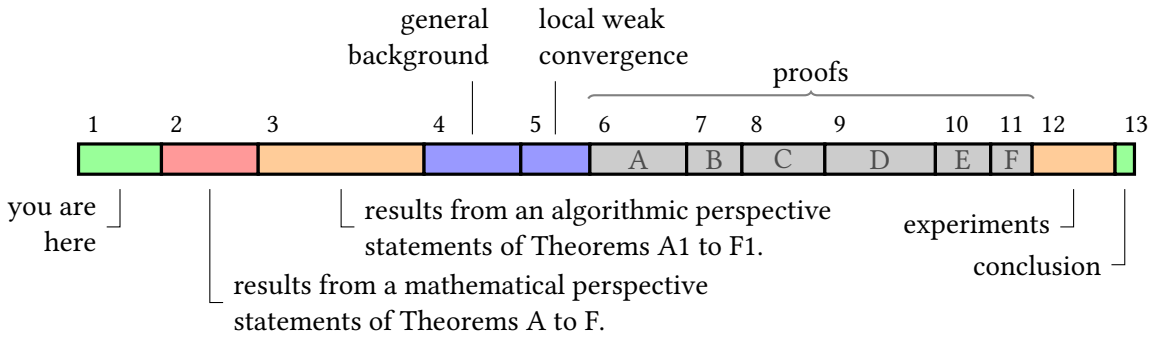
The phenomenon is counterintuitive. There is an analogous physical phenomenon, which is *also counterintuitive*, but undeniably real, for a video demonstration see [Sci14]. Consider the temperature $c^* = 0\,^\circ\mathrm{C}$ and the temperature $c^\triangle$ "at which water freezes". If the water is perfectly pure (which most water is not) it can be *supercooled* to temperatures way below $c^* = 0\,^\circ\mathrm{C}$ (down to $-48.3\,^\circ\mathrm{C}$) while staying liquid. This is because there is no *nucleation site* at which an ice crystal can start to form. By introducing an impurity in the water, we raise $c^\triangle$ to c^*, i.e. obtain water that freezes at $0\,^\circ\mathrm{C}$.

In the context of Result A, the impurities in the hashing scheme are the two ends of the table. If the table is instead glued into a ring, i.e. if intervals may "wrap around" the ends of the table, the peelability threshold is significantly lower.

Unsurprisingly, the literature surrounding the phenomenon is riddled with terminology from statistical physics, but the theorems from [KRU15], through which we access this world, largely isolates us from most of this terminology.

Queueing Theory [Coo90; EZB06]. Among our results, Result C is the only one not building on the "power of multiple choices". Its hashing scheme, using one block subset per key, has more in common with linear probing than with cuckoo hashing. We shall find a connection to Robin Hood hashing and end up using tools from queueing theory in the analysis. Queueing theory can be thought of as dealing with the distribution of waiting times in a supermarket setting, where customers follow a certain arrival distribution and cashiers follow a certain service time distribution.

1.3. Thesis Outline



- **Chapter 2** states our results from a *mathematical perspective*, speaking of peelability, solvability and orientability of hypergraphs with independent random hyperedges. A reader already familiar with applications such as cuckoo hashing and retrieval data structures will find a concise account of the main contributions.
- **Chapter 3** is a more gentle and detailed introduction from an algorithmic perspective, with all results *restated* as claims about the behaviour of certain data structures. Some readers may prefer to start there, and refer to Chapter 2 and notation on Page 12 when needed and with the motivating applications in mind.
- **Chapter 4** is a collection of background information that relates to our results without being so essential to understanding as to warrant including in Chapters 2 and 3. We shall refer to it when appropriate.
- **Chapter 5** is devoted to the notion of *local weak convergence* that is at the heart of several of our proofs.
- **Chapters 6 to 11** contain the proofs of Theorems A to F and are each based on a previously published paper as listed in Table 1.2.
- **Chapter 12** reports on various experiments relating to all theorems. A particular focus is on evaluating the performance of retrieval data structures following from Theorems A2, B2 and C2.

Thm.	Proof in	Publication	Co-Authors
A,A'	Chapter 6	[DW19b][5]/ [Wal20b]	M. Dietzfelbinger / —
B	Chapter 7	[DW19a]	M. Dietzfelbinger
C	Chapter 8	[DW19c]	M. Dietzfelbinger
D	Chapter 9	[Wal18]	—
E	Chapter 10	[MPW18]	M. Mitzenmacher and K. Panagiotou
F	Chapter 11	[MSW19]	T. Maier and P. Sanders

Table 1.2 Previously published papers.

[5] Note that [Wal20b] is a revision of [DW19b]. While both papers describe a similar construction, the new analysis is stronger, more general and more elegant. In this sense the older paper is obsolete and this thesis only presents the updated content.

To avoid misunderstandings, Figure 1.1 shows the limits of the scope of this thesis. Some non-trivial mathematical tools are taken for granted, and not every proof is self-contained. Some technical details about possible implementations are left aside, especially when other authors have pursued such questions in sufficiently similar settings.

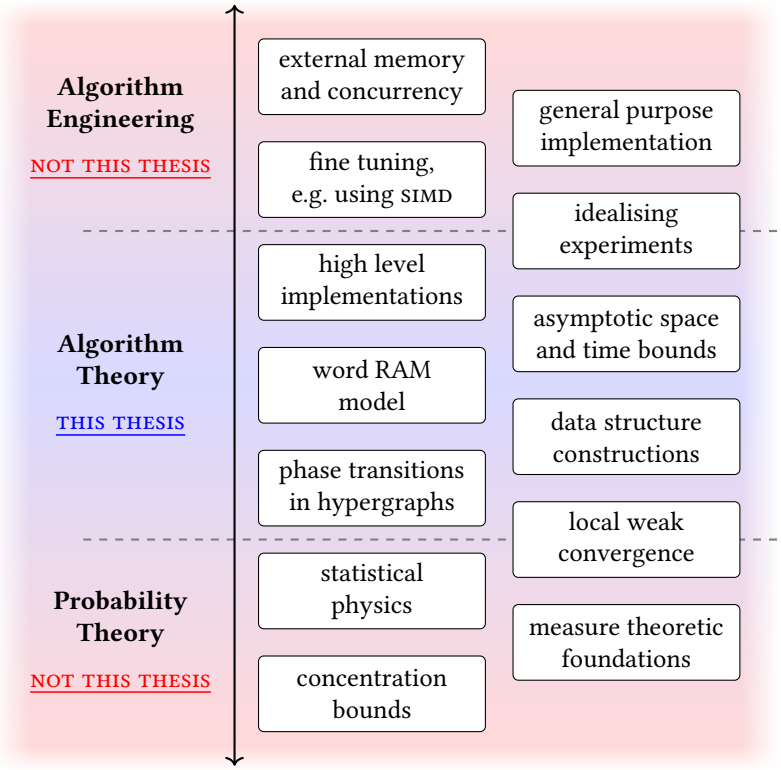


Figure 1.1 What (not) to expect from this thesis.

Part I.

Results

The main results are stated *twice*.

Chapter 2 gives them in pure mathematical form as Theorems A to F. They concern random hypergraphs and the hypergraph properties of being *peelable*, *solvable* and *orientable*.

Chapter 3 explains how the problem of constructing hashing-based data structures can be reduced to finding random hypergraphs with suitable properties. With a corresponding framing our results yield performance guarantees for data structures, stated as Theorems A1 to F1.

Notation used throughout this thesis

Sets

$[n]$	shorthand for $\{1, \dots, n\}$
$[n]_0$	shorthand for $\{0, \dots, n-1\}$
$\mathbb{Z}_n$	same as $[n]_0$, but addition is modulo n
2^V	the power set of V
$\binom{V}{k}$	set of subsets of V of size k, i.e. $\{e \subseteq V \mid \lvert e\rvert = k\}$
$\mathbb{F}_2$	the two-element field $\{0, 1\}$
$X^{\mathcal{U}}$	set of functions with domain $\mathcal{U}$ and range X
$\widetilde{O}$	big-O notation ignoring polylogarithmic factors

Random Sampling

i.i.d.	short for *stochastically independent and with identical distribution*
whp	short for "with high probability", a probability of $1 - o(1)$
$v \leftarrow\!\!\text{⚄}\; V$	v is sampled uniformly from V
$e \leftarrow\!\!\text{⚄}\; \binom{V}{k}$	a (multi-)set $e = \{v_1, \dots, v_k\}$ where $v_1, \dots, v_k \leftarrow\!\!\text{⚄}\; V$
$X \overset{\text{d}}{=} Y$	the random variables X and Y have the same distribution

Hypergraphs

$H = (V, E)$	a hypergraph
V	vertex set of a hypergraph
$E \subseteq 2^V$	hyperedge set of a hypergraph
n, m	usually $n = \lvert V\rvert$ and $m = \lvert E\rvert$. All constructions satisfy $m = \Theta(n)$
$c \in \mathbb{R}^+$	usually the hyperedge density m/n, always a constant
$A \in \{0,1\}^{E \times V}$	incidence matrix of H, i.e. $A \in \{0,1\}^{E \times V}$ with $A_{e,v} = 1 \Leftrightarrow v \in e$ Often indexed simply by $[m] \times [n]$, i.e. $A \in \{0,1\}^{m \times n}$
$e \in E$	a hyperedge $e \subseteq V$
$\vec{e} \in \{0,1\}^V$	the incidence vector A_e of e, i.e. $\vec{e}_v = 1 \Leftrightarrow v \in e$
subhypergraph	(V', E') is subhypergraph of (V, E) if $V' \subseteq V$ and $E' \subseteq E \cap 2^{V'}$
$k \in \mathbb{N}$	uniformity of a hypergraph, i.e. all hyperedges have size k
$\ell \in \mathbb{N}$	vertex capacity when discussing ℓ-peelability or ℓ-orientability

Specialised Notation

$H^k_{n,m}$	the fully random k-uniform hypergraph from Equation (2.1)
$c^{\triangle}_{k,\ell}$	ℓ-peelability threshold of $(H^k_{n,cn})_{c \in \mathbb{R}^+_0, n \in \mathbb{N}}$
$c^{*}_{k,\ell}$	ℓ-orientability threshold of $(H^k_{n,cn})_{c \in \mathbb{R}^+_0, n \in \mathbb{N}}$
F_n	The random hypergraph $F(n, k, c, z)$ from Definition 2.4
$A^L_{m,n}$	the random matrix from Definition 2.5
$M^L_{m,n}$	the random matrix from Definition 2.6
$B^{k,\ell}_{n,m}$	the random hypergraph from Definition 2.7
$W^{k,\ell}_{n,m}$	the random hypergraph from Definition 2.8
$\gamma_{k,\ell}$	orientability threshold of $W^{k,\ell}_{n,m}$
$D^k_{n,m}$	the random hypergraph from Definition 2.9
$H^k_{n,i,m}$	the random hypergraph from Definition 2.10
$\eta_{k,\ell,\theta}$	ℓ-orientability threshold of $(H^k_{n,n/(1+\theta),cn})_{c \in \mathbb{R}^+, n \in \mathbb{N}}$

2. Random Hypergraphs: Peeling, Solving, Orienting

We introduce *random hypergraphs with i.i.d. random hyperedges* and corresponding notation (see facing page). A hypergraph $H = (V, E)$ consists of a finite set V and a set $E \subseteq 2^V$ of subsets of V. The elements of V are *vertices*, the elements of E are *hyperedges*. If all hyperedges have the same size $k \in \mathbb{N}$, then H is called *k-uniform*. A subhypergraph of H is a hypergraph $H' = (V', E')$ with $V' \subseteq V$ and $E' \subseteq E \cap 2^{V'}$. Sometimes it is convenient to work with the transposed incidence matrix $A \in \{0,1\}^{E \times V}$ of H over the field $\mathbb{F}_2 = \{0,1\}$, with rows $\vec{e} := A_e \in \{0,1\}^V$ for $e \in E$ corresponding to hyperedges and columns corresponding to vertices. Each 1-entry indicates an incidence, i.e. for $e \in E, v \in V : A_{e,v} = 1 \Leftrightarrow \vec{e}_v = 1 \Leftrightarrow v \in e$. Examples are given in Figure 2.1.

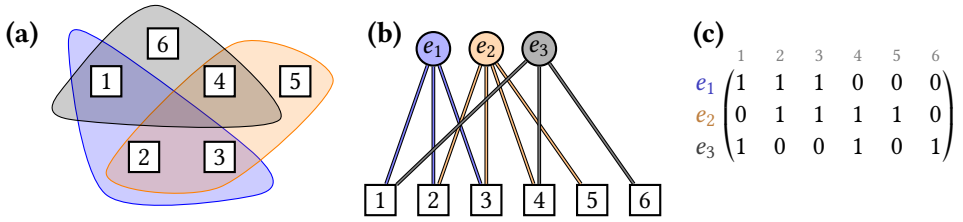


Figure 2.1 The hypergraph $H = (\{1,2,3,4,5,6\}, \{\{1,2,3\}, \{2,3,4,5\}, \{1,4,6\}\})$ represented in three different ways. Instead of drawing geometric regions for hyperedges as in **(a)**, we use bipartite drawings as in **(b)** or incidence matrices as in **(c)**.

This thesis is concerned with random hypergraphs containing random hyperedges that are sampled independently and with identical distribution (i.i.d.). Such a random hypergraph is specified by a vertex set V of size $n \in \mathbb{N}$, a number of hyperedges $m \in \mathbb{N}$, and a distribution D on 2^V. The hyperedge set E of H is obtained by sampling from D independently m times. In this sense, $H = (V, E)$ is a random variable.

Throughout this thesis, we use the notation $[n] = \{1, \dots, n\}$, $[n]_0 = \{0, \dots, n-1\}$ and $\binom{V}{k} = \{e \subseteq V \mid |e| = k\}$. Moreover, $v \leftarrow V$ stands for sampling a value v uniformly from the set V, i.e. it introduces a random variable v with uniform distribution on V that is independent of all other random variables introduced with the same notation. The shorthand $e \leftarrow \binom{V}{k}$ stands for $e = \{v_1, \dots, v_k\}$ where $v_1, \dots, v_k \leftarrow V$. An important example is the *fully random k-uniform hypergraph* $H^k_{n,m}$ for $k, m, n \in \mathbb{N}$ given as

$$H^k_{n,m} := (V = [n], E = \{e_1, e_2, \dots, e_m\}), \quad \text{where } e_i \leftarrow \binom{V}{k} \text{ for } i \in [m]. \tag{2.1}$$

There is a subtle difference between $e \leftarrow \binom{V}{k}$ and $e \leftarrow \binom{V}{k}$ as, in the former case, the k elements of e need not be distinct. We therefore understand e as a multiset.[1] Also, we

[1] In the incidence matrix it is sometimes convenient to understand $\vec{e}_v \in \{0,1\}$ as the number of copies of v in

may have $e_i = e_j$ for $i \neq j$, so E is a multiset as well. Such details are often inconsequential. A hypergraph chosen uniformly from the set of all hypergraphs on vertex set $[n]$ with m *distinct* hyperedges all of which contain k *distinct* vertices would be equivalent to $H^k_{n,m}$ for our purposes, just less convenient to work with.

The parameter c denotes the *hyperedge density* $|E|/|V|$ in all constructions, at least approximately. Rounding issues are often ignored for sake of clarity.

2.1. Three Hypergraph Properties

The following well-studied properties will accompany us throughout the thesis. The second one is not usually taken to be a *hypergraph* property, but we phrase it this way for a unified presentation.

▶ **Definition 2.1** (Peelability, Solvability, Orientability)**.** *Let $H = (V, E)$ be a hypergraph with transposed incidence matrix $A \in \{0,1\}^{E \times V}$.*

- **(i)** *For $\ell \in \mathbb{N}$, H is* ℓ-peelable *if every subhypergraph of H has minimum degree at most ℓ. In other words, H has an empty* $(\ell{+}1)$-core*, which is defined to be the largest subhypergraph of H with minimum degree at least $(\ell + 1)$. Equivalently, the* peeling process *that repeatedly deletes all vertices of degree at most ℓ (and incident hyperedges) reaches the empty hypergraph.* [Coo04; JL07; Mol05]
- **(ii)** *H is* solvable *if A has rank $|E|$. This necessitates $|V| \geq |E|$.* [Die+10; DM02; PS16]
- **(iii)** *For $\ell \in \mathbb{N}$, H is* ℓ-orientable *if every subhypergraph $H' = (V', E')$ of H satisfies $|E'|/|V'| \leq \ell$. By Hall's Theorem [Hal35], this is equivalent to the existence of a map $\sigma\colon E \to V$ with $\sigma(e) \in e$ for $e \in E$ and $|\sigma^{-1}(v)| \leq \ell$ for $v \in V$. We call σ an* ℓ-orientation *of H.* [CSW07; FR07; FKP16; FP10; GW10; Lel12]

It is a simple observation that these properties are contained in each other in the following sense (see, e.g. [Die+10]).

▷ Fact 2.2. For any hypergraph H we have:

- **(i)** $\forall \ell \in \mathbb{N}:$ H is ℓ-peelable $\Rightarrow$ H is ℓ-orientable.
- **(ii)** H is 1-peelable $\Rightarrow$ H is solvable.
- **(iii)** H is solvable $\Rightarrow$ H is 1-orientable.

Proof.

- **(i)** Consider the peeling process when H is ℓ-peelable. The deletion of any vertex v causes the deletion of $\ell_v \leq \ell$ hyperedges $e_1, \ldots, e_{\ell_v}$ and we define $\sigma(e_1) = \ldots = \sigma(e_{\ell_v}) = v$. All hyperedges are deleted eventually, so we obtain an ℓ-orientation σ.
- **(ii)** Assume H is 1-peelable. Consider the sequence $H_0 = H, H_1, \ldots, H_m$ of hypergraphs where H_i arises from H_{i-1} for $i \in [m]$, by picking a vertex v_i of degree 1 in H_{i-1} and removing v_i as well as the incident hyperedge e_i. The final hypergraph H_m contains $n - m$ isolated vertices $v_{m+1}, \ldots, v_n$ and no hyperedges. By construction, $v_i \in e_i$ for $i \in [m]$ and $v_i \notin e_{i'}$ for $1 \leq i < i' \leq m$. Thus, with the ordering $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$, the matrix A is in row echelon form and clearly has rank m. Note that rearranging rows and columns of a matrix does not affect its rank.

---

e modulo 2, meaning double incidences "cancel out".

(iii) Again considering A, solvability guarantees the existence of an $|E| \times |E|$ submatrix of A with determinant 1. Let $V' \subseteq V$ with $|V'| = |E|$ be the inducing vertex set. By the Leibniz formula for the determinant, there exists a bijection $\sigma\colon E \to V'$ with $\prod_{e\in E} A_{e,\sigma(e)} \neq 0$. In particular, σ is injective and satisfies $\sigma(e) \in e$ for $e \in E$. Thus, σ is a 1-orientation. ◀

Later we see data structures that work efficiently only if an underlying random hypergraph H has one of these properties. Subject to H having a property with high probability, we often try to optimise other parameters of H. Asymptotic considerations of this kind are captured in the following definition.

▶ **Definition 2.3** (Threshold, see e.g. [FB99]). *Let $\mathcal{P}$ be a monotone hypergraph property, meaning if H' is a subhypergraph of H then $H \in \mathcal{P}$ implies $H' \in \mathcal{P}$. Moreover, let $(H_{c,n})_{c\in\mathbb{R}_0^+,n\in\mathbb{N}}$ be a family of random hypergraphs that is monotone in c, meaning for $n \in \mathbb{N}$ and $c < c'$ there is a coupling[2] between $H_{c,n}$ and $H_{c',n}$ such that almost surely $H_{c,n}$ is a subhypergraph of $H_{c',n}$. The* threshold *of $(H_{c,n})_{c\in\mathbb{R}_0^+,n\in\mathbb{N}}$ for $\mathcal{P}$ is defined as*

$$c^* := \sup\{c \in \mathbb{R}_0^+ \mid \Pr[H_{c,n} \in \mathcal{P}] \overset{n\to\infty}{\longrightarrow} 1\}.$$

We say the threshold is sharp *if*

$$c^* = \inf\{c \in \mathbb{R} \mid \Pr[H_{c,n} \in \mathcal{P}] \overset{n\to\infty}{\longrightarrow} 0\}.$$

Important examples are, for natural numbers $k \geq 2$ and $\ell \geq 1$, the thresholds $c^*_{k,\ell} \in (0, \ell)$ of $(H^k_{n,cn})_{c\in\mathbb{R}_0^+,n\in\mathbb{N}}$ for ℓ-orientability. These thresholds are sharp and known exactly [FKP11; Lel12; FKP16]. We reproduce them in table Table 2.3. It is easy to see that $c^*_{k,\ell}/\ell$ approaches 1 as $k + \ell$ increases, see Table 2.4. The thresholds $c^{\triangle}_{k,\ell}$ for ℓ-peelability of $(H^k_{n,cn})_{c\in\mathbb{R}_0^+,n\in\mathbb{N}}$ are also known. They are sharp with the exception of $c^{\triangle}_{2,1}$ [Mol05], we give them in Table 2.5.

2.2. Peelable Hypergraphs at High Densities

2.2.1. Spatially Coupled Hypergraphs

For $k \geq 2$ we find a distribution on hyperedges that yields k-uniform random hypergraphs with an *ℓ-peelability threshold* arbitrarily close to the *ℓ-orientability threshold* of fully random k-uniform hypergraphs, for all $\ell \geq 1$ with $(k, \ell) \neq (2, 1)$. This is achieved by *spatial coupling* (see below). Formally:

▶ **Definition 2.4** (Spatially Coupled Hypergraph Family). *For $k \in \mathbb{N}$, $z, c \in \mathbb{R}_0^+$ and $n \in \mathbb{N}$, let $F_n = F(n, k, c, z)$ be the random k-uniform hypergraph with vertex set $V = [n]_0$ and edge set E of size $m = \lfloor cn\frac{z}{z+1} \rfloor$. Each edge $e \in E$ is independently obtained as*

$$e = \left\{\left\lfloor \tfrac{y+o_i}{z+1} n \right\rfloor \mid i \in [k]\right\} \text{ where } y \leftarrow\!\!\boxdot\!\!- \ [\tfrac{1}{2}, z + \tfrac{1}{2}), \quad o_1, \ldots, o_k \leftarrow\!\!\boxdot\!\!- \ [-\tfrac{1}{2}, \tfrac{1}{2}].$$

We call $y \in Y := [\frac{1}{2}, z + \frac{1}{2})$ the position of e and $\frac{v(z+1)}{n} \in X = [0, z + 1)$ the position of $v \in V$.

[2] An embedding of both random variables in a common probability space.

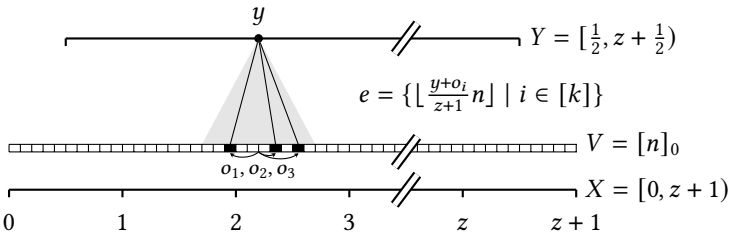


Figure 2.2 In the construction from Definition 2.4, the vertex set $V = [n]_0$ is arranged linearly along the "coupling dimension" $X = [0, z+1)$. Thus each vertex has a position $x \in X$. Each hyperedge e is randomly obtained as follows. First, pick a random position $y \leftarrow Y = [\frac{1}{2}, z + \frac{1}{2})$. Then pick the k incidences of e uniformly at random from the vertices with positions in $[y - \frac{1}{2}, y + \frac{1}{2}]$.

In Figure 2.2 we sketch aspects of the construction. Note that for technical reasons the hyperedge density is $|E|/|V| = c\frac{z}{z+1}$ and only approaches c for large z. We now state our related theorem.

▶ **Theorem A.** *Let $k, \ell \in \mathbb{N}$, with $k \geq 2$ and $k + \ell \geq 4$. Then we have:*

(i) $c < c^*_{k,\ell} \Rightarrow \forall z \in \mathbb{R}^+ \colon \Pr[F_n \text{ is } \ell\text{-peelable}] \xrightarrow{n\to\infty} 1.$

(ii) $c > c^*_{k,\ell} \Rightarrow \exists z^* \in \mathbb{R}^+ \colon \forall z \geq z^* \colon \Pr[F_n \text{ is } \ell\text{-orientable}] \xrightarrow{n\to\infty} 0.$

Let us distil the main takeaways from these claims.

▶ **Theorem A'.** *Let $k, \ell \in \mathbb{N}$ with $k \geq 2$ and $k + \ell \geq 4$. For $z \in \mathbb{R}^+$ consider the family $(F(n, k, c, z))_{c\in\mathbb{R}^+_0, n\in\mathbb{N}}$. Let $f_{k,\ell,z}$ be its threshold for ℓ-peelability and $f^*_{k,\ell,z}$ its threshold for ℓ-orientability. Then we have:*

(i) $\forall z \in \mathbb{R}^+ \colon f_{k,\ell,z} \geq c^*_{k,\ell}$.

(ii) $\limsup_{z\to\infty} f^*_{k,\ell,z} \leq c^*_{k,\ell}$.

(iii) *Let $f_{k,\ell} = \lim_{z\to\infty} f_{k,\ell,z}$ and $f^*_{k,\ell} = \lim_{z\to\infty} f^*_{k,\ell,z}$. Then $f_{k,\ell} = f^*_{k,\ell} = c^*_{k,\ell}$.*

(iv) *The "diagonal" family $(\hat{F}^{k,\ell}_{n,cn} := F(n, k, c\frac{z+1}{z}, z = 2\ell/(c^*_{k,\ell} - c)))_{c\in[0,c^*_{k,\ell}),n\in\mathbb{N}}$ of random hypergraphs has ℓ-peelability threshold[3] $c^*_{k,\ell}$. Note that $\hat{F}^{k,\ell}_{n,cn}$ has n vertices and cn i.i.d. hyperedges of size k.*

Proof of Theorem A'. Claims **(i)** and **(ii)** are immediate consequences of the claims from Theorem A. Since $f_{k,\ell,z} \leq f^*_{k,\ell,z}$ (using Fact 2.2 **(i)**) we conclude **(iii)**.

Lastly for **(iv)**, let $c = c^*_{k,\ell} - \varepsilon$ for some $\varepsilon > 0$ and $z = 2\ell/\varepsilon$. Observe that $c\frac{z+1}{z} = c\frac{2\ell/\varepsilon+1}{2\ell/\varepsilon} = c(1+\varepsilon/(2\ell)) < c^*_{k,\ell} - \varepsilon/2 \leq f_{k,\ell,z} - \varepsilon/2$ using **(i)**. Therefore $\hat{F}^{k,\ell}_{n,cn}$ is ℓ-peelable with probability tending to 1 by definition of $f_{k,\ell,z}$. ◀

[3] This requires a natural extension of Definition 2.3 since the considered family is not monotone in c, and c is restricted to a subset of $\mathbb{R}^+_0$.

We remark that $(\hat{F}^{k,\ell}_{n,cn})_{c,n}$ is mostly of theoretical interest. We do not claim that our choice of $z = \varepsilon/2$ is particularly practical. The fact that the number of hyperedges has an effect on the underlying parameter z is also inconvenient in applications to dynamic data structures.

Discussion. Our construction is in the spirit of a technique from coding theory. Namely, our hypergraphs arise from the fully random hypergraphs via *spatial coupling*, see [KRU15; KRU13], along the one-dimensional coupling dimension $X = [0, z + 1)$. Doing so, we achieve *threshold saturation*. In Section 6.1 we give more background information on this remarkable phenomenon.

Constructions similar to ours can already be found in [HMU13] and [GMU12], however, the goals of these papers are very different. Relative to these results, we can offer: **(1)** A generalisation to $\ell > 1$. **(2)** A more elegant construction using the updated tools from [KRU15] (continuous[4] coupling dimension). **(3)** A framing with data structures in mind and a demonstration of practical benefits for data structures, as found in Chapter 3.

2.2.2. Previous Constructions of Peelable Hypergraphs

For the applications in Chapter 3, we require hypergraph families with i.i.d. random hyperedges, small average hyperedge size k and large ℓ-peelability threshold c^*. The trade-offs achieved by the family $(\hat{F}^{k,\ell}_{n,cn})_{c\in[0,c^*_{k,\ell}),n\in\mathbb{N}}$ with ℓ-peelability threshold $c^*_{k,\ell}$ from Theorem A' surpass all known constructions. Concerning $\ell > 1$, to our knowledge, only the fully random families $(H^k_{n,cn})_{c\in\mathbb{R}^+_0,n\in\mathbb{N}}$ for $k \geq 2$ where studied, with much lower thresholds $c^{\triangle}_{k,\ell}$, see Table 2.5. For 1-peelability, see Figure 2.3.

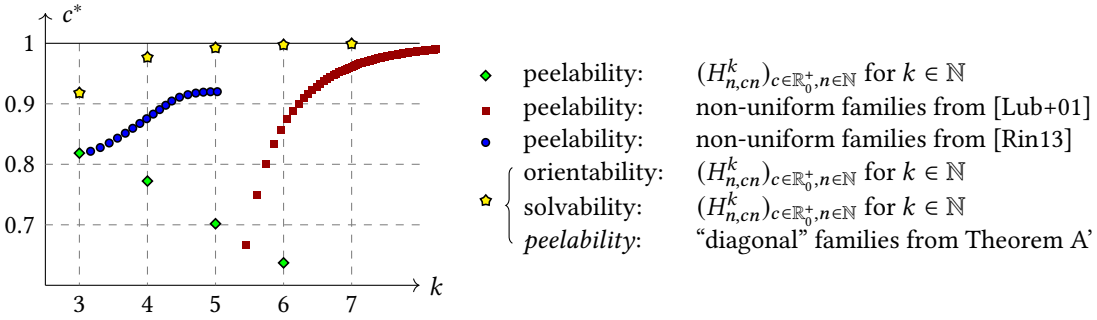


Figure 2.3 Trade-offs between the hyperedge size and threshold density of hypergraph families with respect to 1-peelability, solvability or 1-orientability.
A dot at $(k, c^*) \in \mathbb{R}^2$ indicates a family $(H_{c,n})_{c\in\mathbb{R}^+_0,n\in\mathbb{N}}$ of random hypergraphs where $H_{c,n}$ has n vertices, $\lfloor cn \rfloor$ random independent hyperedges and expected hyperedge size k. The value c^* is the threshold.

The thresholds $c^{\triangle}_{k,1}$ of the families $(H^k_{n,cn})_{c\in\mathbb{R}^+_0,n\in\mathbb{N}}$ for $k \geq 3$ (◆) [Mol05] are decreasing in k and thus only $k = 3$ is of interest. The threshold $c^*(k)$ of a non-uniform construction (■) [Lub+01], well-known in coding theory, approaches 1 for $k \to \infty$. However, the maximum hyperedge size is exponential in the average hyperedge size k, which is problematic for some applications. Further trade-offs (●) were examined by [Rin13], for example, a hyperedge size of 3 for $\approx 89\%$ of the hyperedges and a size of 21 for the rest in an otherwise fully random construction yields an average hyperedge size of ≈ 5.03 and a threshold value of ≈ 0.92. For $k \geq 3$, the threshold of the family $(\hat{F}^{k,1}_{n,cn})_{c\in[0,c^*_{k,1}),n\in\mathbb{N}}$ (★) is the 1-*orientability*

[4] In [HMU13] the coupling dimension is discrete. In our terms, this means that the set of admissible positions of a hyperedge is $Y \cap (\frac{1}{w}\mathbb{Z})$ for some constant $w \in \mathbb{N}$. Our construction arises for $w \to \infty$.

threshold $c^*_{k,1}$ of $(H^k_{n,cn})_{c\in\mathbb{R}^+_0,n\in\mathbb{N}}$. The solvability threshold of $(H^k_{n,cn})_{c\in\mathbb{R}^+_0,n\in\mathbb{N}}$ is known to be equal to $c^*_{k,1}$ as well [DM02; Die+10; PS16].

We remark that, even for 1-orientability, no known hypergraph family with i.i.d. random hyperedges and average hyperedge size at most k exceeds a threshold of $c^*_{k,1}$.

2.3. Solvability and Good Locality at High Densities

We now deal directly with the transposed incidence matrix $A \in \{0,1\}^{m\times n}$ of hypergraphs, a matrix over the field $\mathbb{F}_2 = \{0,1\}$, with i.i.d. random rows. For the applications from Chapter 3 the following is desirable:

(1) H should be solvable, i.e. A should have rank m.

(2) $c = m/n \le 1$ should be as close to 1 as possible.

(3) The scalar product $\langle \vec{e}, \vec{z}\rangle$ of a row $\vec{e} \in \mathbb{F}_2^n$ of A and any $\vec{z} \in \mathbb{F}_2^n$ should be efficient to compute. To avoid discussing algorithms for now, we approximate this with the strictly weaker demand that the entropy of $\vec{e}$ (viewed as a random variable) should be small, ideally $O(\log n)$.

(4) For any $\vec{b} \in \{0,1\}^m$, the linear system $A\vec{z} = \vec{b}$ should be efficient to solve, ideally in time $O(n)$.

Note that if A is the incidence matrix of a 1-peelable hypergraph, then **(1)** is automatically satisfied and for **(4)** we obtain a running time of $O(\psi)$ where ψ is the number of 1-entries of A (c.f. Fact 2.2 **(ii)**). Indeed, our 1-peelable hypergraphs from Section 2.2 work well in some applications.

2.3.1. New Construction with Two Random Blocks per Row

▶ **Definition 2.5** (Random Matrix with Two Aligned Blocks per Row)**.** *Let $n, m, L \in \mathbb{N}$ with $m \le n$ and L a divisor of n. For a* block index $b \in [n/L]$ *and a* pattern $p \in \{0,1\}^L$ *we let $B_{b,p} = 0^{bL-L} \circ p \circ 0^{n-bL} \in \{0,1\}^n$ where "$\circ$" denotes concatenation of bit strings. To obtain the random matrix $A^L_{m,n} \in \{0,1\}^{m\times n}$, each row is independently sampled as $B_{b_1,p_1} \oplus B_{b_2,p_2}$ where $b_1, b_2 \leftarrow [n/L]$ and $p_1, p_2 \leftarrow \{0,1\}^L$.*

We define $A^{L*}_{m,n}$ the same way, except for $p_1, p_2 \leftarrow \{0,1\}^L \setminus \{0^L\}$ the all-zero pattern is forbidden.

An example is given in Figure 2.4 **(a)**. The entropy of a row of $A^L_{m,n}$ is $O(L + \log(n/L))$ which is $O(\log n)$ for $L = O(\log n)$. The following theorem states that $A^L_{m,n}$ also satisfies **(1)** and **(2)** under suitable conditions. We cannot satisfy **(4)**, and will require a work-around in our applications, see Section 3.4.2.

▶ **Theorem B.** *Let $\beta = 39$, $\gamma = 1/4$ and $\delta > 0$ constants. Then we have:*

(i) *If $L = L(n) \ge \beta \log_2(n)$ then $A^L_{n,n}$ is regular with probability $\Theta(1)$.*

(ii) *If $2L = 2L(n) \ge (1+\delta)\log_2 n$ and $1 - c \ge \max\{2^{-\gamma L}, \beta \log_2(n)/n\}$ then $A^L_{cn,n}$ has independent rows with probability at least $1 - \widetilde{O}(n^{-\min(1,\delta)})$.*

(iii) *If $1 - c \ge \max\{2^{-\gamma L}, \beta \log_2(n)/n\}$ then $A^{L*}_{cn,n}$ has independent rows with probability at least $1 - O(\max(n^{-1}, L2^{-L}))$.*

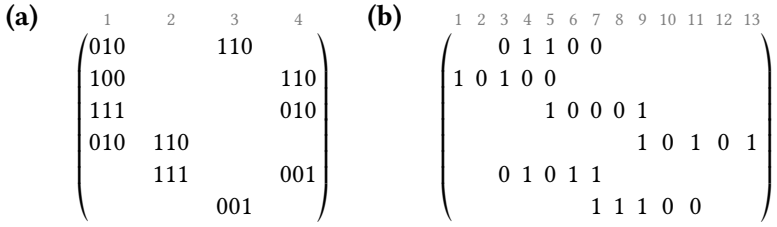


Figure 2.4 **(a)** Possible outcome for $A^3_{6,12}$ from Definition 2.5. There are $n/L = 12/3 = 4$ (aligned) blocks and in each row two randomly chosen blocks are populated with random patterns. Since the same block may be selected twice, rows with only one populated block can occur.
(b) Possible outcome for $M^5_{6,9}$ from Definition 2.6. There are $n + L - 1 = 9 + 5 - 1 = 13$ columns. In each row, one randomly chosen (unaligned) block of size L is populated with a random pattern.

2.3.2. New Construction with One Random Block per Row

To simplify some notation down the line, in the following definition, the number of matrix columns exceeds n by $L - 1$ "auxiliary columns", which will always be negligible.

▶ **Definition 2.6** (Random Matrix with One Unaligned Block per Row). *Let $n, m, L \in \mathbb{N}$ with $m \leq n$. For a* starting position *$s \in [n]$ and a* pattern *$p \in \{0,1\}^L$ we let $B_{s,p} = 0^{s-1} \circ p \circ 0^{n-s} \in \{0,1\}^{n+L-1}$. To obtain the random matrix $M^L_{m,n} \in \{0,1\}^{m \times (n+L-1)}$, each row is independently sampled as $B_{s,p}$ where $s \xleftarrow{\text{🎲}} [n]$ and $p \xleftarrow{\text{🎲}} \{0,1\}^L$.*

An example is given in Figure 2.4 **(b)**. In contrast to Definition 2.5, each row contains only one random pattern, which is not aligned to a multiple of the pattern length.

The row entropy is $O(L + \log n)$. The following theorem illuminates how $M^L_{m,n}$ fares with respect to **(1)**, **(2)** and **(4)**.

▶ **Theorem C.** *Let $0 < \varepsilon < \frac{1}{2}$ be a constant, $n \in \mathbb{N}$ and $m = (1-\varepsilon)n$. For some $L = O((\log n)/\varepsilon)$ the matrix $M^L_{m,n}$ has independent rows with probability $1 - O(1/n)$.*

Moreover, a simple algorithm computes a solution $\vec{z}$ to $M^L_{m,n}\vec{z} = \vec{b}$ for an arbitrary right hand side $\vec{b} \in \{0,1\}^m$. Its expected running time is dominated by $O(n/\varepsilon)$ row additions. Each row is always zero outside of a block of length L.

2.3.3. Previous Constructions of Matrices with Full Rank

Table 2.1 lists known constructions of random matrices with independent rows and compares them to our own with respect to the criteria **(1)** – **(4)** defined above. Constructions previously introduced as hypergraphs should now be interpreted as constructions of corresponding (transposed incidence) matrices.

As a baseline, we consider $C^k_n \in \{0,1\}^{n \times n}$ for $n \in \mathbb{N}, 0 \leq k \leq n$, a special case of a class of matrices studied by Cooper [Coo00]. All entries of C^k_n are i.i.d. random bits and equal to 1 with probability k/n. For $k = n/2$ we obtain fully random matrices, but $k = \log n + \omega(1)$ suffices for C^k_n to have full rank with probability approaching $\phi(\frac{1}{2}) \approx 0.289$, where $\phi(q) = \prod_{k=1}^{\infty}(1 - q^k)$ is known as the Euler function [Coo00].

construction	source	**(1)**: probability	**(2)**: $c = \frac{m}{n}$	**(3)**: row entropy	**(4)** time to solve
$C_n^{n/2}$	[Coo00]	$\Theta(1)$	$c = 1$	n	$\widetilde{O}(n^3)$
$C_n^{\log n+\omega(1)}$	[Coo00; DP08]	$\Theta(1)$	$c = 1$	$\geq \log^2 n$	$\widetilde{O}(n^2)$
$H_{n,cn}^k$	[PS16; Die+10]	$1 - o(1)$	$c < c_{k,1}^*$	$\approx k \log(n)$	$\widetilde{O}(n^2)$
$A_{n,n}^{\Theta(\log n)}$	Theorem B **(i)**	$\Theta(1)$	$c = 1$	$\Theta(\log(n))$	$\widetilde{O}(n^2)$
$A_{cn,n}^{\Theta(\log n)}$	Theorem B **(ii)**	$1 - o(1)$	$c = 1 - O(\frac{\log n}{n})$	$\Theta(\log(n))$	$\widetilde{O}(n^2)$
$M_{cn,n}^{\Theta(\log(n)/\varepsilon)}$	Theorem C	$1 - o(1)$	$c = 1 - \varepsilon$	$\Theta(\log(n)/\varepsilon)$	$\widetilde{O}(n/\varepsilon^2)$
$H_{n,cn}^3$	[Maj+96; BPZ13]	$1 - o(1)$	$c < c_{3,1}^{\triangle}$	$\approx 3 \log n$	$\widetilde{O}(n)$
$\hat{F}_{n,cn}^{k,1}$	Theorem A'	$1 - o(1)$	$c < c_{k,1}^*$	$\approx k \log n$	$\widetilde{O}(n)$

Table 2.1 Performance of certain random matrices with respect to the criteria defined in Section 2.3.

Compared to C_n^k, which contains k ones per row *in expectation*, $H_{n,n}^k$ contains *exactly* k ones per row[5]. The similarity for $k = \Omega(\log n)$ is discussed in [DP08]. For $k < \log n$, all-zero rows are likely occurrences in C_n^k, but not in $H_{n,n}^k$.

Our constructions $A_{cn,n}^{L/2}$ and $M_{cn,n}^{L}$ can be seen as variations on $H_{cn,n}^{L}$ where the 1-entries are concentrated in 1 or 2 randomly placed blocks, which reduces the row entropy from $\approx L \cdot \log n$ to $\approx L + \log n$.

When using $\hat{F}_{n,cn}^{k,1}$ from Theorem A' and $H_{n,cn}^k$ below their respective peelability thresholds, the linear systems can be solved by peeling in $O(kn)$ time, see the proof of Fact 2.2 **(ii)**. All quadratic solving times are achieved using Wiedemann's algorithm, explained in Section 4.4.1. We use $\widetilde{O}$-notation to avoid a discussion about bit-level parallelism for now.

2.4. Orientability of Variations of Fully Random Hypergraphs

The fully random hypergraph $H_{n,cn}^k$ directly underlies many hashing-based data structures, see Chapter 3. It is natural to study variations of it, hoping for improvements in practically relevant metrics.

Our first variation improves upon the orientability threshold. The second one uses double hashing to reduce the entropy per hyperedge from $k \log n$ to $2 \log n$ without affecting the orientability threshold. The third one studies intermediate hypergraphs that arise naturally in dynamic settings, where the vertex set should grow smoothly.

2.4.1. Higher Thresholds using Unaligned Blocks

To clearly expose the following idea, we first reconceptualise what ℓ-orientability of $H_{n,m}^k$ means.

[5] Strictly speaking, the k positions $v_1, \ldots, v_k \xleftarrow{\$} [n]$ may not be distinct. In such rare cases, there are less than k ones per row.

▶ **Definition 2.7** (Random Hypergraph with Aligned Blocks). *Let $n, m, k, \ell \in \mathbb{N}$ with n a multiple of ℓ. The random hypergraph with k* aligned blocks of size ℓ *is given as*

$$B_{n,m}^{k,\ell} := ([n]_0, \{e_1', \dots, e_m'\}), \text{ where } e_i \leftarrow \binom{[n/\ell]_0}{k}$$
$$\text{and } e_i' = \bigcup_{j \in e_i} \{j\ell, \dots, (j+1)\ell - 1\} \text{ for } i \in [m].$$

Each hyperedge is the union of k aligned blocks chosen uniformly at random from the set of all blocks. By *aligned blocks* we mean the n/ℓ intervals of size ℓ in $[n]_0$ that start at a multiple of ℓ.

We can think of $B_{n,m}^{k,\ell}$ as arising from $H_{n/\ell,m}^{k}$ by splitting each of the n/ℓ vertices of $H_{n/\ell,m}^{k}$ into ℓ vertices as seen in Figure 2.5 **(a)** and **(b)**. With this coupling between the probability spaces, $H_{n/\ell,m}^{k}$ is ℓ-orientable if and only if $B_{n,m}^{k,\ell}$ is 1-orientable. In particular the 1-orientability threshold of $(B_{n,cn}^{k,\ell})_{c\in\mathbb{R}_0^+, n\in\mathbb{N}}$ is $c_{k,\ell}^*/\ell$. These "normalised" *load thresholds* (see Table 2.4) are a natural measure of space efficiency on a scale of $[0, 1]$ and will play a role in Chapter 3.

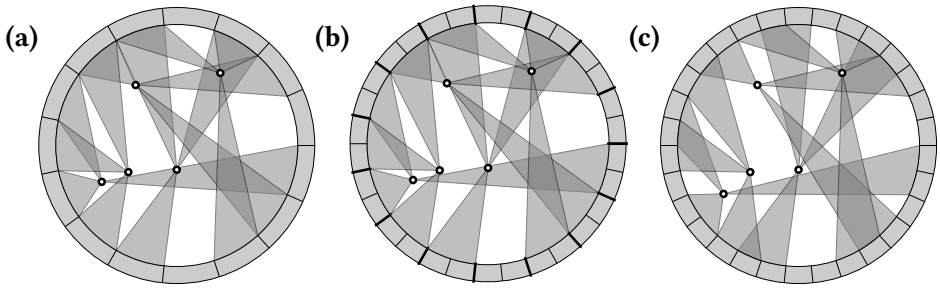


Figure 2.5 Drawing of corresponding outcomes for the hypergraphs $H_{n/\ell,m}^{k}$ and $B_{n,m}^{k,\ell}$ in **(a)** and **(b)**, as well as an outcome for $W_{n,m}^{k,\ell}$ in **(c)**, with parameters $n = 30$, $m = 5$, $k = 3$ and $\ell = 2$. Each hyperedge is drawn as a point and connected to all incident vertices, which are arranged in a circle. In the case of $B_{n,m}^{k,\ell}$, thick lines indicate borders between blocks.

From $B_{n,m}^{k,\ell}$ a slight modification yields the following construction.

▶ **Definition 2.8** (Random Hypergraph with Unaligned Blocks). *Let $n, m, k, \ell \in \mathbb{N}$. The random hypergraph with k* unaligned blocks of size ℓ *is given as*

$$W_{n,m}^{k,\ell} := (\mathbb{Z}_n, \{e_1', \dots, e_m'\}), \text{ where } e_i \leftarrow \binom{\mathbb{Z}_n}{k} \text{ and } e_i' = \bigcup_{j \in e_i} \{j, \dots, j + \ell - 1\} \text{ for } i \in [m].$$

In $W_{n,m}^{k,\ell}$, each hyperedge is the union of k intervals chosen uniformly at random from the set of all n intervals of size ℓ in the cyclic group $\mathbb{Z}_n = ([n]_0, +)$, this time without alignment restriction. Note that intervals wrap around at the ends of the set $[n]_0$ with no awkward "border intervals". An example is given in Figure 2.5 **(c)**.

We provide precise 1-orientability thresholds of $(W_{n,cn}^{k,\ell})_{c\in\mathbb{R}^+, n\in\mathbb{N}}$ for all $k, \ell \geq 2$. In particular this solves the case of $k = 2$ considered in [DW07; LP09].

▶ **Theorem D.** *Let $k, \ell \geq 2$. The 1-orientability threshold $\gamma_{k,\ell}$ of $(W_{n,cn}^{k,\ell})_{c\in\mathbb{R}^+, n\in\mathbb{N}}$ is characterised by Equation (9.8) in Chapter 9. It exceeds the threshold of $(B_{n,cn}^{k,\ell})_{c\in\mathbb{R}^+, n\in\mathbb{N}}$, at least for $(k, \ell) \in \{2, \dots 7\} \times \{2, \dots, 10\}$.*

The thresholds when using unaligned blocks seem to be significantly closer to 1 than when blocks are aligned. For instance, in the case of $k = \ell = 2$ the threshold is at roughly 96.5% instead of at roughly 89.7%. We provide some values of $\gamma_{k,\ell}$ in Table 2.6. A direct visual comparison to $c^*_{k,\ell}/\ell$ is given in Figure 2.6. While we belief that $\gamma_{k,\ell} > c^*_{k,\ell}/\ell$ holds for all $k, \ell \geq 2$, we cannot think of a way to prove this inequality in general.

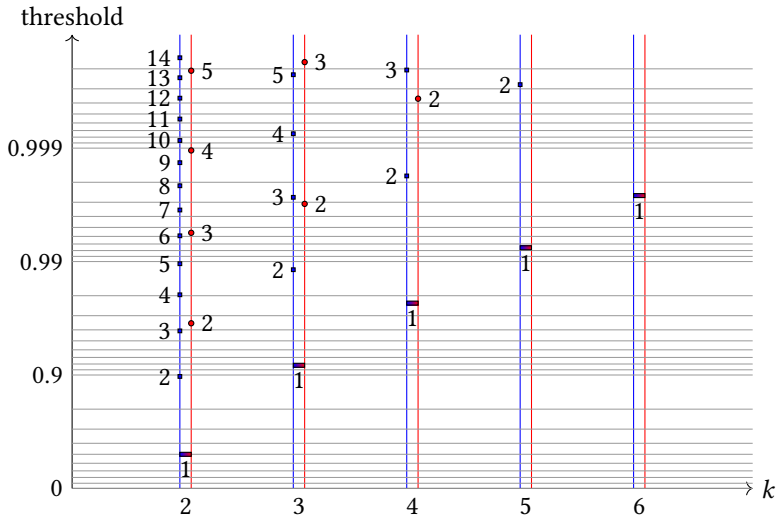


Figure 2.6 Comparison between the thresholds $c^*_{k,\ell}/\ell$ when using k aligned blocks of size ℓ (▪) and the thresholds $\gamma_{k,\ell}$ when using k unaligned blocks of size ℓ (●). The value of ℓ is given next to each plot point. Since $B^{k,1}_{n,m} \overset{\text{d}}{=} W^{k,1}_{n,m}$, there is no difference for $\ell = 1$. When using unaligned blocks for a fixed value of k, smaller block sizes ℓ suffice to achieve the same orientability threshold. See also Tables 2.4 and 2.6.

2.4.2. Less Randomness using Double Hashing

Randomness—be it "true" randomness or pseudo-random bits produced by hash functions—can be a costly resource in practice. The technique of double hashing (we give some background in Section 4.2), when applied to the fully random hypergraph $H^k_{n,m}$, yields the *double hashing hypergraph*.

▶ **Definition 2.9** (Double Hashing Hypergraph). *Let $n, m, k \in \mathbb{N}$ with $k \geq 3$.*

$$D^k_{n,m} := (\mathbb{Z}_n, \{e_1, e_2, \ldots, e_m\}), \quad \text{where } e_i = \{a_i + jb_i \bmod n \mid j \in [k]_0\},$$
$$\text{with } a_i \leftarrow\!\!\text{⚂} \ \mathbb{Z}_n, \ b_i \leftarrow\!\!\text{⚂} \ [n-1] \text{ for } i \in [m].$$

If n is prime, a nice side effect is that each hyperedge is guaranteed to have size k. For convenience, we assume this is the case.

For $k \geq 3$ and compared to $H^k_{n,cn}$, the amount of entropy (and thus "randomness") per hyperedge decreases from $\approx k \log n$ to $\approx 2 \log n$. Yet, the ℓ-orientability threshold remains the same:

▶ **Theorem E.** *For any $k \geq 3, \ell \geq 1$, the ℓ-orientability threshold of $(D^k_{n,cn})_{c \in \mathbb{R}^+, n \in \mathbb{N}}$ is $c^*_{k,\ell}$.*

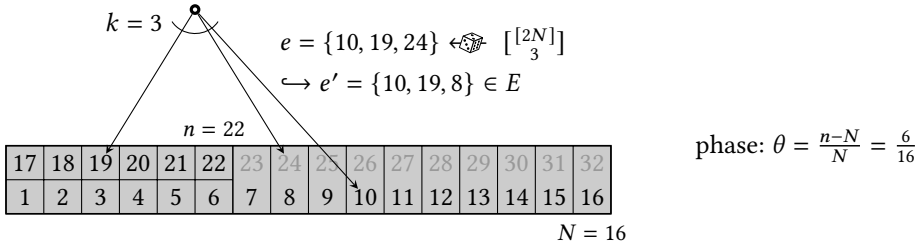


Figure 2.7 Illustration of $H^k_{n,N,m}$ for $n = 22$ and $N = 16$. The generation of one edge e' is shown. Each vertex $j \in \{7, \dots, 16\}$ is twice as likely to occur as an incidence compared to the other vertices and would be split into vertices j and $j + N$ for $n \geq j + N$.

At this point, it should be noted that previous work has come close to this theorem, but completing the argument has proven to be difficult.

Concretely, for $d \in \mathbb{N}$ let us say a hypergraph $H = (V, E)$ is *d-almost ℓ-orientable* if there is $E' \subseteq E$ of size $|E'| = |E| - d$ such that $H' = (V, E')$ is ℓ-orientable. A theorem by Leconte [Lec13] demonstrates that for $c < c^*_{k,\ell}$, the hypergraph $D^k_{n,cn}$ is $o(n)$-almost ℓ-orientable. The main difficulty in our proof of Theorem E thus concerns this pesky gap of $o(n)$ hyperedges.

2.4.3. Orientability Considerations for Dynamic Settings

To motivate the following theorem, we must hint at an application explained in Section 3.3.1, but will only do so briefly and vaguely.

Bear in mind that the hypergraph $H^k_{n,m}$ models data structures, where m is a number of objects to be stored in a table of size n. A dynamic data structure must allow for n and m to change. If the corresponding operations are to be efficient, we should see correspondingly simple and local changes in our model as n and m changes. For changes in m this is clearly the case: If $H^k_{n,m}$ and $e \leftarrow \binom{[n]}{k}$ are independent, then $H^k_{n,m} + e \stackrel{\mathrm{d}}{=} H^k_{n,m+1}$ where "+" denotes adding e to the hypergraph and "$\stackrel{\mathrm{d}}{=}$" asserts equality in distribution. The relationship between $H^k_{n+1,m}$ and $H^k_{n,m}$ is not as simple. For the data structure, this might mean that increasing the table size by 1 prompts an unacceptably expensive rebuild of the data structure. Consider therefore the following construction implicit in [MS17], and illustrated in Figure 2.7.

▶ **Definition 2.10** (DySECT Hypergraph). *Let $k, n, N, m \in \mathbb{N}$ with $N \leq n \leq 2N$.*

$H^k_{n,N,m} := ([n], E)$, *with $|E| = m$ and each $e' \in E$ chosen i.i.d. as follows:*

$$e' = \{f(j) \mid j \in e\} \text{ where } e \leftarrow \binom{[2N]}{k} \text{ and } f(j) = \begin{cases} j - N & j > n \\ j & \text{otherwise.} \end{cases}$$

It is easy to see that $H^k_{N,N,m} \stackrel{\mathrm{d}}{=} H^k_{N,m}$ and $H^k_{2N,N,m} \stackrel{\mathrm{d}}{=} H^k_{2N,m}$. Definition 2.10 therefore generalises fully random hypergraphs. Moreover, if $N < n \leq 2N$ and $H^k_{n,N,m}/[n \equiv n - N]$ denotes the hypergraph obtained by merging vertices n and $n - N$ in $H^k_{n,N,m}$, then $H^k_{n,N,m}/[n \equiv n - N] \stackrel{\mathrm{d}}{=} H^k_{n-1,N,m}$. The reverse step can be described as a randomised de-contraction. Therefore, a step by step transformation from $H^k_{N,m}$ to $H^k_{2N,m}$ is possible. Each step is a simple operation and adds one vertex.

To analyse a corresponding hashing scheme that dynamically scales the table size, it is important to get a grip on corresponding thresholds. The parameter $\theta = \frac{n-N}{N} \in [0,1]$ will turn out to be useful and we call it the *phase*.

▶ **Theorem F.** *Let $k, \ell \in \mathbb{N}$ with $k + \ell \geq 4$ and $\theta \in [0,1]$. The ℓ-orientability threshold $\eta_{k,\ell,\theta}$ of $(H^k_{n,n/(1+\theta),cn})_{c\in\mathbb{R}^+,n\in\mathbb{N}}$ is characterised by Equation (11.3) in Chapter 11.*

Figure 3.4 depicts the *load thresholds* $\eta_{k,\ell,\theta}/\ell \in [0,1]$ that capture the memory efficiency of the hashing scheme.

2.4.4. Previous Work on ℓ-orientability Thresholds

Table 2.2 lists work on thresholds relating to hypergraphs relevant for this thesis. There are three groups: The first group deals with ordinary orientability questions on fully random hypergraphs. The second generalises the notion of orientability. The third relates to different random hypergraphs.

	reference	hypergraph	threshold type	symbol
classic	[ER60; PR04]	$H^2_{n,cn}$	1-orientability	$c^*_{2,1}$
	[Die+10; FM12; FP12]	$H^k_{n,cn}$	1-orientability for $k \geq 3$	$c^*_{k,1}$
	[FR07; CSW07]	$H^2_{n,cn}$	ℓ-orientability for $\ell \geq 2$	$c^*_{2,\ell}$
	[FKP16]	$H^k_{n,cn}$	ℓ-orientability for $k \geq 3, \ell \geq 2$	$c^*_{k,\ell}$
generalised orientability	[GW10; GW15]	$H^k_{n,cn}$	(ℓ,k')-orientability for $\ell \geq \ell_0(k,k')$	
	[Lel12]	$H^k_{n,cn}$	(ℓ,k')-orientability for $(\ell,k') \neq (1,k-1)$	
	[LP14]	$H^k_{n,cn}$	$(1,k-1)$-orientability for $k \geq 3$	
	[LLM13]	$H^k_{n,cn}$	(ℓ,k',r)-orientability for $(\ell,k') \neq (r,(k-1)r)$	
other hypergraphs	[LP09]	$W^{2,\ell}_{n,cn}$	1-orientability for $\ell \geq 2$ (asymptotic lower bound)	$\gamma_{2,\ell}$
	Theorem D	$W^{k,\ell}_{n,cn}$	1-orientability for $k, \ell \geq 2$	$\gamma_{k,\ell}$
	[Lec13]	$D^k_{n,cn}$	$o(n)$-almost ℓ-orientability for $k \geq 3, \ell \geq 1$	$c^*_{k,\ell}$
	Theorem E	$D^k_{n,cn}$	ℓ-orientability for $k \geq 3, \ell \geq 1$	$c^*_{k,\ell}$
	Theorem F	$H^k_{n,n/(1+\theta),cn}$	ℓ-orientability for $\theta \in [0,1], k \geq 2, \ell \geq 1, (k,\ell) \neq (2,1)$	$\eta_{k,\ell,\theta}$

Table 2.2 Known orientability thresholds of various hypergraphs and notions of orientability.

Classic Results. The 1-orientability threshold for fully random graphs, i.e. $c^*_{2,1}$ is already implicit in classical results on the emergence of giant components due to Erdős and Renyi [ER60]. Precise values of $c^*_{k,1}$ for $k \geq 3$ were derived independently by three groups [Die+10; FM12; FP10] and values for $c^*_{2,\ell}$ for $\ell \geq 2$ independently by [FR07; CSW07]. The general case, i.e. the remaining thresholds $c^*_{k,\ell}$ for any $k \geq 3, \ell \geq 2$ are due to [FKP16].

Generalised Orientability. Consider the following generalisation: For a k-uniform hypergraph $H = (V,E)$ and $\ell \geq 1, 1 \leq k' < k, r \leq \min\{k',\ell\}$, an (ℓ,k',r)-orientation is a multi-set σ containing pairs from $E \times V$. Each $v \in V$ may occur in at most ℓ pairs while each $e \in E$ must occur in exactly k' pairs. The multiplicity of the pairs in σ must not exceed r. If $r = 1$, then $\sigma \subseteq E \times V$ is a set and we call it an (ℓ,k')-orientation. If $r = k' = 1$ then $\sigma\colon E \to V$ is a function and an ℓ-orientation in the ordinary sense. As seen in Table 2.2, almost all corresponding thresholds have by now been determined.

The techniques in the cited papers are highly heterogeneous and often specific to the cases at hand. A notable exception is an approach by Lelarge [Lel12; LLM13] using techniques from statistical physics that seem to grasp more directly at the core phenomena. Not only are almost all cases relating to $H^k_{n,cn}$ handled with one unified approach, the main theorem in [LLM13] is much more general still, dealing with a wide range of hypergraph families that have certain Benjamini-Schramm limits [BS11]. The technique is a crucial ingredient for proving Theorems D, E and F and is discussed in detail in Chapter 5.

Other Hypergraphs. Lehman and Panigrahy [LP09] analysed $B^{2,\ell}_{n,cn}$ and $W^{2,\ell}_{n,cn}$ showing that, asymptotically, the 1-orientability thresholds are $1-(2/e+o_\ell(1))^\ell$ and $1-(1/e+o_\ell(1))^{1.59\ell}$, respectively, with no implication for constant ℓ. Beyer [Bey12] showed in his master's thesis that the orientability threshold of $W^{2,2}_{n,cn}$ satisfies $0.829 \leq \gamma_{2,2} \leq 0.981$. As already mentioned in Section 2.4.2, the ℓ-orientability thresholds for $D^k_{n,cn}$ were already considered in [Lec13], but the proof remained incomplete.

Our Theorem F is a highly specialised result for Dynamic Space Efficient Cuckoo Tables (see Section 3.3.1) with no predecessors in the literature.

$\ell\backslash k$	2	3	4	5	6	7
1	0.5	0.9179352767	0.9767701649	0.9924383913	0.9973795528	0.9990637588
2	1.7940237365	1.9764028279	1.9964829679	1.9994487201	1.9999137473	1.9999866878
3	2.8774628058	2.9918572178	2.9993854302	2.9999554360	2.9999969384	2.9999997987
4	3.9214790971	3.9970126256	3.9998882644	3.9999962949	3.9999998884	3.9999999969
5	4.9477568093	4.9988732941	4.9999793407	4.9999996871	4.9999999959	≈ 5
6	5.9644362395	5.9995688805	5.9999961417	5.9999999733	5.9999999998	≈ 6

Table 2.3 The ℓ-orientability thresholds $c^*_{k,\ell}$ of the fully random k-uniform hypergraphs $(H^k_{n,cn})_{c\in\mathbb{R}^+_0,n\in\mathbb{N}}$, as obtained by [PR04; CSW07; FR07; Die+10; FM12; FP12; FKP16].

$\ell\backslash k$	2	3	4	5	6	7
1	0.5	0.9179352767	0.9767701649	0.9924383913	0.9973795528	0.9990637588
2	0.8970118682	0.9882014140	0.9982414840	0.9997243601	0.9999568737	0.9999933439
3	0.9591542686	0.9972857393	0.9997951434	0.9999851453	0.9999989795	0.9999999329
4	0.9803697743	0.9992531564	0.9999720661	0.9999990737	0.9999999721	0.9999999992
5	0.9895513619	0.9997746588	0.9999958681	0.9999999374	0.9999999992	≈ 1
6	0.9940727066	0.9999281468	0.9999993570	0.9999999956	≈ 1	≈ 1

Table 2.4 Load thresholds $c^*_{k,\ell}/\ell$ for k-ary cuckoo hashing with buckets of size ℓ, as obtained by [PR04; CSW07; FR07; Die+10; FM12; FP12; FKP16].

$\ell\backslash k$	2	3	4	5	6	7
1	0	0.8184691608	0.7722798398	0.7017802665	0.6370811273	0.5817751770
2	1.6754594358	1.5528299396	1.3336365241	1.1577691116	1.0216304657	0.9145757127
3	2.5747013735	2.1744920259	1.8108662162	1.5456982483	1.3487892629	1.1976611907
4	3.3996377443	2.7467258764	2.2497675711	1.9021610989	1.6491970269	1.4574480793
5	4.1826703850	3.2893534383	2.6654433095	2.2394560137	1.9332573430	1.7029657970
6	4.9376453624	3.8116594555	3.0650876889	2.5634847034	2.2059844373	1.9385792322

Table 2.5 The ℓ-peelability thresholds $c^{\triangle}_{k,\ell}$ of the fully random k-uniform hypergraphs $(H^k_{n,cn})_{c\in\mathbb{R}^+_0,n\in\mathbb{N}}$, see [Mol05; Coo04; PSW96; JL07] and [MM09, Chapter 18].

$\ell\backslash k$	2	3	4	5	6	7
1	0.5	0.9179352767	0.9767701649	0.9924383913	0.9973795528	0.9990637588
2	0.9649949234	0.9968991072	0.9996335076	0.9999529036	0.9999937602	0.9999991631
3	0.9944227538	0.9998255112	0.9999928198	0.9999996722	0.9999999843	0.9999999992
4	0.9989515932	0.9999896830	0.9999998577	0.9999999977	≈ 1	≈ 1
5	0.9997922174	0.9999993863	0.9999999972	≈ 1	≈ 1	≈ 1
6	0.9999581007	0.9999999635	0.9999999999	≈ 1	≈ 1	≈ 1

Table 2.6 The 1-orientability thresholds $\gamma_{k,\ell}$ for $(W^{k,\ell}_{n,cn})_{c\in\mathbb{R}^+,n\in\mathbb{N}}$ from Definition 2.8, or equivalently, load thresholds for k-ary cuckoo hashing with unaligned blocks of size ℓ, see Theorem D1. The row for $\ell = 1$ corresponds to plain k-ary cuckoo hashing, reproduced for comparison.

3. Hashing-Based Data Structures: Cuckoo Tables, Retrieval and Beyond

This chapter concerns variations of dictionary data structures. We explain their specification as well as general implementation strategies that use hashing and the "power of multiple choices" paradigm as an essential component. None of the general strategies is new. However, they have flexibilities in certain details, in particular the distribution of underlying hash functions. Depending on these, different families of hypergraphs model the behaviour of the data structure. The (likely) properties of the hypergraph then imply (likely) performance characteristics of the data structure operations. Using such connections, we translate our Theorems A to F from Chapter 2 about improved hypergraph constructions into theorems about improved data structures.

An overview is given in Figure 3.1. Note that only hashing-based data structures *relevant for this thesis* are shown, and many connections leading "outside" of the diagram are omitted. We say some words about the bigger picture in Section 4.6.

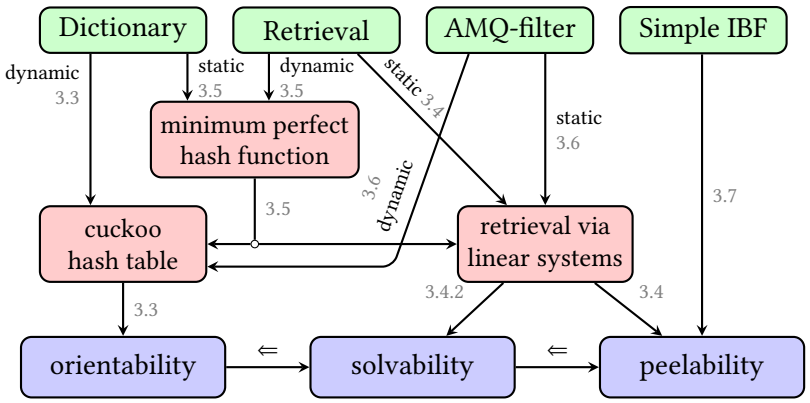


Figure 3.1 At the top, we list variations of dictionary data structures, some of which come in "static" and "dynamic" versions (see Section 3.1.2). At the bottom, we list properties that the hypergraphs we constructed in Chapter 2 are designed to have. An arrow A ⟶ B indicates a reduction, meaning A can be built if B is available. Corresponding section numbers are given in grey.

3.1. The Dictionary and its Modest Siblings

A dictionary, sometimes called associative array, map or symbol table, is a fundamental data type. There are many different ways of explaining what it is (see e.g. [San+19, Chapter 4] [Cor+09, Chapter 11]). We choose to give a mathematical model as follows.

3.1.1. The Ideal Dictionary

Let $\mathcal{U}$ be a set called *universe* with elements called *keys* and R a set called *range*. A state of an ideal dictionary is a function $f : S \to R$ where $S \subseteq \mathcal{U}$ is finite. As is usual, we often treat $f \subseteq S \times R$ as a set of pairs. We now list the operations supported by a dictionary, starting with its creation.

construct($f : S \to R$): Creates a dictionary with state f.

Some implementations of **construct** may require additional arguments that determine how the dictionary should be configured. Others may not have f as a parameter and initialise the dictionary to the empty state. The following query operations return information about f.

member($x \in \mathcal{U}$):	Returns 1 if $x \in S$ and 0 if $x \notin S$.
lookup($x \in \mathcal{U}$):	Returns $f(x)$ if $x \in S$ and $\perp$ if $x \notin S$.
eval($x \in \mathcal{U}$):	Returns $f(x)$ if $x \in S$ and arbitrary $r \in R$ if $x \notin S$.
listMembers():	Returns f as a list of pairs $((x, f(x))_{x \in S}$.

Finally, the following operations modify the state f, yielding a new state $f' : S' \to R$.

insert($x \in \mathcal{U} \setminus S, v \in R$):	Yields $f' = f \cup \{(x, v)\}$.
delete($x \in S$):	Yields $f' = f \setminus \{(x, f(x)\}$.
update($x \in S, v \in R$):	Yields $f' = (f \setminus \{(x, f(x))\}) \cup \{(x, v)\}$.

We make the following important remarks:

- The set of operations is intentionally *not minimal*. For instance, **lookup** is a combination of **member** and **eval**, and **update** can be built from **delete** and **insert**. The larger operation set is useful for discussing data structures that only support a strict subset of the dictionary operations, see below.
- Some operations may only be executed with certain parameters. For instance, an **insert** with $x \in S$ or a **delete** with $x \notin S$ has *unspecified behaviour*. Of course the desirable behaviour would be that nothing happens or that an error is reported, but we want to permit data structures that do not support **member** and cannot react so gracefully in this case.
- If R is trivial, i.e. $|R| = 1$, then we obtain a set data structure representing S. In this case the operations **eval** and **update** are meaningless.

3.1.2. Less Powerful Variations

We now define interfaces that contain a subset of the dictionary operations listed above.

Dictionary.	**construct**, **member**, **lookup**, **listMembers**, **update**, ***insert***, ***delete***.	
Retrieval.	**construct**, **eval**, ***update***.	
AMQ-Filter.	**construct**, **member***, ***insert***, ***delete***	set data structures, i.e. $\lvert R\rvert = 1$.
Simple-IBF.	**construct**, **insert**, **delete**, **listMembers***	

An operation written in italics indicates that a *dynamic implementation* supports it, while a *static implementation* does not. For instance, a static retrieval data structure does not support **update**.

An asterisk (*) indicates that a query operation of that kind may fail to output the correct result. Of course, the circumstances and probabilities for such failures will have to be discussed in detail.

To highlight the salient properties of the non-standard variations: Retrieval data structures *forget the key set* and *do not support* **member**. Approximate membership query filters (AMQ-Filters) represent sets, but **member** *may falsely report "true"* when the right answer is "false" but *not vice versa*. An invertible Bloom filter (IBF) is a set data structure that has a space utilisation not growing with S. It temporarily loses the ability to list the elements of S while it is *too full*, but *recovers* that ability when sufficiently many elements are deleted.

3.1.3. When Less Power is Sufficient

As we shall see, the less powerful dictionary variations tend to admit more efficient implementations in terms of memory usage or running time. Of course, this is only relevant if there are important applications that make do with the more restricted set of operations. We shall list a few and provide corresponding references.

Static Dictionaries. Use cases for dictionaries with a static key set S are very common. Think of any computational task that has two phases: One where a large amount of data S is acquired and a second distinct phase where information about S is (repeatedly) extracted. The data structure that stores S during the second phase need not support operations that modify S.

To give just one example, in [ZTR20] information about the human genome is stored in a large cuckoo hash table. The resulting (static) data structure can then very efficiently decide whether a sequence of base pairs is of human origin.

AMQ-Filters. In situations where memory is scarce (e.g. in cache) an AMQ-filter can replace a more memory-intensive set data structure whenever false positives are harmless or easily mitigated. In a classical example by Bloom [Blo70], an AMQ-filter D_S stores a set S of English words for which heuristics fail to produce the correct hyphenation ("hy-phen-ation"). Given a word w for which $\mathbf{member}(D_S, w)$ reports "false", it is safe to use the heuristic, and a costly access to a large data structure containing the correct hyphenation is avoided. If $w \notin S$ but $\mathbf{member}(D_S, w)$ incorrectly reports "true", a costly (but otherwise harmless) access to the data structure is made, even though the heuristic would have produced the right hyphenation.

To give a second example, assume Alice and Bob hold sets S_A and S_B and wish to compute the intersection $S_A \cap S_B$. Instead of sending S_A, Alice sends only an AMQ-filter D_{S_A} to Bob. Bob then determines $T = \{x \in S_B \mid \mathbf{member}(D_{S_A}, x)\}$ and sends T to Alice (note $S_A \cap S_B \subseteq T \subseteq S_B$) who can then determine $S_A \cap S_B$. This can significantly reduce the amount of information that needs to be transferred. For details on this and many other applications, see [BM03].

Retrieval. Retrieval data structures or "static functions" are mainly used as an auxiliary data structure within other data structures. Using them we can construct perfect hash functions, see Section 3.5, and AMQ-filters, see Section 3.6. In [Bel+10] they are applied to the *weak prefix search* problem. To take a toy example for retrieval as a stand-alone data structure from [DP08]: Assume there is a database f of n names annotated to be either boy's names or girl's names. From this, we can build a retrieval data structure D_f that correctly reproduces the gender of any name in the database and a random

output for names not in the database. The size of D_f is only $(1+\varepsilon)n$ bits for a small $\varepsilon > 0$, much less than any representation of f as a set of pairs.

Simple-IBF. The restricted versions of IBFs we introduced above have limited applicability. To borrow an example from [EG11]: Assume a security guard of a building sees every person that enters (**insert**) or exits (**delete**) throughout a day. Late at night, when almost everyone has gone home, the guard should be able to list the small number of *stragglers* that are still in the building. Surprisingly, the guard can use a data structure with a size linear in the number of stragglers, even when the peak number of people in the building is vastly greater.

In Section 3.7 we list common extensions to IBFs that widen their applicability considerably, for instance to the *set reconciliation* problem, explained there.

3.2. Modelling Hashing-Based Data Structures via Random Hypergraphs

In the standard word RAM model (see Section 4.1), we have to work with memory that presents itself as a linear sequence of memory cells or *buckets* that we index by $[n] = \{1, \dots, n\}$. We assume n to be proportional to the size m of the key set $S \subseteq \mathcal{U}$ to be stored. But in which buckets should the information on $x \in S$ be stored and how can it be located later on? In general, the universe $\mathcal{U}$ is unbounded, so clearly not every $x \in \mathcal{U}$ can have its private designated location.

There are many ways of approaching this problem, and we get back to some of them in Section 4.6. This thesis deals with approaches that exploit the "power of multiple choices" paradigm [Mit91]. In this setting, each $x \in \mathcal{U}$ is associated with *several* buckets $e(x) \subseteq [n]$ via a function $e\colon \mathcal{U} \to 2^{[n]}$ with the convention that the information about $x \in S$ should be stored exclusively in the buckets with indices from $e(x)$. For any set $S \subseteq \mathcal{U}$ the situation is modelled by the hypergraph $H = ([n], \{e(x) \mid x \in S\})$. If e is a *random* function, then H is a *random* hypergraph.

Often there is a constant number $k \geq 2$ of random functions $h_1, \dots, h_k\colon \mathcal{U} \to [n]$ and $e(x) = \{h_1(x), \dots, h_k(x)\}$. For instance, if $h_1, \dots, h_k \leftarrow\!\!\boxed{}\!\!\rightarrow [n]^{\mathcal{U}}$ are fully random and independent, then $H \stackrel{\mathrm{d}}{=} H^k_{n,m}$, i.e. H is the fully random hypergraph from Equation (2.1). The functions $h_1, \dots, h_k$ and e are called *hash functions*, indicating that we have no control over the random values they produce (though we *do* control the *distribution* of the values).

We will soon see how the properties of peelability, solvability and orientability of H from Definition 2.1 come into play.

Independence of the Hyperedges. In Chapter 2, we exclusively considered random hypergraphs with stochastically independent random hyperedges. The corresponding assumption we make now is that the family $\{e(x) \mid x \in S\}$ of sets produced by the hash function e is stochastically independent. Note that the elements of $e(x)$ for any $x \in S$ may still be correlated.

There are two immediate reasons to be worried about this independence assumption. Firstly, it is unclear how it could be satisfied by efficient, practical hash functions. We shall address this problem in Section 4.3. Secondly, one might fear that we unnecessarily narrow our focus. This second worry is readily dispelled with the following well-known argument:

Recall that we do not control the key set $S \subseteq \mathcal{U}$ and the evaluation of $e\colon \mathcal{U} \to 2^{[n]}$ needs to be efficient. If e were specifically tailored to S, it is unclear how evaluating e could be achieved without the need for another data structure relating to S (precluding that we build a fundamental data structure). Now assume e to be independent of S and consider the use case where $|\mathcal{U}| \gg |S|$ and the elements of S are chosen independently at random from $\mathcal{U}$ (we may assume repetitions do not occur). Then the hyperedges of H are stochastically independent with a distribution implicit in e. So the use case with i.i.d. random hyperedges is *unavoidable*. We simplify our job by turning *all* use cases into this case by assuming $\{e(x) \mid x \in S\}$ to be an independent family with a distribution we control.

Assumptions on the Computational Model. In following theorems, we make claims about the performance characteristics of algorithms. These only make sense in the context of a computational model. We make the following assumptions on it.

Word RAM with $O(1)$ parity. We work on a word RAM, which can perform common arithmetic operations on *words* of $w = \Theta(\log n)$ bits in $O(1)$ time, see Section 4.1. For any bit string $x = b_1 b_2 \dots b_w \in \{0,1\}^w$, the (slightly less common) $\textsc{parity}(x) = (b_1 + \dots + b_w) \bmod 2$ can also be computed in $O(1)$ time.

Keys and Values fit into Words. The elements of $\mathcal{U}$ and R can be read, copied and compared in $O(1)$ time. Otherwise, our running times are correspondingly scaled or standard mitigation techniques such as fingerprinting can be used [KR87; FAK13; Mül+14].

Fully Random Hashing is Free. We assume we can obtain for any range X a fully random function $h \leftarrow\!\!\text{\faCube}\; X^{\mathcal{U}}$ in time $\Theta(1)$ and $h(x)$ can then be evaluated in time $\Theta(1)$ for any $x \in \mathcal{U}$, by an "oracle" that only requires x and (a description of) X. If we draw several functions $h_0, h_1, h_2, \dots \leftarrow\!\!\text{\faCube}\; X^{\mathcal{U}}$ then evaluating $h_s(x)$ requires only x, X and the *seed value* s. As we explain in Section 4.3, this assumption is vindicated in practice by good pseudo-random hash functions, and theoretical techniques to weaken it exist as well.

Counting Cache Misses. Modern computers use a hierarchical memory architecture (see e.g. [HP12]) and designing algorithms to be cache efficient is important in practice. While the word RAM model is unaware of such effects, we do not want to leave aside such considerations completely. We take the (simplistic) stance that a cache miss is a discontinuity in the sequence of memory accesses, in other words, reading a sequence of k *consecutive* memory words for $k \in \mathbb{N}$ takes $\Theta(k)$ time and causes exactly 1 cache miss. Accessing auxiliary data structures, program code and temporary variables, together occupying an $o(1)$-fraction of the total memory used, never causes a cache miss.

With these general remarks out of the way, we are ready to review and revise variations of dictionary data structures, starting with cuckoo hash tables.

3.3. Dictionaries via Cuckoo Hash Tables

A *cuckoo hash table* implements a dictionary and thus represents a function $f\colon S \to R$ for a set $S \subseteq \mathcal{U}$ of size $m \in \mathbb{N}$. We use an array T with space for $n \in \mathbb{N}$ elements of $\mathcal{U}$. The space efficiency $c = \frac{m}{n} \in [0,1]$ is called the *load* of the table. Each array cell $T[i]$ for $i \in [n]$ can usually hold only one pair $(x, r) \in \mathcal{U} \times R$ and for each $x \in S$ the pair $(x, f(x))$ must be placed in $T[i_x]$ for some $i_x \in e(x) \subseteq [n]$ where $e(x)$ is generated from x via hash functions.

The distribution of $e\colon \mathcal{U} \to 2^{[n]}$ is given by the *cuckoo hashing scheme* and affects the distribution of the *cuckoo hypergraph* $H = ([n], \{e(x) \mid x \in S\})$. We say the scheme *works* if all key/value pairs can be placed, which clearly corresponds to a 1-orientation of H. Thus, the notion of orientability thresholds from Chapter 2 for hypergraph families translates into a notion of *load thresholds* of cuckoo hashing schemes. We now list the most popular schemes.

- In **standard cuckoo hashing** [PR04], we have $e(x) = \{h_1(x), h_2(x)\}$ for $h_1, h_2 \leftarrow [n]^{\mathcal{U}}$, meaning each key x is associated with two independent and uniformly random cells. The cuckoo hypergraph is $H \stackrel{\mathrm{d}}{=} H^2_{n,m}$ from Equation (2.1).
- In ***k*-ary cuckoo hashing**, due to Fotakis et al. [Fot+05], any constant number $k \geq 2$ of hash functions is used, i.e. $e(x) = \{h_1(x), \dots, h_k(x)\}$ for $h_1, \dots, h_k \leftarrow [n]^{\mathcal{U}}$ and $H \stackrel{\mathrm{d}}{=} H^k_{n,m}$.
- Dietzfelbinger and Weidling [DW07] propose **cuckoo hashing with aligned blocks of size ℓ** where the table T is partitioned into $\frac{n}{\ell}$ contiguous blocks of ℓ cells each for a constant $\ell \geq 1$. Two random blocks are assigned to each $x \in \mathcal{U}$ via two hash functions, allowing the pair $(x, f(x))$ to reside anywhere within those blocks. In our terms, $e(x) = \bigcup_{i=1,2}\{\ell h_i(x) + 1, \dots, (\ell+1)h_i(x)\}$ where $h_1, h_2 \leftarrow [n/\ell]^{\mathcal{U}}$ and the cuckoo hypergraph is $H \stackrel{\mathrm{d}}{=} B^{2,\ell}_{n,m}$ from Definition 2.7.
  An equivalent way to express the same idea assumes that every table cell of T can hold up to ℓ pairs, but reduces the number of table cells to n/ℓ to compensate. In this case we say **cuckoo hashing with buckets of size ℓ** is used. Then $e(x) = \{h_1(x), h_2(x)\}$ for $h_1, h_2 \leftarrow [n/\ell]^{\mathcal{U}}$ and a placement of all keys corresponds to an ℓ-orientation of the cuckoo hypergraph $H \stackrel{\mathrm{d}}{=} H^2_{n/\ell,m}$. [CSW07; FR07]
  The approach can be generalised in the obvious way to ***k*-ary cuckoo hashing with aligned blocks or buckets of size ℓ** with cuckoo hypergraphs $B^{k,\ell}_{n,m}$ or $H^k_{n/\ell,m}$, respectively. [FKP16; Lel12]
- By **cuckoo hashing with unaligned blocks of size ℓ** we mean a related idea—proposed in Lehman and Panigrahy [LP09] and the appendix in [DW07]—where $(x, f(x))$ may reside in any cell from $e(x) = \bigcup_{i=1,2}\{h_i(x), \dots, h_i(x) + \ell - 1\}$ where $h_1, h_2 \leftarrow [n]^{\mathcal{U}}_0$ (all indices understood modulo n). The cuckoo hypergraph is $H \stackrel{\mathrm{d}}{=} W^{k,\ell}_{n,m}$ from Definition 2.8. Note that blocks can overlap and do not form a partition of $[n]_0$ and there is no obvious corresponding concept of a "bucket".
  Again a generalisation to a "***k*-ary**" variant is immediate.
- Porat and Shalem [PS12] and independently Dietzfelbinger, Mitzenmacher and Rink [DMR11] consider variants of cuckoo hashing with $|e(x)| = k$ where memory is partitioned into **pages** and $e(x)$ is scattered over only two (or few) pages to guarantee good cache efficiency.

General implementations of the dictionary operations **construct**, **lookup**, **insert** and **delete** using cuckoo hash tables are given in Figure 3.2 (neglecting the cases with buckets of size $\ell \geq 2$). A few comments are in order.

construct. The 1-orientability threshold of the cuckoo hypergraph (or the ℓ-orientability threshold in the case of buckets of size ℓ) dictates which loads can be achieved with high probability and thus how n should be chosen given m. We summarise the known thresholds in Table 3.1 (but see also Table 2.2). For cuckoo hashing with pages, lower bounds on thresholds were given in [PS12].

```
1  Algorithm construct(f : S → R):
2      pick table size n ∈ ℕ
3      repeat
4          pick e : 𝒰 → 2^[n] // depends on scheme
5      until H([n], {e(x) | x ∈ S}) is
         1-orientable
6      σ ← 1-orientation of H
7      T ← [⊥, ⊥, ..., ⊥] // empty array of size n
8      for x ∈ S do
9          T[σ(e(x))] ← (x, f(x))
10     return (e, T)

1  Algorithm lookup(x ∈ 𝒰):
2      for i ∈ e(x) do
3          (y, r) ← T[i]
4          if y = x then
5              return r
6      return ⊥ // not found

1  Algorithm insert(x ∈ 𝒰 \ S, r ∈ R):
2      repeat // todo: graceful failure
3          for i ∈ e(x) do
4              if T[i] = ⊥ then
5                  T[i] ← (x, r)
6                  return
7          j ↚🎲 e(x)
8          swap((x, r), T[j])

1  Algorithm delete(x ∈ S):
2      for i ∈ e(x) do
3          (y, r) ← T[i]
4          if y = x then
5              T[i] ← ⊥
6              return
```

Figure 3.2 Implementations of four dictionary operations with cuckoo hash tables. For cuckoo hashing schemes using buckets, i.e. when table cells have a capacity for $\ell \geq 2$ elements, corresponding adjustments are needed.

It should be noted that the thresholds tend to predict the orientability of H for relevant finite values of n very well. For instance, experiments for $n = 10^6$ suggest that the probability for $H^3_{n,cn}$ to be 1-orientable drops from well above 99% to well below 1% when c changes from $c^*_{3,1} - 0.01$ to $c^*_{3,1} + 0.01$.

To compute an orientation in general, the bipartite matching algorithms due to Hopcroft and Karp [HK73] gives a guaranteed running time of $O(n^{3/2})$, which is not satisfactory. In the simple case where the underlying hypergraph is peelable an orientation can be obtained in linear time. In the case of 2-ary cuckoo hashing with buckets of size ℓ at a load $c < c^*_{2,\ell}/\ell$ the *selfless algorithm* [CSW07] and *excess degree reduction* [FR07] both find an orientation in linear time with high probability. For k-ary cuckoo hashing and $c = c^*_{k,1} - \varepsilon$, the *local search allocation* (LSA) algorithm [Kho13] always succeeds if an orientation exists. Its expected running time is linear in n but also depends on ε. It is plausible but not proven that the selfless algorithm and LSA algorithm give linear running times on more general classes of random hypergraphs [Rin15; Kho13; KA19].

It should be noted that some authors introduce a quality measure on orientations that they try to optimise. In cuckoo hashing with pages [DMR11] all but one element from $e(x)$ are options in the *primary page* of x. The other option is on the *backup page* and should be avoided if possible. For [ZTR20] the elements of $e(x)$ are ordered and using the i-th option incurs a cost of i. In both cases, the goal is to reduce the number of cache misses caused by (successful) lookup operations.

lookup. It is an important advantage of cuckoo hashing that the running time of **lookup** is bounded and independent of n. For k-ary cuckoo hashing with buckets of size ℓ or

cuckoo hashing scheme	H	$\frac{\#\text{cache miss}}{\text{lookup}}$	$\frac{\#\text{key comp.}}{\text{lookup}}$	**construct** in $O(n)$?	threshold
standard	$H^2_{n,cn}$	2	2	✓	$c^*_{2,1} = \frac{1}{2}$
k-ary	$H^k_{n,cn}$	k	k	✓	$c^*_{k,1}$
buckets of size ℓ	$H^2_{n/\ell,cn}$	2	2ℓ	✓	$c^*_{1,\ell}/\ell$
aligned blocks of size ℓ	$B^{2,\ell}_{n,cn}$	2	2ℓ	✓	$c^*_{1,\ell}/\ell$
k-ary, buckets of size ℓ	$H^k_{n/\ell,cn}$	k	$k\ell$	✗	$c^*_{k,\ell}/\ell$
↳ using double hashing	$D^k_{n/\ell,cn}$	k	$k\ell$	✗	$c^*_{k,\ell}/\ell$
↳ using spatial coupling	$F(\frac{n}{\ell}, k, c, z)$	k	$k\ell$	✓	$\approx c^*_{k,\ell}/\ell$
k-ary, unaligned blocks of size ℓ	$W^{k,\ell}_{n,cn}$	k	$k\ell$	✗	$\gamma_{k,\ell}$

Table 3.1 Overview of cuckoo hashing schemes and the corresponding cuckoo hypergraphs H, including results from Section 3.3.1 (grey). With respect to construction times, we treat k as constant, i.e. $O(kn) = O(n)$.

blocks of size ℓ the worst-case number of key comparisons is $k\ell$ and the worst-case number of cache misses is k. Since cache misses are normally more costly than key comparisons, one rarely sees values of k larger than 3, while $\ell = 8$ is not uncommon (see, e.g. [MS17]).

insert. Figure 3.2 shows the random walk insertion algorithm for cuckoo hash tables because it is simple, popular and explains the name *cuckoo* hashing. When a key/value pair $(x, r) \in \mathcal{U} \times R$ should be inserted and some free position $T[i] = \perp$ with $i \in e(x)$ exists, then it is used. Otherwise, (x, r) takes the place $T[j]$ for a random $j \xleftarrow{\text{🎲}} e(x)$ and *evicts* the pair that currently resides in $T[j]$, reminiscent of a cuckoo fledgling that upon hatching evicts other eggs from its nest. The happy twist compared to nature is that the evicted key/value pair will then search for a new "nest" using the same insertion algorithm. A common optimisation (omitted for clarity) is not to allow a pair to immediately go back to the position it was just evicted from.

Note that an insertion may not terminate. In practice, an implementation should fail gracefully, i.e. it should limit the number of evictions and react by picking a new hash function or by increasing the size of the table if the limit is exceeded.

Even though simulations indicate that random walk insertion can give expected constant insertion time and logarithmic insertion time with high probability (as long as the load of the table is bounded away from the threshold) [Fot+05; Mit09; FMM11], a rigorous analysis has only succeeded in special cases [FMM11; FJ19]. Note that this thesis has nothing to add in that regard.

Breadth-first search insertion that systematically searches for a shortest *augmenting path* in the cuckoo hypergraph is known to yield expected constant insertion time in a wider range of situations [Fot+05]. Some authors claim that breadth-first search insertion is less efficient than random walk insertion in practice [Mit09; FMM11], but the opposite has been observed as well [MSW19].

3.3.1. Results on Cuckoo Hash Tables

Cuckoo Hashing with Unaligned Blocks. We present the analogue of Theorem D from Section 2.4.1. Translated to the present application, it yields load thresholds for cuckoo hashing with unaligned blocks.

▶ **Theorem D1.** *Let $k, \ell \geq 2$. Consider k-ary cuckoo hashing with blocks of size ℓ. When using unaligned blocks rather than aligned blocks, the load threshold changes from $c^*_{k,\ell}/\ell$ to $\gamma_{k,\ell}$. This is an improvement at least for the values $(k, \ell) \in \{2, \dots 7\} \times \{2, \dots, 10\}$. The worst-case number of cache misses and key comparisons per lookup remain unaffected.*

Cuckoo Hashing with Double Hashing. We present the analogue of Theorem E from Section 2.4.2. Recall that k-ary cuckoo hashing with buckets of size ℓ uses k hash functions $h_1, \dots, h_k \leftarrow\!\!\boxed{?}\!\!\rightarrow [n/\ell]_0^{\mathcal{U}}$ (for a zero-based array T). A popular technique to save evaluations of hash functions for $k \geq 3$ is double hashing. We give some background in Section 4.2. Assuming $n' = n/\ell \geq 3$ is prime, we choose $h_a \leftarrow\!\!\boxed{?}\!\!\rightarrow [n']_0^{\mathcal{U}}$, $h_b \leftarrow\!\!\boxed{?}\!\!\rightarrow [n'-1]^{\mathcal{U}}$ and[1] define $h_i^{\mathrm{dh}}(x) := h_a(x) + (i-1)h_b(x) \pmod{n}'$ for $i \in [k]$ and $x \in \mathcal{U}$. With this definition, the sequence $h_1^{\mathrm{dh}}(x), \dots, h_k^{\mathrm{dh}}(x)$ is a fully random arithmetic progression in the integers modulo n' and obtained from only 2 evaluations of hash functions. Our theorem is:

▶ **Theorem E1.** *Let $k \geq 3$, $\ell \geq 1$. Consider k-ary cuckoo hashing with buckets of size ℓ. Instead of using k independent, fully random hash functions, consider using 2 hash functions and double hashing. This neither affects the load threshold, nor the worst-case number of cache misses and key comparisons per query.*

This strengthens [Lec13] and proves [MT12, Conjecture 6].

Cuckoo Hashing with Spatial Coupling. We present an analogue of Theorem A from Section 2.2. Again we consider a variation of k-ary cuckoo hashing with buckets of size ℓ, where for a parameter $z \in \mathbb{R}^+$ the hash functions $h_1, \dots, h_k$ are defined differently, namely

$$n' = \lceil n/\ell \rceil, \quad n'' = \lceil n'/(z+1) \rceil, \quad h_y \leftarrow\!\!\boxed{?}\!\!\rightarrow [n'-n'']_0^{\mathcal{U}}, \quad h_{o_1}, \dots h_{o_k} \leftarrow\!\!\boxed{?}\!\!\rightarrow [n'']_0^{\mathcal{U}} \text{ for } i \in [k],$$

and finally $h_i(x) := h_y(x) + h_{o_i}(x)$ for $i \in [k]$.

We call the resulting scheme cuckoo hashing with spatial coupling (and parameters k, ℓ, z).

▶ **Theorem A1.** *Let $k, \ell \in \mathbb{N}, k \geq 2, \ell \geq 1$, $(k, \ell) \neq (2, 1)$ and $z \in \mathbb{R}^+$. Consider k-ary cuckoo hashing with buckets of size ℓ using spatially coupling with parameter $z \in \mathbb{R}^+$. There exists an $\varepsilon = \varepsilon(z)$ with $\varepsilon(z) \xrightarrow{z\to\infty} 0$ such that the load threshold of the scheme is at most $c^*_{k,\ell}/\ell - \varepsilon$. Moreover, at loads $c < c^*_{k,\ell}/\ell - \varepsilon$ **construct** can be carried out successfully by a peeling algorithm in time $O(kn)$ whp.*

Cuckoo Hashing with Dynamic Space Efficiency By space efficiency of a dictionary data structure we mean the quotient between ideal memory usage and actual memory usage. A particular strength of cuckoo hashing schemes lies in settings where good space

[1] It is easy to see that $h_b \leftarrow\!\!\boxed{?}\!\!\rightarrow [\frac{n'-1}{2}]^{\mathcal{U}}$ is equivalent. Each arithmetic progression is then enumerated in a canonical "forward" way.

efficiency is desired, see Section 4.6. Up to this point, we have implicitly assumed that the hash table is static or that an estimate of the largest number $m_{\max}$ of elements that are simultaneously contained in the dictionary at some point during its lifetime is available beforehand. Only then can we make an informed choice for the capacity $n = \frac{m_{\max}}{c_{\max}}$ of the hash table during construction, where $c_{\max} \in [0, 1]$ is the highest load we aim for.

In other cases, the table has to grow dynamically as elements are inserted. When migrating the elements to a larger table of size $(1+\varepsilon)n$ for some $\varepsilon > 0$, the price is twofold. Firstly, if both tables exist simultaneously during the migration, then the space efficiency is temporarily at most $\frac{1}{2+\varepsilon} < \frac{1}{2}$. If the table can be extended in-place, then space efficiency is at most $\frac{1}{1+\varepsilon}$. Secondly, rescaling is a time-intensive operation if all elements are reinserted into the larger table from scratch. Since large ε sacrifices space efficiency and small ε leads to frequent rescaling, achieving the best of both worlds seems impossible.

Maier and Sanders addressed this by developing *Dynamic Space Efficient Cuckoo Tables* (DySECT) [MS17]. For some $k \geq 2, \ell \geq 1$ (e.g. $k = 3$, $\ell = 8$), they use a variation of k-ary cuckoo hashing with buckets of size ℓ. The hash table consists of $t = 2^a$ subtables for $a \in \mathbb{N}$ (e.g. $t = 256$), see Figure 3.3. During the b-th growth epoch, $b \in \mathbb{N}$, there is always some $0 \leq j < t$ such that j subtables have size 2^{b+1} while $t - j$ subtables have size 2^b. Each of the k hash functions maps elements of $\mathcal{U}$ to bit strings. The most significant a bits indicate a subtable index $r \in [t]_0$. Depending on whether $r < j$ or $r \geq j$, the next $b+1$ bits or the next b bits indicate a position in table t. To grow the hash table, the j-th subtable is doubled in size and j is increased by 1. Only the entries that reside in table j need to be reinserted by considering a previously unused bit in the relevant hash values. Each entry in table cell $0 \leq i < 2^b$ of subtable j, will thus be moved to either table cell $2i$ or $2i+1$ of the resized subtable. Once j reaches t, we reset it to 0 and growth epoch $b+1$ begins. Many more details on how this hashing scheme can be implemented efficiently are found in [MS17; MSW19], including strategies for shrinking and in-place growing of tables using the operation system's virtual memory system.

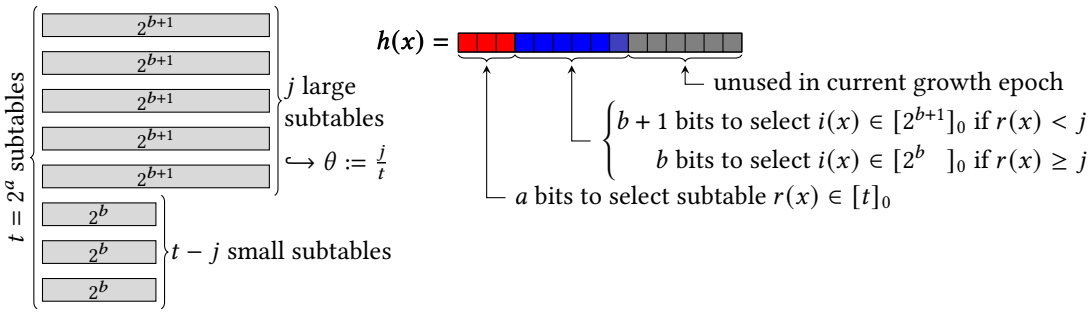


Figure 3.3 Schematic representation of how elements are hashed to table cells in a Dynamic Space Efficient Cuckoo Table (DySECT).

For $j = 0$, the hashing scheme is clearly identical to ordinary k-ary cuckoo hashing with buckets of size ℓ. For $0 < j < t$, however, hash values are uniformly distributed among tables of unequal size with non-obvious implications for the achievable loads. We let $n = 2^b(t + 2j)$ and call $\theta := j/t$ the *phase* of the growth. Then the situation with m table entries is modelled by the hypergraph $H^k_{n,n/(1+\theta),m}$ from Definition 2.10, and we now state

the analogue of Theorem F.

▶ **Theorem F1.** *Let $k, \ell \in \mathbb{N}$ with $k + \ell \geq 4$ and $\theta \in [0, 1]$. Consider the DySECT hashing scheme with k hash functions, buckets of size ℓ at growth phase θ. The load threshold $\eta_{k,\ell,\theta}/\ell$ is characterised by Equation* (11.3) *in Chapter 11.*

Experiments by Tobias Maier [MSW19] show that these load thresholds accurately predict up to what loads a DySECT hash table works for practical set and table sizes, see Figure 3.4.

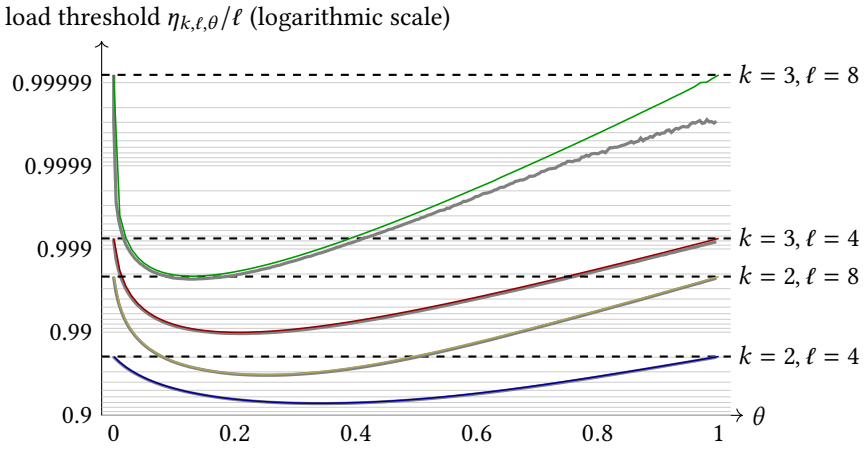


Figure 3.4 Plot of the load thresholds $\eta_{k,\ell,\theta}/\ell$ of the k-ary DySECT hashing scheme with buckets of size ℓ, for four combinations of (k, ℓ) that are useful in practice (see Section 3.3.1). The dashed lines are the corresponding load thresholds $c^*_{k,\ell}/\ell$ of ordinary k-ary cuckoo hashing with buckets of size ℓ. As expected, for phases $\theta = 0$ and $\theta = 1$ we have $\eta_{k,\ell,\theta}/\ell = c^*_{k,\ell}/\ell$. For intermediate phases $\theta \in (0, 1)$, the threshold is smaller, but still close enough to 1 for practical purposes.
The grey lines (that barely deviate from the asymptotic predictions) are experimental results obtained by Tobias Maier [MSW19] and show maximum achievable loads in (finite) DySECT tables.

3.4. Static Retrieval via Linear Systems

Recall from Section 3.1.2 that a static retrieval data structure D_f represents a function $f : S \to R$ where $S \subseteq \mathcal{U}$ is a set of m keys, but other than **construct**, only the operation **eval** needs to be supported. As we shall see, there are implementations where D_f does not store S. It is *by design* that **member** is not supported and **eval** returns unspecified[2] values from R when called with $x \notin S$. The central performance characteristics of a retrieval data structure are:

[2] Some applications may benefit from **eval**(D_f, x) yielding a *fully random* value if $x \notin S$. This behaviour can easily be retrofitted: For $R = \mathbb{F}_2^r$, pick a fully random hash function $g : \mathcal{U} \to R$ and store $\{(x, f(x) \oplus g(x)) \mid x \in S\}$ in a retrieval data structure $D_{f \oplus g}$. Then **eval**$(D_{f \oplus g}, x)$ yields $f(x)$ for $x \in S$ and a fully random value for $x \in \mathcal{U} \setminus S$.

- The space taken up by D_f. We aim for $(1+\varepsilon)rm$ bits of memory where $r = \log_2 |R|$ and $\varepsilon \geq 0$ is $O(1)$. Note that storing all pairs $(x, f(x))$ for $x \in S$ requires $\Omega(m(\log |\mathcal{U}| + r))$ bits of memory. This misses the goal in the relevant setting where an element of $\mathcal{U}$ occupies $\Omega(\log m)$ bits and $r = o(\log n)$.
- The running time of **construct**. The fastest approaches achieve the best possible time of $O(m)$, but even construction times of $O(m^3)$ are no deal breaker as we will see general compensation strategies in Section 3.4.2.
- The running time of **eval**. We aim for small constants (possibly dependent on ε) and a small number of cache misses.

The basic setup of the data structure is well known and well studied, see e.g. [DP08; GOV16; ADR09; Por09; Cha+04; BPZ13]. In the following we assume $R = \mathbb{F}_2^r$ for some $1 \leq r = O(\log m)$ where $\mathbb{F}_2 = \{0, 1\}$ is the two elements field. (This will be generalised when discussion Theorem A2.)

Assume $\varepsilon \geq 0$ and $n = (1+\varepsilon)m$. As in Section 3.2, a hash function e associates a set $e(x) \subseteq [n]$ to each $x \in \mathcal{U}$, yielding a hypergraph $H = ([n], \{e(x) \mid x \in S\})$. Assume $S = \{x_1, \ldots, x_m\}$ is arbitrarily ordered. The transposed incidence matrix of H is denoted by $A \in \mathbb{F}_2^{m \times n}$, its i-th row, for $i \in [m]$, being the incidence vector $\vec{e}(x_i) \in \mathbb{F}_2^n$ of the hyperedge $e(x_i)$, i.e. $\vec{e}(x_i)_j = 1 \Leftrightarrow j \in e(x_i)$ for $j \in [n]$.

The data structure D_f consists of a description of the hash function e and a matrix $Z \in \mathbb{F}_2^{n \times r}$ where Z satisfies $\vec{e}(x) \cdot Z = f(x)$ for all $x \in S$. If $B \in \mathbb{F}_2^{m \times r}$ is the matrix whose i-th row is $f(x_i)$ for $i \in [m]$, then we may write this compactly as $A \cdot Z = B$.

For a given $x \in \mathcal{U}$, an **eval** operation on such a data structure simply computes the vector-matrix product $\vec{e}(x) \cdot Z$ which yields $f(x)$ if $x \in S$. We give two implementations, see Figure 3.5. In $\textbf{eval}_1$ we assume that Z is stored row-wise with rows $z_1, \ldots, z_n \in \mathbb{F}_2^r$. The running time is $O(|e(x)|)$. In $\textbf{eval}_2$, we assume Z is stored column-wise with columns $z^{(1)}, \ldots, z^{(r)} \in \mathbb{F}_2^n$. The running time is then dominated by r evaluations of scalar products with $\vec{e}(x)$. Depending on r and the structure of $\vec{e}$ either approach may or may not make good use of bit parallelism. Note that $\textbf{eval}_2$ can be faster than $\textbf{eval}_1$, e.g. if $|e(x)| = \Theta(\log n)$ and scalar products with $\vec{e}(x)$ can be evaluated in $O(1)$ time.

Algorithm construct$(f\colon S \to \{0,1\}^r)$**:**
- pick $\varepsilon \geq 0$ and $n = (1+\varepsilon)|S|$
- **repeat**
  - pick $\vec{e}\colon \mathcal{U} \to \{0,1\}^n$
  - try to find solution $Z \in \{0,1\}^{n \times r}$ to $\left(\vec{e}(x) \cdot Z = f(x)\right)_{x \in S}$
- **until** *solution found*
- **return** $(\vec{e}, Z)$

Algorithm $\textbf{eval}_1(x \in \mathcal{U})$**:**
- let $e(x) = \{i \in [n] \mid \vec{e}(x)_i = 1\}$
- **return** $\bigoplus_{i \in e(x)} z_i$

Algorithm $\textbf{eval}_2(x \in \mathcal{U})$**:**
- **return** $\left(\langle \vec{e}(x), z^{(j)} \rangle\right)_{j \in [r]}$

Figure 3.5 Operations for static retrieval data structures. Details on how to efficiently solve the linear system in **construct** depend on the hashing scheme. We give two implementations of **eval**. Which is faster depends on $|e(x)|$, r and the time to evaluate a scalar product with $\vec{e}(x)$.

The crux of the matter lies in obtaining Z in **construct**. A sufficient condition for the existence of a solution Z to $A \cdot Z = B$ is that A has linearly independent rows, in other

words, that H is solvable. Otherwise, a solution exists only for some values of B.

3.4.1. Pure Results on Retrieval Data Structures

To obtain efficient retrieval data structures from the above "pure" framework (it will be augmented in Section 3.4.2), we require choices for the hash function $\vec{e}$ such that A has full rank whp even when ε is small, the system $A \cdot Z = B$ is efficiently solvable and $\mathbf{eval}_1$ or $\mathbf{eval}_2$ is fast to evaluate. This is considered in Section 2.3 and Table 2.1 , but only for the special case where $r = 1$, i.e. Z and B have only one column. For $r > 1$ a trivial option is to compute the columns $z^{(1)}, \dots, z^{(r)}$ of Z by solving the systems $Az^{(j)} = b^{(j)}$ individually for each $j \in [r]$, where $b^{(j)}$ is the j-th column of B. However, for "Gauss-like" linear system solvers (see Section 4.4), choosing $r > 1$ typically incurs *no* significant additional cost. The main cost comes from row operations on $(A \mid B)$ to bring A into echelon form and the effect of a row operation on B can be computed in $O(1)$ time for $r = O(\log m)$.

Retrieval with Two Random Blocks per Row. The following theorem follows from Theorem B **(i+ii)** and is proved in Chapter 7. Note the extremely small overhead of $\varepsilon = O(\frac{\log m}{rm})$ in **(i)**.

▶ **Theorem B1.** *We aim to build a retrieval data structure for $f : S \to \{0,1\}^r$ where $|S| = m$.*

- **(i)** *When using $A = A^L_{m,m}$ as in Theorem B **(i)** for suitable $L = O(\log m)$ in the retrieval framework from Section 3.4, then the following can be achieved:*
  - *One trial of* **construct** *takes time $O(\frac{m^3}{w \log m})$ and succeeds with probability $\Theta(1)$. The number of trials s until the first trial succeeds has expectation $\mathbb{E}[s] = O(1)$.*
  - *The data structure consists of $Z \in \{0,1\}^{m \times r}$, m and s, occupying rm bits, $O(\log m)$ bits and $O(\log s)$ bits, respectively.*
  - *An eval-operation (using $\mathbf{eval}_2$) takes time $O(r)$ and causes two cache misses when Z is stored appropriately.*
- **(ii)** *Using Wiedemann's algorithm (see Section 4.4.1), the expected construction time in **(i)** can be reduced to $\widetilde{O}(rm^2)$.*
- **(iii)** *Using $A^L_{m,n}$ as in Theorem B **(ii)** for suitable n and L instead, increases the size of D_f by $O(r \log m)$ bits. The implementation is more convenient and each trial of* **construct** *succeeds whp.*

Retrieval with One Random Block per Row. Similarly, Theorem C implies the following result, proved in Chapter 8.

▶ **Theorem C1.** *We aim to build a retrieval data structure for $f : S \to \{0,1\}^r$ where $|S| = m$.*

Let $\varepsilon > 0$. When using $A = M^L_{m,n}$ from Theorem C for suitable n and $L = O(\frac{\log m}{\varepsilon})$ in the retrieval framework from Section 3.4, then the following holds.

- **(i)** *One trial of* **construct** *succeeds whp and has expected running time $O(m/\varepsilon^2 + rm/\varepsilon)$.*
- **(ii)** *The resulting data structure occupies at most $(1+\varepsilon)rm$ bits whp.*
- **(iii)** *An eval-operation (using $\mathbf{eval}_2$) takes time $O(r/\varepsilon)$ and causes one cache miss when Z is stored appropriately.*

Retrieval Based on Peeling. The following lemma shows that we can drop the restriction that $|R|$ is a power of two (or any prime power if other fields are used) if the underlying hypergraph is peelable. We consider this folklore; for instance, it is implicit in [Maj+96]. It holds for any group R, with slight notational inconveniences for non-Abelian groups. Not much is lost by sticking to a special case.

▶ **Lemma 3.1.** *Let $R = \mathbb{Z}_d$ be the integers modulo $d \in \mathbb{N}$ and $f\colon S \to R$ for some $S \subseteq \mathcal{U}$ of size m. The basic idea from Section 3.4 can still be used to build a retrieval data structure for f if the hypergraph $H = ([n], \{e(x) \mid x \in S\})$ is 1-peelable. Construction takes time $O(\sum_{x\in S} |e(x)|)$ and yields $D_f = (e\colon \mathcal{U} \to 2^{[n]}, z \in R^n)$. An* **eval** *on D_f for $x \in S$ takes time $O(|e(x)|)$.*

Proof. Similarly as before, the idea is to find a vector $z = (z_1, \dots, z_n) \in R^n$ that satisfies the equation $\vec{e}(x) \cdot z = f(x)$ for all $x \in S$. Then eval-operations can simply evaluate the left-hand side. However, since the "unknowns" and "right-hand sides" come from the group R while the coefficients come from $\mathbb{F}_2$, we do not have a linear system in the traditional sense. For instance the "scalar product" $\vec{e}(x) \cdot z$, which we take to mean $\sum_{i\in e(x)} z_i$, is not bilinear.[3]

But since H is 1-peelable, there is a vertex i incident to only one hyperedge $e(x)$. Thus there is a variable z_i, $i \in [n]$, involved only in the equation for one $x \in S$. We can solve all other equations by induction (using that $H' = ([n] - \{i\}, E \setminus \{e(x)\})$ is 1-peelable) and then choose $z_i = f(x) - \sum_{i'\in e(x)\setminus\{i\}} z_{i'}$ which satisfies the remaining equation and only exploits the group structure of R. The claimed running times are easily verified. ◀

Given Lemma 3.1, we can turn the 1-peelable hypergraph family from Theorem A' into a retrieval data structure with corresponding parameters.

▶ **Theorem A2.** *Let $R = \mathbb{Z}_d$ for $d \in \mathbb{N}$, $k \geq 3$, $c < c^*_{k,1}$ and $\varepsilon := \frac{1-c}{c}$. When using $H = \hat{F}^{k,1}_{m/c,m}$ from Theorem A' to build a retrieval data structure for $f\colon S \to R$ with $|S| = m$ as in Lemma 3.1, then the following holds.*

- **(i)** *One trial of* **construct** *succeeds whp and has expected running time $O(km)$.*
- **(ii)** *The resulting data structure D_f needs space to store $z \in R^{(1+\varepsilon)m}$, plus $O(\log m)$ bits whp.*
- **(iii)** *An* **eval** *takes time $O(k)$ and causes k cache misses.*

Note that if d is not a power of 2, then storing $z \in \mathbb{Z}_d^n$ using $n\lceil \log d \rceil$ bits may be wasteful compared to the lower bound $\lceil n \log d \rceil$. For $d = 3$ the overhead is $\approx 26\%$. A simple trick in this case would be to store 5 values ($3^5 = 243$ possibilities) in a byte ($2^8 = 256$ possibilities) with lookup tables for decoding and encoding which reduce the overhead to $< 1\%$. Of course, such tricks work for other values of d as well.

3.4.2. Input Partitioning for Faster Construction

Assume we have a retrieval data structure where **construct** takes $t(m) = \omega(m)$ on input $f\colon S \to R$ with $|S| = m$. A common and well-explored trick [BKZ05; DR09; Por09; GOV16; GOV20; EGV20] for improving this running time is to split the function f into "chunks" of approximately C key/value pairs each, for a *desired chunk size* C chosen later.

[3] For instance $(\vec{e}(x) \oplus \vec{e}(x')) \cdot z = (\vec{e}(x) \cdot z) + (\vec{e}(x') \cdot z)$ does not hold in general.

Concretely, pick a fully random "partitioning" hash function $h_0 \leftarrow [m/C]^{\mathcal{U}}$ and obtain the chunks $f_i := \{(x, f(x)) \in f \mid h_0(x) = i\}$ for $i \in [m/C]$. For each $i \in [m/C]$ build an individual retrieval data structure D_{f_i} that stores f_i. Since the actual chunk sizes are tightly concentrated around C, the expected construction time is, under weak conditions on t, reduced from $t(m)$ to $\frac{m}{C}t(C) + O(m)$. For instance, if Gaussian elimination is used, $t(m) = \widetilde{O}(m^3)$ becomes $\widetilde{O}(mC^2)$. An **eval** of $x \in S$ simply evaluates $i = h_0(x)$ and then performs an **eval** on D_{f_i}. We consider two ways of applying this trick.

The Flexible Way. The slight fluctuations in the chunk sizes $|f_i|$ for $i \in [m/C]$ lead to corresponding fluctuations in the sizes of the data structures $D_{f_1}, \dots, D_{f_{m/C}}$, which are stored sequentially. We therefore need to store for each $i \in [m/C]$ a pointer or offset o_i indicating where D_{f_i} starts in memory. This takes $O((m/C)\log m)$ bits[4]. Moreover we need to store a seed for each chunk, indicating the hash function with which construction succeeded for that chunk. Even when a chunk's construction attempt succeeds only with constant probability, whp no chunk requires more than $O(\log m)$ attempts, so a total of $O((m/C)\log m)$ bits suffice for all seeds.

When combining Theorem B1 with this we obtain immediately:

► **Theorem B2.** *We aim to build a retrieval data structure for $f : S \to \{0,1\}^r$ where $|S| = m$.*

(i) *For $C = m^\alpha$ with $\alpha \in (0,1)$, the following performance characteristics can be achieved:*
- *Construction takes time $O(\frac{mC^2}{w \log C})$ whp.*
- *The data structure occupies at most $(1+\varepsilon)rm$ bits of memory whp, with overhead $\varepsilon = O(\frac{\log m}{C})$.*
- *An **eval** takes time $O(r)$ and causes two cache misses[5].*

(ii) *Using Wiedemann's algorithm (see Section 4.4.1), the expected construction time in **(i)** can be reduced to $\widetilde{O}(rmC)$.*

In a similar way, Theorem C1 can be strengthened. A short proof is given in Chapter 8.

► **Theorem C2.** *We aim to build a retrieval data structure for $f : S \to \{0,1\}^r$ where $|S| = m$. For $\varepsilon > 0$ and chunks of size $C = m^\varepsilon$, the following performance characteristics can be achieved.*

(i) *One trial of **construct** succeeds whp and has expected running time $O(m/\varepsilon + rm)$.*

(ii) *The resulting data structure occupies at most $(1+\varepsilon)rm$ bits whp.*

(iii) *An **eval** takes time $O(r)$ and causes one cache miss.*

The Rigid Way. Now assume the same amount of memory is allocated for each D_{f_i}, $i \in [m/C]$. In this case, no pointers are required to locate D_{f_i} in memory (the chunks are stored at regular offsets), but random fluctuations in the chunk sizes can cause chunks to "overflow". Thus, a fall-back data structure is used to accommodate "bad" pairs $(x, f(x)) \in f$ from chunks where construction failed more than a certain number of times. To ensure that only $o(m)$ pairs are bad, the space for D_{f_i} is larger than would be required for the

[4] For small values of C more compact encodings of $0 = o_1 \le \dots \le o_{m/C} = O(m)$ can be pursued. In [EGV20] a customised Elias-Fano encoding is used.

[5] Note that navigating to the retrieval data structure of an element's chunk requires accessing an array of pointers. Since this array has size $\widetilde{O}(m^{1-\alpha})$, this is not counted as a cache miss by assumptions from Section 3.2.

expected number C of key/value pairs. Roughly speaking, the standard deviation for the number of keys in a chunk is $\sqrt{C}$, so D_{f_i} should be appropriate for $C + \Omega(\sqrt{C})$ keys. An overhead of $\varepsilon = \Omega(C^{-1/2})$ is then unavoidable.

3.4.3. Comparison of Static Retrieval Data Structures

In Table 3.2, we compare the performance of various approaches to build static retrieval data structures. We identify three groups, and one approach very different from all others.

Two sources for overhead ε are distinguished. "Design overhead" comes from the hashing scheme, i.e. from considerations about when matrices are likely to have full rank. The "partition overhead" comes from using a strategy from Section 3.4.2. Note that retrieval data structures with no overhead of either kind still need $O(\log m)$ bits to store their size and a global seed value. Such "other overhead" of order $O(\frac{\log m}{m})$ is not counted here.

		Reference	t_{eval}	$\frac{\#\text{cache miss}}{\text{eval}}$	$t_{\text{construct}}$	"design overhead" + "partition overhead"
Impractical	**(1)**	[DP08]	$O(k)$	k	$O(m^3)$ or $\widetilde{O}(m^2)$	$e^{-k} + o(e^{-k}) + 0$
	(2)	[DP08; Por09]	$O(\log m)$	$\log m$	$O(m^3)$ or $\widetilde{O}(m^2)$	$0 + 0$
	(3)	Theorem B1	$O(r)$	2	$\widetilde{O}(m^3)$ or $\widetilde{O}(m^2)$	$0 + 0$
	(4)	Theorem C2	$O(r)$	1	$O(m/\varepsilon + rm)$	$\varepsilon + \Omega(\frac{\log m}{m^\varepsilon})$
	(5)	[DP08]	$O(k)$	1	$O(m)$	$e^{-k} + o(e^{-k}) + \Omega((\log m)^{-1/4})$
	(6)	[Por09]	$O(1)$	1	$O(m)$	$0 + \Omega(\frac{\log\log m}{\sqrt{\log m}})$
Peeling	**(7)**	[BPZ13; Maj+96]	$O(1)$	3	$O(m)$	$0.23 + 0$
	(8)	[Rin13]	$O(1)$	$\approx 5^*$	$O(m)$	$0.087 + 0$
	(9)	[Lub+97]	$O(k)^*$	k^*	$O(mk)$	$O(e^{-k}) + 0$
	(10)	Theorem A2	$O(k)$	k	$O(mk)$	$e^{-k} + o(e^{-k}) + 0$
Gaussian elimination	**(11)**	[ADR09]	$O(k)$	k	$O(mC^2)$ or $\widetilde{O}(mC)$	$e^{-k} + o(e^{-k}) + \Omega(C^{-1/2})$
	(12)	[GOV16]	$O(1)$	3 [or 4]	$O(\frac{mC^2}{w})$ or $\widetilde{O}(mC)$	0.09 [or 0.024] $+ \Theta(\frac{\log m}{C})$
	(13)	Theorem B2	$O(r)$	2	$O(\frac{mC^2}{w \log C})$ or $\widetilde{O}(mC)$	$\Theta(\frac{\log m}{C}) + \Theta(\frac{\log m}{C})$
	(14)	Theorem C1	$O(r/\varepsilon)$	1	$O(m/\varepsilon^2 + rm/\varepsilon)$	$\varepsilon + 0$
	(15)	[Mül+14]	$O(1)$	$O(1)$	$O(m)$	$O(1)$

Table 3.2 Comparison of various retrieval data structures. If $t_{\text{construct}}$ reports two alternatives, the second can be achieved by using Wiedemann's algorithm (see Section 4.4.1). Two sources of overhead are distinguished, see Section 3.4.3. An asterisk indicates that an expectation of a random quantity with high variance is given.

The Impractical ones. Approaches **(1)**, **(2)** and **(3)** use k random 1's per row, $(1+\delta)\log m$ random 1's per row (for $\delta > 0$) and 2 random blocks of $O(\log m)$ bits per row (c.f. the hypergraphs $H^k_{n,cn}$, $H^{(1+\delta)\log m}_{n,cn}$ and the matrix $A^L_{n,n}$ in Section 2.3.3). With no good way of solving the linear systems, they remain theoretical. Still **(2)** and **(3)** are interesting as they avoid essentially all overhead and **(1)** inspired **(5)** and **(11)**.

Approach **(5)** uses a "rigid" partitioning strategy (see Section 3.4.2) with $C = O(\sqrt{\log m})$. The resulting tiny linear systems can then be solved using lookup tables. Experiments in [ADR09] show that this does not work well in practice. Approaches **(4)** and **(6)** suffer from a similar problem. The partition overhead is simply not small in practice for reasonable values of m.

The ones that Peel. If the underlying hypergraph is peelable, then by Lemma 3.1, the linear system can be solved in linear time (in the number of incidences). We list the four peelable constructions from Section 2.2.2. Approach **(7)** from [BPZ13] was highly influential due to its simplicity and good practical performance. Approaches **(9)** and **(10)** both allow to increase k for larger **eval** times but smaller overhead. Among the two, **(9)** offers a worse trade-off for small k and has worst-case **eval** times of $O(e^k)$, much larger than the reported average **eval** times.

The ones that Gauss. In an attempt to implement **(5)**, approach **(11)** forgoes the use of lookup tables for linear system solving in favour of Gaussian elimination. On practical input sizes with $C = 100$, it beats **(5)** in terms of overhead. In **(12)**, flexible partitioning[6], bit-level parallelism and structured Gaussian elimination are combined to achieve significant practical improvements over **(11)** and **(5)** using $k \in \{3, 4\}$.

Both **(13)** and **(14)** aim for extreme cache efficiency in **eval** operations. Since they spend their "bit parallelism budget" for this, **eval** times scale with r. Consequently, they are most interesting if r is small. To foreshadow experimental results, in our implementations **(13)** achieves the lowest overhead, and a compromise between **(4)** and **(14)** achieves the fastest construction time, due to excellent cache efficiency.

Approach **(15)** is completely unrelated to linear system solving and has a wildly different focus. On the one hand, its overhead is much higher: It can barely achieve $\varepsilon < 1$ for $r = 8$ and gets significantly worse for smaller r. On the other hand, it has excellent running times, supports **update** and can even support **insert** and **delete** if f is available on (slow but abundant) background memory.

Lastly we should mention that the retrieval problem has also been studied in situations where the values $f(x)$ are drawn from a non-uniform distribution $\mathcal{D}$ on R [GOV20; HKP09]. In this case, compression techniques can be used and the lower bound on the memory requirement is no longer $m \log_2 |R|$ bits but $m \cdot H(\mathcal{D})$ bits, where $H(\mathcal{D})$ is the entropy of $\mathcal{D}$.

3.5. (Minimum) Perfect Hash Functions via Retrieval

For $S \subseteq \mathcal{U}$ of size m, a *perfect hash function* (PHF) is an injective function $p\colon S \to [m']_0$ for some $m' \geq m$. If $m' = m$, then p is a *minimum perfect hash function* (MPHF). From a PHF p for S, we immediate obtain a *dynamic* retrieval data structures for $f\colon S \to R$, i.e. one with an **update** operation, by keeping $f(x)$ in cell $T[p(x)]$ of an array $T[0 \dots m' - 1]$. Similarly, when storing $(x, f(x))$ in $T[p(x)]$ we obtain a static dictionary, i.e. one without **insert** and **delete**. In both cases, the space requirement is minimal except for the $m' - m$ unused table cells (if p is not perfect) and the space for storing p.

There is a huge body of work on perfect hashing [FKS84; CHM92; Maj+96; CHM97; HT01; BPZ07; Bot08; BBD09; BPZ13; Bel+14; GOV16; Lim+17; EGV20]. For an overview see [Rin15] or [Die07], and for an up-to-date comparison of practical techniques, see [EGV20].

Some approaches rely on peelable or solvable hypergraphs [CHM92; Maj+96; BPZ13; Bel+14; GOV16]. Plugging our spatially coupled hypergraphs into the framework by [Bot08], described below, yields:

[6] Technically, this requires admitting that $C = \Omega(\log m)$. Construction time is therefore super-linear.

▶ **Theorem A3.** *For any $\alpha > 2/c^*_{3,1} \approx 2.18$ a simple algorithm computes for $S \subseteq \mathcal{U}$ of size m a* MPHF $p\colon S \to [m]$ *occupying $\alpha m + o(m)$ bits of memory in expected time $O(m)$. Evaluating p takes $O(1)$ time and causes 3 cache misses.*

For comparison, note that $\alpha = \log_2 e \approx 1.44$ is a known lower bound [FKS84], achieved only by theoretical constructions [HT01]. The currently best practical algorithm achieves $\alpha = 1.8$ comfortably and $\alpha = 1.56$ with high sacrifices in running time [EGV20].

While we are not breaking new ground in terms of space efficiency, we suspect that *among the known constructions that achieve $\alpha = 2.2$ or better*, ours is the simplest and possibly the fastest.

The BPZ Framework [BPZ13]. To build a MPHF p for a set $S \subseteq \mathcal{U}$ of size m, follow these steps:

Step 1. Pick a constant $k \in \mathbb{N}$, a suitable number $m' \geq m$, with $m' = \Theta(m)$ and hash functions $h_0, \dots, h_{k-1}\colon \mathcal{U} \to [m']$ such that for $e(x) := \{h_i(x) \mid i \in [k]_0\}$ the hypergraph $H = ([m']_0, E := \{e(x) \mid x \in S\})$ is 1-orientable whp. Let $\sigma\colon E \to [m']_0$ be a 1-orientation and let $\varphi\colon S \to [k]_0$ describe which option is realised for each key, i.e. $\sigma(e(x)) = h_{\varphi(x)}(x)$ for $x \in S$.

Step 2. Build a retrieval data structure D_φ for φ. Note that $p'\colon S \to [m']_0$ with $p'(x) = h_{\varphi(x)}(x)$ is a PHF and $\Theta(m \log(k))$ bits suffice to store D_φ.

Step 3. Let $\mathrm{rank}(i) := |p'^{-1}(\{0, \dots, i-1\})|$ for $i \in [m']$ be the number of "used" hash values less than i. Use standard techniques for building a *rank data structure* D_{rank} that occupies $m' + o(m')$ bits and allows to evaluate rank in time $O(1)$, see e.g. [Pag01; OS07; RRS07; Vig08; FPS16]. Now $p\colon S \to [m]_0$ with $p(x) := \mathrm{rank}(p'(x))$ is a MPHF and given by $h_0, \dots, h_{k-1}, D_\varphi$ and D_{rank}.

Proof of Theorem A3. The authors of [BPZ13] present an instantiation of the above framework that allows for clever optimisations. We follow their idea but have substituted the underlying hypergraph for one with a higher peelability threshold. Pseudo-code is given in Figure 3.6. The memory requirement is $\frac{2(z+1)}{zc} m + o(m)$ bits where z and c are chosen in line 2. For $z \to \infty$ and $c \to c^*_{3,1}$ this achieves the promised result.

In lines 3-7 we generate a hypergraph $H \stackrel{\mathrm{d}}{=} F(m', 3, c, z)$ as in Definition 2.4. It contains m' vertices and is 1-peelable whp by Theorem A. Thus, a suitable $\varphi\colon S \to \{0, 1, 2\}$ for **Step 1** can be found in linear time (lines 8–13). For **Step 2**, we base the retrieval data structure D_φ on *the same* hypergraph H and obtain, conceptually, a vector $z \in \mathbb{F}_3^{m'}$ with $(\sum_{i \in e(x)} z[i]) \bmod 3 = \varphi(x)$ for all $x \in S$. However, we use two bits for the entries of z and initialise them to 3. Lines 15–16 overwrite an entry $z[i]$ with a value from $\{0, 1, 2\}$ if $i = h_{\varphi(x)}(x)$ for some $x \in S$ and $z[i]$ remains 3 otherwise. We therefore have $\mathrm{rank}(i) = \#\{i' \in [m']_0 \mid i' < i \wedge z[i'] \neq 3\}$ in **Step 3**. As it turns out, this makes it possible to reuse z as a component in D_{rank} and only $o(m')$ additional bits are needed. All in all, z requires $2m'$ bits of memory while h_0, h_1, h_2 and D_{rank} require $o(m)$ bits of memory. ◀

Algorithm constructMPHF($S \subseteq \mathcal{U}$):
- pick $c < c^*_{3,1}$ and $z \in \mathbb{R}^+$ // e.g. $z = 100$, $c = 0.9$ for $|S| = 10^7$
- let $m = |S|$, $m' = \lfloor \frac{z+1}{zc} m \rfloor$, $N = \lfloor \frac{m'}{z+1} \rfloor$
- $h \xleftarrow{\$} [m' - N]_0^{\mathcal{U}}$, $o_0, o_1, o_2 \xleftarrow{\$} [N]_0^{\mathcal{U}}$
- let $h_\varphi(x) = h(x) + o_\varphi(x)$, for $\varphi \in \{0,1,2\}$ and $x \in S$
- let $e(x) = \{h_\varphi(x) \mid \varphi \in \{0,1,2\}\}$, for $x \in S$
- let $H = ([m']_0, \{e(x) \mid x \in S\})$ // 1-peelable whp, assume this is the case
- stack $\leftarrow \emptyset$
- **while** *H is not empty* **do**
  - pick $i \in [m']_0$ with $\deg_H(i) = 1$
  - identify $x \in S$ and $\varphi \in \{0,1,2\}$ with $h_\varphi(x) = i$
  - remove $e(x)$ from H
  - push (x, φ) to stack
- $z \leftarrow [3, 3, \ldots, 3] \in \{0,1,2,3\}^{m'}$ // $2m'$ bits
- **for** $(x, \varphi) \in$ *stack* **do** // reversed order
  - $z[h_\varphi(x)] \leftarrow (\varphi - \sum_{\varphi' \in \{0,1,2\} \setminus \{\varphi\}} z[h_{\varphi'}(x)]) \bmod 3$
- let $\mathrm{rank}(i) := |\{i' < i \mid z[i'] \neq 3\}|$ for $i \in [m']$
- build succinct data structure D_{rank} for rank-queries // takes $o(m)$ bits, given z
- **return** $(h_0, h_1, h_2, z, D_{\mathrm{rank}})$

todo: to reduce the memory footprint and running time of peeling use:
- xor-trick [Bel+14, Section 4.1]
- single-stack-peeling [GOV20, Section 6.3]

Algorithm evaluateMPHF($x \in S$):
- $\varphi \leftarrow (\sum_{\varphi' \in \{0,1,2\}} z[h_{\varphi'}(x)]) \bmod 3$
- **return** $\mathrm{rank}(h_\varphi(x))$

Figure 3.6 A simple algorithm to compute, for a set $S \subseteq \mathcal{U}$ of size m, a minimum perfect hash functions $p\colon S \to [m]_0$. For $z \to \infty$ and $c \to c^*_{3,1}$ the number of required bits per element approaches $2/c^*_{3,1} \approx 2.18$ bits.

3.6. Approximate Membership via Retrieval and Cuckoo Tables

Recall that an AMQ-Filter represents a set $S \subseteq \mathcal{U}$ of size m. It supports **member**-queries that may erroneously return "true" for $x \notin S$ with probability $\varepsilon \in [0, 1)$, called the *false positive rate*. The operations **insert** and **delete** may or may not be support. We briefly mention three types of AMQ-Filters, compared in Table 3.3, two of which are intimately related to the subject of this thesis.

Bloom Filter. The elephant in the room is the venerable Bloom filter [Blo70]. We will *not* explain it here. Instead, we refer to a self-contained introduction including standard tricks and applications [BM03], as well as to an up to date survey covering countless Bloom filter variations [Luo+19].
We only mention that standard Bloom filters require at least $\log_2(e)\log_2(1/\varepsilon) \approx 1.44\log_2(1/\varepsilon)$ bits per element of S and that **insert** can be supported while **delete** cannot.

xor Filter. The following construction was introduced and analysed in [CC08; DP08]. In

Filter Type	$\frac{\text{bits}}{\text{element}}$	$\frac{\#\text{cache misses}}{\text{query}}$	**insert**	**delete**	builds on
Bloom Filter	$\geq 1.44 \log 1/\varepsilon$	$\log 1/\varepsilon$	✓	✗	–
XOR Filter	$\geq \log 1/\varepsilon$	$O(1)$	✗	✗	retrieval data structure
Cuckoo Filter	$\geq \log 1/\varepsilon$	2	✓	✓	cuckoo hash table

Table 3.3 Comparison of basic approximate membership data structures in terms of (tight) space lower bounds, number of cache-misses per **member**-query and supported dynamic operations.

[GL19] it was appropriately dubbed *XOR filter*.

To represent S in a XOR filter, we pick a "fingerprint" function $f : \mathcal{U} \to \{0,1\}^r$ uniformly at random where $r = \log_2 1/\varepsilon$. Let $f_S : S \to \{0,1\}^r$ denote the restriction of f to domain S. The filter is given by f and a retrieval data structure D_{f_S} for f_S. As usual, we assume that the hash function f requires essentially no space to store. A **member** query for $x \in \mathcal{U}$ simply performs an **eval** of x on D_{f_S} and returns true if the result equals $f(x)$. For $x \in S$ this is clearly the correct result. For $x \notin S$, note that (x, D_{f_S}) is stochastically independent of the fully random value $f(x) \in \{0,1\}^r$, so $\Pr[\textbf{member}(x) = \text{true}] = \Pr[\textbf{eval}(D_{f_S}, x) = f(x)] = 1/2^r = \varepsilon$.

The XOR filter directly inherits the performance characteristics from the underlying retrieval data structure, see Table 3.2, some constructions taking little more than mr bits of space. Note that **insert** and **delete** cannot be supported. For an experimental evaluation for particular choices of the retrieval data structure, highlighting advantages over cuckoo filters, see [GL19]. See also [WRS18].

Cuckoo Filter. Cuckoo Filters were introduced in [FAK13] and have since gained widespread popularity.

Again, a fully random fingerprint function $f : \mathcal{U} \to \{0,1\}^r$ is used, for $r = \log 1/\varepsilon$ and $f_S : S \to \{0,1\}^r$ denotes the restriction from f to domain S. To store f_S in a (2-ary) cuckoo *dictionary* with buckets of size $\ell \in \mathbb{N}$, we would pick $h_1, h_2 : \mathcal{U} \to [n/\ell]^{\mathcal{U}}$ and store $(x, f_S(x))$ in bucket $h_1(x)$ or $h_2(x)$. To build a cuckoo *filter* we do just that, *except* we only store fingerprints without keys. As should be expected, a **member** query for $x \in \mathcal{U}$ returns true if the buckets $h_1(x)$ and $h_2(x)$ contain at least one copy of $f(x)$. As ℓ increases, the hash table admits loads close to 1, i.e. requires little more than r bits per element.

Since cuckoo filters support **insert** (and **delete**), there is, however, a major complication. When evicting a key $x \in S$ from its bucket $h_i(x)$ ($i \in \{1,2\}$), its alternative bucket $h_{2-i}(x)$ must now be computed from $f(x)$ and $h_i(x)$ alone, as x itself is not stored. In [Fan+14] this is solved by defining $h_2(x) := h_1(x) \oplus h(f)$ where $h : \{0,1\}^r \to [n/\ell]$ is a hash function on the fingerprints and $[n/\ell]$ is assumed to be a power of 2. Note that $h_1(x) = h_2(x) \oplus h(f)$ holds as well. Unfortunately, this trick breaks the existing analysis of cuckoo hashing. Empirically, cuckoo filters work well and a simplified version has been successfully analysed [Epp16].

3.7. Straggler Identification via Invertible Bloom Filters

Assume the universe $\mathcal{U} = (\{0,1\}^w, \oplus)$ is a set of bit strings with bitwise XOR. The *straggler identification problem* [EG11] asks for a set data structure that can, after N insertions and $N-m$ deletions of previously inserted elements, identify the m *stragglers* that were inserted but not deleted. The interesting case is when only $O(m)$ memory words are used at all times, despite the possibility for intermediate sets of size $O(N)$.

Adapting the presentation in [GM11; EG11; Epp+11], an invertible Bloom filter (IBF) uses a hash table of size $n = (1+\varepsilon)m$ and a hash function $e\colon \mathcal{U} \to 2^{[n]}$ of the form $e(x) = \{h_1(x), \dots, h_k(x)\}$. To store $S \subseteq \mathcal{U}$ in the table, each $x \in S$ is stored in *all* table cells $i \in e(x)$, however, if $S_i \subseteq S$ is stored in cell i then only $\bigoplus_{x \in S_i} x$ and the *count* $|S_i|$ are represented. In particular, if $|S_i| > 1$, then it is impossible to reconstruct S_i from the information in cell i alone.

Pseudo-code for constructing an empty IBF, insertion, deletion and listing all element (in a destructive variant) is provided in Figure 3.7. The **listMembers** operation repeatedly identifies table cells with counts of 1 and subsequently deletes the identified element and has an $O(kn)$ time implementation. It succeeds if and only if the underlying hypergraph $H = ([n], \{e(x) \mid x \in S\})$ is 1-peelable. For instance, if $k = 3$, $m/n = c < c^{\triangle}_{3,1} \approx 0.81$, and $h_1, h_2, h_3 \leftarrow\!\!\circledast [n]^{\mathcal{U}}$, then this is the case whp.

```
1 Algorithm construct():
2     allocate table T of size n
3     pick e: U → 2^[n]
4     for i ∈ [n] do
5         T[i].val ← 0^w
6         T[i].count ← 0
7     return (e, T)

1 Algorithm insert(x ∈ U):
2     for i ∈ e(x) do
3         T[i].val ← T[i].val ⊕ x
4         T[i].count ← T[i].count + 1

1 Algorithm delete(x ∈ U):
2     for i ∈ e(x) do
3         T[i].val ← T[i].val ⊕ x
4         T[i].count ← T[i].count − 1

1 Algorithm listMembers():
2     D ← ∅
3     while ∃i ∈ [n] : T[i].count = 1 do
4         D ← D ∪ {T[i].val}
5         delete(T[i].val)
6     if ∃i ∈ [n] : T[i].count > 0 then
7         return FAILURE
8     else
9         return D
```

Figure 3.7 Operations of a simple invertible Bloom filter (IBF). Slightly deviating from Section 3.1.2, **construct** can only produce empty IBFs, and **listMembers** is destructive.

By substituting the underlying hypergraph for our spatially coupled construction from Theorem A', we obtain a slight improvement:

▶ **Theorem A4.** *In the above setting, let $k \in \mathbb{N}$ and $c < c^*_{k,1}$. If $h_1, \dots, h_k$ are such that $H \overset{d}{=} \hat{F}^{k,1}_{n,cn}$, then* **listMembers** *succeeds in time $O(kn)$ whp.*

The IBFs in [EG11; Epp+11] and the invertible Bloom *lookup table* (IBLT) in [GM11] enrich the above construction as follows. There is no interaction with the concrete hypergraph choice so Theorem A4 could be extended in similar ways.

- Firstly [EG11; Epp+11; GM11] all solve the generalised problem where some elements $X_1 \subseteq \mathcal{U}$ are inserted, some elements $X_2 \subseteq \mathcal{U}$ are deleted and X_2 is not necessarily a subset of X_1. Both $X_1 \setminus X_2$ and $X_2 \setminus X_1$ should be computed and both are known to be small. This can be used to solve the *set reconciliation problem*, where two parties hold similar data sets X_1 and X_2 and both should compute the union $X_1 \cup X_2$ with communication overhead linear in the symmetric difference of X_1 and X_2.
  Using the same approach, it is then cells with a count of 1 or -1 that might hold a single element of $X_1 \setminus X_2$ or $X_2 \setminus X_1$, respectively, but additional checksums are needed. For instance, a count of 1 might be produced by an interaction of 2 elements from $X_1 \setminus X_2$ and one element from $X_2 \setminus X_1$.
- Both [EG11] and [GM11] handle the case where X_1 and X_2 are multisets.
- In [GM11], keys are associated with values, making IBLTs a variant of dictionaries. However, a **delete** needs to know the value associated with the key that is to be deleted. The authors point out that **lookup** can be supported, though only with a certain probability for success. The case of the same key appearing with different associated values can be addressed to a limited degree.
- In [Epp+11], auxiliary algorithms such as IBF-subtraction and an estimation of m – and thus the problem choosing n – are discussed.

Part II.

Background

4. Fundamental Models, Techniques and Algorithms

4.1. The Word RAM Model vs. Real Computers

The computation model underlying our theorems is a unit-cost word RAM model, explained, e.g. in [MS08, Chapter 2.2]. There is *word size* $w \in \mathbb{N}$, and a *word* is a sequence of w bits. Arithmetic operations (such as $+$, $-$, $\cdot$, $/$, div, mod) as well as simple bit operations (such as AND, OR, XOR) can be executed on words in one unit of time. Memory consists of up to 2^w words with *random access*, meaning the content of a word may be used to address another word in memory. We assume $w \geq C \log n$ such that an amount of memory polynomial in the input size can be addressed.

There exist word RAM algorithms that exploit the computational model in questionable ways, sometimes assuming very powerful constant-time operations on words, sometimes requiring unrealistically large values of w in an asymptotic analysis. We promise that the algorithms in this thesis do not stretch the word RAM model in such ways and admit efficient implementations on real computers with running times similar to what one would expect from the analysis.

We use one slightly uncommon operation where a few comments are in order.

4.1.1. The Parity Operation

The operation PARITY(x) for $x \in \{0,1\}^w$ counts the number of 1 bits in x, modulo 2. We assume it can be evaluated in time $O(1)$ despite the fact that many programming languages, including C++ do not support it as a primitive at the time of writing this thesis[1]. Of course the operations can be implemented manually, see Figure 4.1 for an implementation for 32 bit words. However, these implementations take time $O(w)$ and $O(\log w)$ instead of $O(1)$ as desired.

```
bool parity_w(uint32_t x) {
    bool c = 0;
    for (; x != 0; x >>= 1)
        c ^= x & 1;
    return c & 1;
}
```

```
bool parity_logw(uint32_t x) {
    for(int i = 16; i != 0; i >>= 1)
        x ^= x >> i;
    return x & 1;
}
```

Figure 4.1 Possible implementation of PARITY for word size $w = 32$ in C/C++. Generalising to arbitrary word sizes yields $O(w)$ and $O(\log w)$ algorithms, respectively.

[1] We ignore a detour via `std::bitset`.

One solution is to precompute parity values for (sub)words as shown in Figure 4.2. The best solution by far is, however, to simply use the POPCNT instruction provided by

```
vector<bool> table(0x10000,false);
void precalc_parity() {
    for(int i = 1; i < 0x10000; ++i) {
        table[i] = !table[i & (i-1)];
    }
}

bool parity_by_table(uint32_t x) {
    return table[x & 0xFFFF]
         ^ table[x >> 16   ];
}
```

Figure 4.2 Implementation of PARITY for 32 bit words using precomputed values for 16 bit words (taking 8 KiB or space). The precomputation uses the standard trick that `i & (i-1)` clears the least significant bit from a number $i \neq 0$.

many modern processors [Int07; AMD08] which counts the number of bits set in a word.[2] Compilers like gcc and msvc have since defined built-in functions that are translated into the POPCNT instruction on compilation. Moreover, it is expected that with the upcoming standard C++20 a popcount function will become an official part of the C++ language [Mau19]. We provide corresponding code in Figure 4.3.

```
#if __cplusplus >= 202005L // C++20 expected in May 2020
    #include<bit>
    bool parity(uint32_t x) {
        return std::popcount(x) & 1;
    }
#elif defined __GNUC__ // gnu compiler
    bool parity(uint32_t x) {
        return __builtin_parityl(x);
    }
#elif defined _MSC_VER // microsoft compiler
    #include <intrin.h>
    bool parity(uint32_t x) {
        return __popcnt(x) & 1;
    }
#endif
```

Figure 4.3 C++ code that makes use of the POPCNT instruction available on Intel and AMD CPUs, when compiled with a corresponding compiler.

All in all, the assumption that PARITY takes time "$O(1)$", i.e. about the same as other kinds of operations, is perfectly justified in practice.

4.1.2. Use Case: Scalar Product of Bit Vectors

In the context of Theorems B and C we will frequently assume that the scalar product $\bigoplus_{i\in[L]} a_i b_i$ of two bit sequences $a, b \in \{0,1\}^L$ can be computed in time $O(L/w)$. We provide corresponding code using XOR operations and a PARITY operation in Figure 4.4.

[2] Intel implements POPCNT since the Nehalem Architecture (2008) and AMD since AMD 10h (2007).

```
bool scalarProduct(uint32_t* a, uint32_t* b, size_t numWords) {
    uint32_t result = 0;
    for(size_t i = 0; i < numWords; ++i) {
        result ^= a[i] & b[i];
    }
    return parity(result);
}
```

Figure 4.4 Implementation of the scalar product of two bit strings $a, b \in \{0,1\}^L$. If a and b do not start at a 32-bit word boundary or if `numWords` $= L/32$ is not an integer, slight adjustments are needed.

4.2. Double Hashing and its Uses

Quite a few data structures associate a given key x with a sequence $h_1(x), h_2(x), h_3(x), \ldots \in [n]_0$ of hash values that are assumed to be independent in the analysis. In such cases one can try to use *double hashing*, which means using an arithmetic progression modulo n of the form $h_1(x), h_1(x) + h_2(x), h_1(x) + 2h_2(x), \ldots, h_1(x) + kh_2(x), \ldots \in [n]_0$ instead. The *empirical* observation is that, in many cases, the overall behaviour of the data structure does not significantly change. Running time, on the other hand, is improved since only two hash function evaluations are required for each key. Of course, *proving* that this is the case is an entirely different matter. We list a few settings where double hashing has been considered.

Hash Tables with Open Addressing. Double hashing as a probing strategy for hash tables with open addressing is basic textbook material, for instance [Cor+09, Chapter 11.4] or [Knu98, Chapter 6.4]. A partial proof that search times are unaffected is given in [GS78] (for loads $\alpha_0 \leq 0.31$). The proof was completed in [LM93; SS90].

Bloom Filters. Double hashing does not affect the false positive rate of Bloom filters [KM08].

Balanced Allocations. Double hashing does not affect the maximum load in balanced allocation settings [Mit14; Mit16].

Hypergraph Peeling. A *partial* proof that double hashing does not affect peeling thresholds of random hypergraphs is found in [MT12].

Cuckoo Hashing. Experiments in [MT12] suggest that double hashing does not affect the load threshold of cuckoo hash tables. An incomplete proof is found in [Lec13]. It is the topic of Chapter 10 and Theorems E and E1 to complete the argument.

4.3. Full Randomness Assumption

As described in Section 3.2 we rely on the full randomness assumption, which gives us access to functions $h \xleftarrow{\$} R^{\mathcal{U}}$ "for free", for any range R and universe $\mathcal{U}$. "For free" means such functions can be selected and evaluated in constant time and require no memory to store.

This is, of course, a gross simplification as such functions would require at least $|\mathcal{U}| \log |R|$ bits to store, a completely unacceptable amount. To defend the practical relevance of our results in light of this problem, the following three strategies come to mind:

1. Review our arguments to base them on weaker assumptions on h that can be fulfilled in practice.
2. "Bite the bullet" and construct random functions $h\colon \mathcal{U} \to [n]$, but save space by only making them fully random on relevant subsets of $\mathcal{U}$.
3. Replace the full randomness assumption with the *cryptographic* assumption that random functions exist that are *computationally indinstinguishable* from the fully random functions we need. While we cannot hope to prove such an assumption, promising candidate functions exist.

We say a few words about all three strategies.

4.3.1. Building on Weaker Assumptions

Instead of sampling a hash function h from the set of all functions $R^{\mathcal{U}}$, we can sample $h \xleftarrow{\$} \mathcal{H}$ for a smaller family $\mathcal{H} \subset R^{\mathcal{U}}$. In that case h is not fully random, but may satisfy the weaker properties of *universality* [CW79], meaning $\Pr[h(x) = h(y)] \leq 1/|R|$ for $x \neq y \in \mathcal{U}$ or *k-wise independence* [WC81], meaning for distinct $x_1, \ldots, x_k \in \mathcal{U}$ and arbitrary $r_1, \ldots, r_k \in R$ we have $\Pr\left[\forall i \in [k]\colon h(x_i) = r_i\right] = |R|^{-k}$. Universal and 2-wise independent families with highly efficient evaluation times in practice are described in [Tho15], for instance multiply-shift hashing [Die+97].

For some hashing-based data structures, it is known that universality or k-wise independence for some k is sufficient to achieve the same asymptotic behaviour as when using fully random functions. This includes some results on *standard* cuckoo hashing. A good overview is given in [Rin15, Section 5.2]. For generalised cuckoo hashing and most other data structures discussed in this thesis, not much seems to be known and this thesis does not pursue this line of research.

If the hash function and the input data are both random, more can be said. Indeed, if each element of the input contains a sufficient amount of entropy given all previous elements, then universal hashing is sufficient to achieve the same performance as with fully random hashing in a wide range of applications, as was shown in [CMV13]. The result goes a long way in explaining "Why Simple Hash Functions Work". However, it does not help with getting rid of the full randomness assumption. We can only trade it for an assumption on randomness in the inputs limits, which uncomfortably limits the scope of applicability of our data structure.

4.3.2. Constructing Fully Random Functions: Split and Share

A slightly simplified formulation of a theorem from [DR09] reads as follows.

▶ **Theorem 4.1** (Dietzfelbinger and Rink [DR09]). *For any $c > 0$, $N \in \mathbb{N}$ and any range R a random hash function $h\colon \mathcal{U} \to R$ with evaluation time $O(c)$ and occupying $O(N \log |R|)$ bits can be constructed such that the following holds. For any $D \subseteq \mathcal{U}$ of size N, conditioned under an event E_D with $\Pr[E_D] = 1 - O(N^{-c})$, the function h is fully random on D.*

Assume we wish to build a dictionary-type data structure for a set $S \subseteq \mathcal{U}$ of size m. When applying this theorem directly for $N = m$ and $D = S$ we obtain a hash function $h\colon \mathcal{U} \to R$ that is fully random on S whp. However, storing h requires $O(m \log |R|)$ bits which is less than $O(|\mathcal{U}| \log |R|)$ but still very large.

The *splitting trick* [DR09] is to first choose a desired chunk size $C = m^\varepsilon$ and partition the set S via an outer hash function $h_0 \colon \mathcal{U} \to [m/C]$ into chunks $S_1, \ldots, S_{m/C}$. We require that $h_0 \leftarrow\!\!\boxdot\!\!- \mathcal{H}$ is chosen from universal class of hash functions $\mathcal{H}$ (see above), which guarantees that the maximum chunk size is $\hat{C} = O(C)$ whp. Applying Theorem 4.1 for $N = \hat{C}$ and $c = 2/\varepsilon$ yields a hash function $h \colon \mathcal{U} \to R$ that requires $O(m^\varepsilon \log |R|)$ bits to store. Crucially, using a first moment argument, h is whp fully random on *all* chunks $S_1, \ldots, S_{m/C}$ simultaneously (but not fully random on S as a whole). Thus h can be shared among the chunks. For each chunk, a data structure of the desired type is constructed based on the full randomness assumption using h.

From a theoretical point of view, this is a satisfying solution since the overhead for saving h can be made arbitrarily small. Moreover, input partitioning can be useful for other reasons as well, see Section 3.4.2.

From a practical perspective, however, the overhead seems unnecessary, given that pseudo-random hash functions seem to exist, as we discuss now.

4.3.3. Hoping for Indistinguishability from Randomness

There are quite a few functions that are fast to evaluate in practice and behave like random functions in many ways. The test suite `SMHasher` by Appleby [App12] tests the performance and statistical properties of non-cryptographic hash functions in that sense. Any fast hash function that does not exhibit any of the weaknesses for which a test is implemented might be a good general-purpose hash function. Some popular non-cryptographic hash functions that perform well are `MurmurHash`, `CityHash` and `xxHash`. Experience shows that the behaviour of many hashing-based data structures and algorithms does not noticeably change when using these hash functions instead of fully random functions.

One notable exception is *adversarial settings*, where an *attacker* tries to compromise the performance of a data structure by carefully crafting malicious input data. Since the attacker is granted (partial) knowledge about the data structures internals, any performance guarantees relying on the independence of hash function and input are void, and non-cryptographic hash functions have limited utility. The topic has recently gained some attention, in particular in the context of Bloom filters [MNS11; NY19; GKL15; CPS19], but the problem is far more fundamental. So-called *hash-flooding* is a realistic *denial of service attack* on systems that insert user-selected key/value pairs into a hash table. If many or all keys share the same hash value, then a hash table using linear chaining or linear probing will exhibit linear running time per operation, crippling system performance [LN93; CW03].

A corresponding vulnerability in the widely used hash functions `MurmurHash2` and `MurmurHash3` was demonstrated in [AB12a], fabricating arbitrary multi-collisions that work for all seed values. Conveniently, the authors simultaneously propose an alternative hash function `SipHash` [AB12b] that combines good performance with cryptographic security claims. Previously, cryptographic hash functions were mostly shunned by the data structure community due to their significantly slower evaluation time. `SipHash` has since found its way into several standard libraries (including Python, Ruby, Rust, Haskell) and an improved implementation `HighwayHash` is described in [ACW16].

The author of this thesis has little expertise in these matters, but the development seems promising. If `SipHash` (or `HighwayHash`) really is a pseudo-random function

in a *cryptographic sense*[3], meaning it is (when given a secret key/seed) computationally indistinguishable from a fully random function, then no efficient hashing-based algorithm that relies on a fully random function behaves noticeably different when used with such a hash function instead (otherwise that algorithm would be a distinguisher). In that happy case, the need for full randomness and the need for fast, practical hash functions would not be at odds.

4.4. Solving Linear Systems

Constructing retrieval data structures as explained in Section 3.4 requires finding solutions $\vec{z} \in \mathbb{F}_2^n$ to linear systems $A \cdot \vec{z} = \vec{b}$ for given $A \in \mathbb{F}_2^{m \times n}$ and $\vec{b} \in \mathbb{F}_2^m$. In the cases we consider $m = \Theta(n)$ and A contains only $\widetilde{O}(n)$ many 1-entries. Since the cubic running time of naive Gaussian elimination is infeasible already for moderately large m, and since the technique of input partitioning we discussed in Section 3.4.2 is not without disadvantages, we need better algorithms for solving sparse linear systems over $\mathbb{F}_2$. In this section we consider the following three:

1. **Wiedemann's randomised algorithm** [Wie86] with expected running time $\widetilde{O}(n^2)$.
2. Clever variants of Gaussian elimination called **structured** [LO90] or **lazy** [GOV16] **Gaussian elimination** performing row operations in an order that preserves the sparsity of the system as long as possible.
   None of our theorems mentions these techniques since, as far as we know, there is no mathematical analysis of them. In our experiments, on the other hand, they drastically reduce the running time of naive Gaussian elimination on sparse random matrices, outperforming Wiedemann's algorithm for small to medium inputs.
3. The **Method of Four Russians** improves upon Gaussian elimination by precomputing certain row sums, achieving a speedup by a factor of $\Theta(\log m)$.

In all cases, there are opportunities to exploit bit parallelism improving running times by another factor of w.

4.4.1. Wiedemann's Algorithm

In Section 3.4 and Chapter 7 we apply the following theorem by Wiedemann in cases with $q = 2$ and $\psi = \widetilde{O}(n)$, concluding that solutions to sparse linear systems with n variables can be computed in time $\widetilde{O}(n^2)$.

► **Theorem 4.2** (Wiedemann's Algorithm [Wie86]). *Let $\mathbb{F}_q$ be the finite field with q elements, $A \in \mathbb{F}_q^{m \times n}$ a matrix with ψ non-zero entries, $m, n, \psi \in \mathbb{N}$ and $\vec{b} \in \mathbb{F}_q^m$.*

(i) *If $m = n$ and A is regular, then a solution $\vec{z} \in \mathbb{F}_q^n$ to $A\vec{z} = \vec{b}$ can be computed with $O(n\psi)$ field operations ([Wie86, Algorithm 1]).*

(ii) *If $m < n$ and A has rank m, then a solution $\vec{z} \in \mathbb{F}_q^n$ to $A\vec{z} = \vec{b}$ can be computed with $O(n(\psi + n \log n))$ field operations ([Wie86, Theorem 1']).*

[3] Note that the existence of pseudo-random functions is an open question in cryptography. By extension no prove for this claim exists.

While we use this interesting result in some of our theorems, there seems to be no good niche for it in our experiments: For large n, the quadratic running time is still prohibitive, and if input partitioning techniques are used (see Section 3.4.2), we can decrease the size of linear systems to regimes where asymptotically cubic approaches with small constants and high cache efficiency are faster – at least in our implementations.

We remark that there are opportunities for shaving logarithmic factors from Wiedemann's algorithm:

(1) Running time for setting **(i)** is dominated by multiplications of A with a vector, each multiplication taking time $O(\psi)$. If the ψ non-zero entries of A are concentrated in blocks, for instance, if A is a matrix from Definition 2.5, this can be reduced to $O(\psi/w)$, where w is the word length of our word RAM.

(2) The proof of **(ii)** relies on a reduction to **(i)** that appends random rows to A, hoping for the resulting square matrix to be regular. There is a strategy where the added rows have $O(n \log n)$ non-zero entries and attempts succeed with constant probability. It seems plausible that, at least for some A, a smaller number of non-zero entries suffices,g which improves running time of **(ii)** if $\psi = o(n \log n)$.

(3) Occasionally we wish to solve $A \cdot Z = B$ where Z and B have $r = O(\log n)$ columns. This can be done more quickly than by using Wiedemann's algorithm independently r times. With word-level parallelism, the author suspects, a factor of $O(\log r)$ suffices.

Since we wish to side-step such details, we use $\widetilde{O}$-notation when providing running times for Wiedemann's algorithm.

4.4.2. Heuristic Presolving: Lazy Gaussian Elimination

Lazy Gaussian Elimination is best understood as a presolver that takes a sparse linear system A as input and outputs a dense linear system A' with significantly fewer equations and variables. The transformed system A' is solvable if and only if A is solvable and any solution for A' can be extended to a solution for A by initialising the "presolved" variables using back substitution.

We adapt the descriptions from [LO90; GOV16] to linear systems $A \cdot \vec{z} = \vec{b}$ where $A \stackrel{\mathrm{d}}{=} A^L_{m,n}$ is the random matrix from Definition 2.5. An experimental evaluation of the substantial benefits this provides is found in Section 12.2.6.

During the algorithm, the intermediate linear systems have the form shown in Figure 4.5. The $B \in \mathbb{N}$ *light blocks* have individual sizes $L_1, \ldots, L_B \in \mathbb{N}$ meaning the i-th light block contains L_i *light variables*. Each equation e is *incident* to 0, 1 or 2 light blocks and has a vector of coefficients of appropriate length for each. Moreover, there is a single *heavy block* of L_0 *heavy variables*, for which every equation has a coefficient vector.

Note that the initial system $A^L_{m,n} \cdot \vec{z} = \vec{b}$ fits this description with n/L uniformly sized light blocks ($L_1 = \ldots = L_{n/L} = L$) and an empty heavy block ($L_0 = 0$). The algorithm will terminate with a matrix containing only the heavy block and no light blocks.

We assume that light blocks of size 0, light blocks not incident to any equation, and incidences between light blocks and equations with coefficient vector $\vec{0}$ are implicitly removed. Each step of the algorithm modifies the linear system in one of two ways.

Case 1: Each equation is incident to 0 or 2 light blocks. If no light block is left, the algorithm ends. Otherwise, denote by $\deg(i)$ the number of equations incident to the

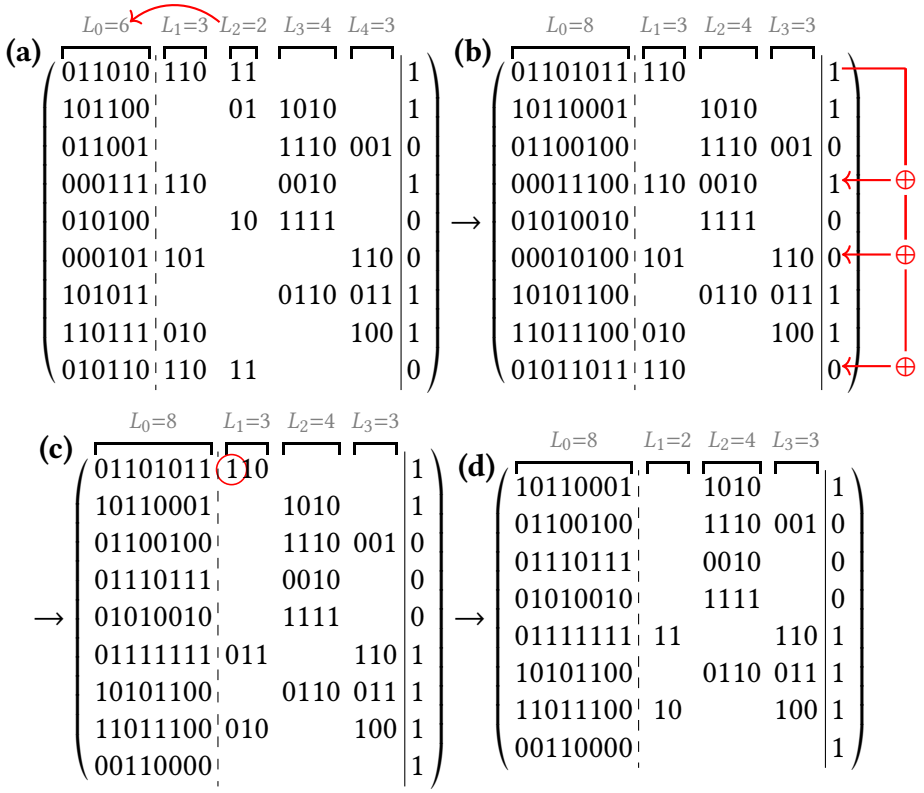


Figure 4.5 Example for two steps of lazy Gaussian elimination. Case 1 applies to **(a)** and the second block of variables is made heavy by migrating the variables to the heavy block. This yields **(b)** where Case 2 applies because the first equation is incident only to the first light block. One variable, here the first, is eliminated from all other equations, yielding **(c)**. Removing the equation and the eliminated variable yields **(d)**.

i-th light block and pick $i \in [B]$ that maximises $\deg(i)/L_i$.[4]

Consider the step from **(a)** to **(b)** in Figure 4.5 where $i = 2$. The L_i variables of block i are *made heavy*, meaning L_0 increases by L_i, block i is removed, the coefficients associated with block i are moved to the heavy block and the equations not previously incident to block i are assigned zero coefficients for the new heavy variables.

Note that the equations that were incident to block i are now incident to only 1 light block, meaning Case 2 applies to them.

Case 2: An equation e is incident to precisely one light block i. Consider Figure 4.5 **(b)** to **(c)** where e is the first equation. Pick a variable v from block i for which e has coefficient 1. Then eliminate this variable from all other equations by adding e if necessary. This exclusively affects equations that were already incident to block i and only by changing their coefficients for block i and the heavy block (as well as the right-hand side). Crucially, no new incidences are created (though incidences may

[4] This heuristic makes sense if $\deg(i)$ is conceptualised as *utility*, as it is the number of equations that can then be removed using Case 2, and L_i as a *price*, as the number of heavy variables should be kept small. Picking $i \xleftarrow{\$} [B]$ at random has slightly worse performance.

vanish).

Now consider Figure 4.5 **(c)** to **(d)**. Equation e and the column of v are removed (this decrements L_i). Clearly, by initialising v to the right value from $\mathbb{F}_2$, any solution to the reduced system can be extended to a solution also satisfying e.

Implementation and Running Time. To track applicability of Case 2, we keep a queue of equations incident to 1 light block. To quickly settle on a block in Case 1, we maintain a priority queue of light blocks (the key of $i \in [B]$ being $\deg(i)/L_i$).

Proving that the algorithm is *useful* would require getting a grip on the size of the remaining system, which we cannot provide. To see that it "cannot hurt", we consider its running time. The main cost lies in eliminating variables. Eliminating a variable from a light block i requires at most $\deg(i) - 1$ additions of equations, which is dominated by adding coefficients of the $L_0 = O(n)$ variables in the heavy block, w bits at a time. Since each variable is eliminated at most once, the total running time is $\sum_i L_i \deg(i) O(n/w) = \sum_i \deg(i) O(nL/w) = 2mO(nL/w) = O(mnL/w)$. For $L = O(\log n)$ this is $O(mn)$ and therefore faster than any linear system solver we consider.

4.4.3. The Method of Four Russians

The name "Method of Four Russians" goes back to a paper on computing the transitive closure of a directed graph [Arl+70] but is used for various ideas that exploit precomputation to shave logarithmic factors from matrix algorithms, including matrix multiplication [AH74, Chapter 6], edit distance [MP80] and linear system solvers [Bar09, Chapter 9].

We give our own version for linear system solving, since, compared to [Bar09] our algorithm is slightly simpler as it does not require Gray Codes and makes no assumption on the input matrix being random. We claim:

► **Lemma 4.3.** *Let $A \in \{0,1\}^{m \times n}$ be a matrix of rank m.*

(i) *A can be brought into row echelon form in $O(\frac{mn^2}{w \log m})$ word operations.*

(ii) *Let $r < n$ and $B \in \{0,1\}^{m \times r}$. A solution $Z \in \{0,1\}^{n \times r}$ to $A \cdot Z = B$ can be computed in $O(\frac{mn^2}{w \log m})$ word operations.*

Proof. **(i)** In the first pass of the algorithm we transform

$$A \in \{0,1\}^{m \times n} \text{ into } A' = \begin{pmatrix} A'_1 \in \{0,1\}^{d \times k} & A'_2 \in \{0,1\}^{d \times (n-k)} \\ 0 & A'_3 \in \{0,1\}^{(m-d) \times (n-k)} \end{pmatrix}$$

via row operations where k is a parameter chosen later, $d \leq k$ and A'_1 is in row echelon form. Subsequent passes then transform A'_3 into row echelon form. Note that Gaussian elimination corresponds to $k = 1$. In the following we only describe the first pass.

For a row $v \in \{0,1\}^n$ call $v[1 \ldots k] \in \{0,1\}^k$ the *prefix* of v. The rows of A are examined one by one. We maintain the vector-space $P \subseteq \{0,1\}^k$ spanned by the prefixes of all examined rows and a table T that assigns to each $p \in P$ a vector $T[p] \in \{0,1\}^n$ with prefix p that is a linear combination of examined rows. Let d be the dimension of P. Initially we have $d = 0$, $P = \{0^k\}$ and $T[0^k] = 0^n$. When row $v \in \{0,1\}^n$ with prefix p is examined, there are two cases. If $p \in P$ then $T[p]$ is added to v which yields a transformed row v' with prefix 0. If $p \notin P$ then we set $P' = P \oplus \{0^k, p\}$ and for $p' \in P$ we set $T[p' \oplus p] = T[p] \oplus v$, maintaining the invariant.

We end up with a modified matrix where all rows have prefix 0^k, except for the $d = \dim(P) \leq k$ rows that increased the dimension of P. We bring them to the top of A by swapping rows. It is easy to see that instead of using Gaussian elimination on these d rows to bring them into echelon form, we may instead select appropriate rows from T to replace the first d rows. The full algorithm is given as Figure 4.6.
One pass of the algorithm requires $O(2^d n) = O(2^k n)$ bit operations to build T and $O((m-d)n) = O(mn)$ bit operations for adding the precomputed rows from T to rows from A. Choosing $k = \log m$ this sums to $O(mn)$. Since $O(\frac{n}{k})$ passes are needed and since all relevant operations admit bit-parallel execution, the claim follows.
One can check that $k = \log m - \log\log m$ minimises the constant factor.

```
1  Algorithm MethodOfFourRussians(A ∈ {0,1}^{m×n}):
2      (r, c) ← (1, 1) // next row and column index to consider
3      while r ≤ m and c ≤ n do
           // todo: exploit that the first c − 1 bits of all rows are zero in this pass
4          k ← if m − r + 1 ≤ 4 then 1 else ⌊log(m − r + 1) − log log(m − r + 1)⌋
5          (d, P_0, T[0^k]) ← (0, {0^k}, 0^n) // trivial space of prefixes
6          for r' = r, ..., m do
7              p ← A[r'][c ... c+k−1]
8              if p = 0^k then
9                  continue
10             else if p ∈ P_d then
11                 A[r'] ← A[r'] ⊕ T[p]
12             else
13                 P_{d+1} ← P_d ⊕ {0, p}
14                 for p' ∈ P_d do
15                     T[p ⊕ p'] ← A[r'] ⊕ T[p']
16                 A[r'] ← A[r + d]
17                 A[r + d] ← empty // create a gap to be filled later
18                 d ← d + 1
           // A[r ... r+d−1] are empty. replace with echelon-shaped basis of equations in T
19         r' ← r
20         lastLZ ← 0
21         for p ∈ P_d \ {0^k}, lexicographically descending do
22             if leadingZeroes(p) > lastLZ then
23                 lastLZ ← leadingZeroes(p)
24                 A[r'] ← T[p]
25                 r' ← r' + 1
26         (r, c) ← (r + d, c + k)
27     return A
```

Figure 4.6 Transforms the input matrix A into row echelon form using the Method of Four Russians. By adding right hand sides and a back substitution step, we obtain a linear system solver.

(ii) Simply generate the matrix $A' = (A \mid B) \in \{0,1\}^{m \times (n+r)}$ and bring it into row echelon form using **(i)**. The solution matrix Z can then be obtained by back substitution in $O(mnr/w)$ word operations. In the untypical case of large r, say $r = \Theta(n)$, where back substitution would dominate the running time, similar techniques as in **(i)** save a factor of $\log m$. ◀

4.5. Connection to Coding Theory

The core of our proof of Theorem A is a technique from coding theory. The connection is not at all surprising. To explain why, we briefly introduce the binary erasure channel and point out the relationship to our notions on hypergraphs from Definition 2.1. Afterwards, we examine how closely the task of constructing good codes aligns with the task of constructing good hashing-based data structures.

4.5.1. The Binary Erasure Channel and Low Density Parity Check Codes

The *binary erasure channel* (BEC) is a simple but important setting. We recommend [RU08, Chapter 3] for an excellent introduction to this subject. When a sequence $(x_1, \dots, x_m) \in \{0,1\}^m$ is sent over the BEC, the receiver sees a sequence $(y_1, \dots, y_m) \in \{0,1,?\}^m$ where for each $i \in [m]$ independently, the i-th bit is *erased* ($y_i = ?$) with probability $\varepsilon \in [0,1]$ and unchanged ($y_i = x_i$) with probability $1 - \varepsilon$. For reliable communication over such channels, redundancy is introduced. In *linear codes*, several *parity conditions* are each specified by a set $P \subseteq [m]$ and dictate that $\bigoplus_{i \in P} x_i$ is zero. The set of admissible messages (*codewords*) then forms a linear subspace of $\{0,1\}^m$.

To relate this to hypergraphs, let V be the set of all parity conditions and let $E^+ = \{e_1, \dots, e_m\}$ where $v \in e_i$ if x_i is involved in parity condition v. The incidence graph of $H^+ = (V, E^+)$ is known as the *Tanner graph* [Tan81]. In *low density parity check (LDPC) codes* the Tanner graph is sparse.

After transmission, bits corresponding to some set $E \subseteq E^+$ are erased and we consider $H = (V, E)$. When decoding, we seek an assignment $x_{\text{dec}} \colon E \to \{0,1\}$ such that for $v \in V$ we have $\bigoplus_{e \in E, e \ni v} x_{\text{dec}}(e) = c_v$ where c_v is the parity of the successfully transferred bits involved in parity condition v. The existence of a solution is guaranteed by construction, namely $x_{\text{dec}}(e_i) = x_i$ for $e_i \in E$. Uniqueness of the solution and thus success of the ideal *maximum a posteriori probability decoder* (MAP-decoder) requires the kernel of the incidence matrix of H to be trivial – which is equivalent to H being solvable.

Success of the linear time *belief propagation decoder* (BP-decoder) requires 1-peelability of H. This decoder iteratively identifies a parity condition, where all but one of the involved bits are known, and then decodes the unknown bit.

4.5.2. Good Codes vs. Good Data Structures

We now know two perspectives for evaluating the properties of random hypergraphs. The first is that of constructing hashing-based data structures (HBDS) such as cuckoo dictionaries and retrieval data structures. The second is that of decoding LDPC codes.

There is substantial alignment concerning the issue of what makes a "good" hypergraph, see e.g. [MV12a; MV12b].

Hyperedge Size. The (average) hyperedge size k is, in HBDS, related to (average) query time and (average) number of cache faults per query. In LDPC codes, k is the (average) number of parity conditions relating to each message bit and contributes to overall encoding and decoding time. Thus, *small k is good.*

Density. In HBDS, a high edge density $c = |E|/|V|$ means accommodating many objects in little space (high load), while in LDPC codes it means recovering many erased bits from little redundancy (high rate). Thus, *large c is good.* In both cases, $c = 1$ is often an obvious information-theoretic upper bound.

Peelability, Solvability, Orientability. As far as we are aware, among the properties from Definition 2.1 important in HBDS, only 1-peelability and solvability play a role for LDPC codes. Luckily, in the context of Theorem A, the thresholds for solvability and 1-orientability coincide and the generalisations to $\ell > 1$ are easily established.

The role of randomness. An LDPC code is given by a fixed hypergraph H^+ that we are free to design. For instance, we might give all vertices the same degree. Since H arises from a random ε-fraction of the hyperedges of H^+, this gives us control (proportional to ε) on H as well.

When building a HBDS, we are essentially restricted to i.i.d. random hyperedges as explained in Section 3.2. The general techniques from LDPC codes still apply in this special case.

4.6. Dictionaries not Building on Random Hypergraphs

This thesis throws a very specific solution strategy (involving random hypergraphs) at various dictionary-type data structures. It should be noted that, in particular in the case of general-purpose dictionaries, the cuckoo-type approaches we discussed are not the most widespread in practice. We now take a moment to widen our perspective and mention some well-known approaches briefly, without explaining them. Needless to say, the literature on the topic is vast, and any one-page account of the field is necessarily incomplete.

The Big Players. Consider dictionaries storing m key/value pairs of $O(\log m)$ bits each, using memory sufficient for $(1+\varepsilon)m$ pairs with $\varepsilon \geq 0$, and average lookup times independent of m. Note that this already excludes search trees that store elements in sorted order, due to lookup times of $\Omega(\log m)$.

The simplest data structure fulfilling the requirements is a hash table with *linear chaining*. With a table of n buckets the expected lookup time is $O(1 + m/n)$ [Cor+09, Chap. 11.3.3]. The overhead is due to $O(m + n)$ pointers, that are often as large as key/value pairs themselves. Note that for $n = \Theta(m)$ the approach is optimal "up to constants". The logical next step is to make ε a variable and optimize running times as $\varepsilon \to 0$.

This can be done for open addressing schemes where data is stored in a large table without pointers. Access times now depend on the probing strategy and whether a lookup is *successful* (faster) or *unsuccessful* (slower). In the case of *linear probing*, the *average* lookup times are $O(1/\varepsilon)$ and $O(1/\varepsilon^2)$, respectively [Cor+09, Chap. 11.4], in the case of *uniform probing*, they are $O(\log(1/\varepsilon))$ and $O(1/\varepsilon)$ [Sed03, Section 14.3]. To reduce the average times of unsuccessful lookups (without affecting average times of successful lookups) *Robin Hood hashing* can be used [CLM85]. One potential problem remains, however: The *maximum*

lookup times depend on m, in the case of Robin Hood hashing with uniform probing it is still $O(\log \log m)$ whp [DMV04].

In cuckoo hashing, in contrast, the worst-case lookup times are independent of m. By considering the relationship between the number of options per key and the load threshold, we see that the average *and* worst-case lookup times are both $O(\log(1/\varepsilon))$.

We remark that this is not the end of the line for theory on dictionary data structures. For instance, Arbitman, Naor and Segev [ANS10] achieve fully dynamic dictionaries with constant time operations in the worst case and $\varepsilon = O((\log \log m / \log m)^{1/2})$, though it seems unlikely that such a dictionary is competitive in practice.

Practical Perspective. The reason why cuckoo hashing does not cast a larger shadow in practice is at least in part because very high memory efficiency (ε close to 0) and good worst-case lookup times (as opposed to average lookup times) often play a limited role.

The c++11 standard specifies that the default dictionary data structure `std::unordered_map` uses linear chaining[5]. As far as the author can tell, the most successful attempts at improving upon the standard use linear probing, sometimes including Robin Hood hashing. A highly optimised hash table `absl::flat_hash_map` provided by Google [Goo18] uses a variant of linear probing (without a Robin Hood component), probes 16 positions at a time using SIMD instructions and has load factors between 43% and 88%.

So when it comes to general-purpose dictionaries, the algorithmically simplest approaches (be it with significant engineering effort added) seem to be the favoured choice. The niche for cuckoo hashing seems to be high static loads, and high dynamic loads, as explained in Section 3.3.1.

[5] Technically it only specifies that an iteration over a "bucket" of elements with the same hash value is possible in time proportional to that bucket.

Notation of Chapter 5

Local Weak Limit

$\mathcal{G}^*$	space of *rooted* locally finite graphs (possibly weighted)
$(G)_t \in \mathcal{G}^*$	for $G \in \mathcal{G}^*$ and $t \in \mathbb{N}$, the t-neighbourhood of the root
$\mathrm{GWT}_{\mathrm{Po}(\lambda)} \in \mathcal{G}^*$	Galton-Watson tree with offspring distribution $\mathrm{Po}(\lambda)$
μ	probability measure on $\mathcal{G}^*$
$(\mu_n)_{n\in\mathbb{N}} \rightsquigarrow \mu$	weak limit of probability measures, we also write $(X_n)_{n\in\mathbb{N}} \rightsquigarrow X$ when $X_n \sim \mu_n$ and $X \sim \mu$
$G(\circ) \in \mathcal{G}^*$	obtained from *unrooted* graph G by making $\circ \in V(G)$ the root
$U(G)$	distribution of $G(\circ) \in \mathcal{G}^*$ when $\circ \leftarrow$ 🎲 $V(G)$
$(G_n)_{n\in\mathbb{N}} \xrightarrow{\text{LWL}} \mu$	the local weak limit of the *unrooted* graphs $(G_n)_{n\in\mathbb{N}}$ is μ equivalent to $(U(G_n))_{n\in\mathbb{N}} \rightsquigarrow \mu$; we write $(G_n)_{n\in\mathbb{N}} \xrightarrow{\text{LWL}} G$ if $G \sim \mu$

Lelarge's Theorem

G	an unrooted bipartite graph $G = (A, B, E)$
η	vertex weights $\eta\colon A \cup B \to \mathbb{N}$
ζ	edge weights $\zeta\colon E \to \mathbb{N}$
$M(G)$	size of a maximum matching or allocation in G
Φ^A (and Φ^B)	distributions describing degree of A-vertices (B-vertices), sometimes including vertex weight and weights of incident edges
$\mathrm{GWT}_{\Phi^A,\Phi^B}$	bipartite Galton-Watson tree based on distributions Φ^A, Φ^B
$\mathcal{M}(\Phi^A, \Phi^B)$	the value $\lim_{n\to\infty} M(G_n)/\lvert A_n\rvert$ characterised by Lelarge's Theorem

General

$\mathbb{1}[p]$	1 if the predicate p is true and 0 otherwise
δ_x	Dirac measure with $\delta_x(A) = \mathbb{1}[x \in A]$ for measurable sets A
$\mathrm{Po}(\lambda)$	Poisson distribution with parameter λ
$\mathrm{Bin}(n, p)$	Binomial distribution with parameters n and p

5. Local Weak Limits and Lelarge's Theorem

As mentioned in Section 2.4.4, Lelarge [Lel12] and later Leconte, Lelarge and Massoulié [LLM13] have unified and extended previous results on orientability of random hypergraphs. We refer to both results [Lel12, Thm. 4.1] [LLM13, Thm. 2.1] as *Lelarge's Theorem.* It is crucial in our proofs of Theorems D and F, implicit in the proof of Theorem E and also used, though less prominently, for proving Theorem A.

The Form of Lelarge's Theorem. Consider the simplest case where $(H_n = (V_n, E_n))_{n\in\mathbb{N}}$ is a family of hypergraphs and we wish to know whether H_n is 1-orientable whp. Note that a 1-orientation $\sigma\colon E_n \to V_n$ of H_n is simply a matching $\sigma \subseteq E_n \times V_n$ of size $|\sigma| = |E_n|$ in the bipartite incidence graph $G_n = (E_n, V_n, \text{“}\ni\text{”})$ of H_n. By $M(G_n)$ we denote the size of a maximum matching in G_n. In this context, Lelarge's Theorem asserts:

$$\textbf{If } (G_n)_{n\in\mathbb{N}} \xrightarrow{\text{LWL}} \mathrm{GWT}_{\Phi^A,\Phi^B} \textbf{ then } \lim_{n\to\infty} \frac{M(G_n)}{|E_n|} = \mathcal{M}(\Phi^A, \Phi^B).$$

Here, $\xrightarrow{\text{LWL}}$ stands for *local weak limit* [Lec13; LLM13] also known as *Benjamini-Schramm limit* [BS11] or *random weak limit* [Lel12; Bor16; AL07]. Roughly speaking, the limit captures the distribution of the k-hop neighbourhood of a random vertex of G_n, for any constant k as $n \to \infty$.

The random variable $\mathrm{GWT}_{\Phi^A,\Phi^B}$ is a *bipartite Galton-Watson tree* depending on two distributions Φ^A, Φ^B and naturally occurs in the cases we consider. The theorem characterises the asymptotic proportion $M(G_n)/|E_n|$ of elements from E_n that can be matched as a function $\mathcal{M}$ of Φ^A and Φ^B, the details of which we ignore for now.

Applying Lelarge's Theorem. If we wish to apply Lelarge's Theorem to decide whether H_n is 1-orientable whp, three things remain to be done.

1. Identify the distribution Φ^A and Φ^B and prove that $(G_n)_{n\in\mathbb{N}}$ has local weak limit $\mathrm{GWT}_{\Phi^A,\Phi^B}$.
2. Compute $\mathcal{M}(\Phi^A, \Phi^B)$. Typically H_n, G_n, Φ^A and Φ^B depend on a density parameter $c \in \mathbb{R}^+$ and we are really interested in a threshold $c^* = \max\{c \in \mathbb{R}^+ \mid \mathcal{M}(\Phi^A_c, \Phi^B_c) = 1\}$. Finding c^* may be hard to do analytically, but is easy numerically in all cases we consider.
3. If $\mathcal{M}(\Phi^A, \Phi^B) < 1$ we conclude $M(G_n) < |E_n|$ whp, meaning H_n is not 1-orientable whp. If $\mathcal{M}(\Phi^A, \Phi^B) = 1$ we only get $M(G_n) = |E_n| - o(n)$ at first, leaving a gap to the desired result “$M(G_n) = |E_n|$ whp”, which has to be bridged with an independent argument.

This chapter is structured as follows.

Section 5.1. Definition of the notion of local weak limit.

Section 5.2. Statement of Lelarge's Theorem.

Section 5.3. Prototypical case for a proof of "$(G_n)_{n\in\mathbb{N}} \xrightarrow{\text{LWL}} \text{GWT}_{\Phi^A,\Phi^B}$". Explanation of how proofs of "$G_n \xrightarrow{\text{LWL}} \text{GWT}_{\Phi^A,\Phi^B}$" can be structured in general.

5.1. Defining the Local Weak Limit

A precise and accessible definition of the local weak limit can be found in lecture notes by Bordenave [Bor16, Chapter 3]. In the following, we shall be more brief, give more examples but omit many measure-theoretic details.

The Space $\mathcal{G}^*$ of Rooted Graphs. We consider the space $\mathcal{G}^*$ of connected rooted graphs with finite degree at every vertex but possibly infinitely many vertices. Except for the root, vertices are unlabeled and therefore indistinguishable.

A Metric on $\mathcal{G}^*$. For $G \in \mathcal{G}^*$ and $t \in \mathbb{N}_0$ let $(G)_t$ be the rooted subgraph of G induced by vertices with hop-distance at most t from the root of G.
The distance of $G_1, G_2 \in \mathcal{G}^*$ is $d(G_1, G_2) = 1/(\hat{t}+1)$ where $\hat{t} = \max\{t \in \mathbb{N}_0 \cup \{\infty\} \mid (G_1)_t = (G_2)_t\}$. With this metric, $\mathcal{G}^*$ is complete. For instance, the Cauchy sequence $C_3, C_4, \ldots, C_n, \ldots \in \mathcal{G}^*$ of rooted circles with n vertices satisfies $\lim_{n\to\infty} C_n = C_\infty$ where C_∞ is the bi-infinite rooted path, see Figure 5.1. Indeed, for $3 \le n_1 < n_2 \le \infty$ we have $d(C_{n_1}, C_{n_2}) = 1/\lfloor n_1/2 \rfloor$.

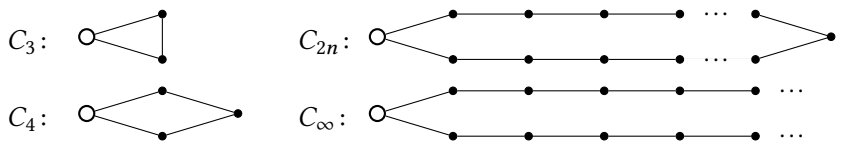


Figure 5.1 A sequence $C_3, C_4, \ldots \in \mathcal{G}^*$ of rooted graphs with $\lim_{n\to\infty} C_n = C_\infty \in \mathcal{G}^*$.

Via the metric, we also obtain the usual "ε-δ"-notion of continuous functions $f\colon \mathcal{G}^* \to \mathbb{R}$. Intuitively a continuous function can capture local properties expressed in the root's vicinity but is insensitive to distant changes. For instance $f_1(G) := \deg(\text{root}(G))$ is continuous, while, as seen in the previous example, $f_2(G) := \mathbb{1}[G \text{ is acyclic}]$ is not.

Probability Measures on $\mathcal{G}^*$. Introducing the Borel σ-algebra on $\mathcal{G}^*$, we consider random variables G with values in $\mathcal{G}^*$ and the corresponding probability measures μ_G.
An example important in the sequel is the *Galton-Watson tree* $\text{GWT}_{\text{Po}(\lambda)}$ with parameter $\lambda \in \mathbb{R}^+$. In this random, possibly infinite tree, every vertex v has a Poisson distributed number of children $X_v \sim \text{Po}(\lambda)$, see Figure 5.2. Formally, we could define $\text{GWT}_{\text{Po}(\lambda)}$ to be the unique tree in $\mathcal{G}^*$ where, in a breadth first traversal starting from the root, the i-th visited vertex has X_i children (if it exists) where $X_1, X_2, \ldots \sim \text{Po}(\lambda)$ are i.i.d. random variables. It is a classical result that $\text{GWT}_{\text{Po}(\lambda)}$ is almost surely finite for $\lambda \le 1$.

Weak Convergence of Measures on $\mathcal{G}^*$. Having a topology and σ-algebra on $\mathcal{G}^*$, we can define *weak convergence* of measures. We say a sequence $(\mu_n)_{n\in\mathbb{N}}$ of measures on $\mathcal{G}^*$ has the measure μ as *weak limit* and write $(\mu_n)_{n\in\mathbb{N}} \rightsquigarrow \mu$ if for every bounded continuous function $f\colon \mathcal{G}^* \to \mathbb{R}$ we have $\lim_{n\to\infty} \int f \,\mathrm{d}\mu_n = \int f \,\mathrm{d}\mu$. Intuitively, for every *local* property the probability under μ_n is asymptotically equal to the probability under μ.

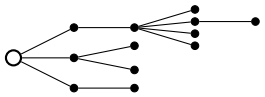


Figure 5.2 A possible outcome of a Galton-Watson tree $\mathrm{GWT}_{\mathrm{Po}(\lambda)}$. The probability for this outcome is $12p_0^7p_1^3p_2p_3p_4$ where $p_i = \Pr[\mathrm{Po}(\lambda) = i] = \mathrm{e}^{-\lambda}\frac{\lambda^i}{i!}$. The factor of 12 is needed because 12 distinct trees with ordered children are isomorphic to this tree.

We give a very simple example illustrated in Figure 5.3. Let T_n be the random binary search tree obtained by inserting the keys $\{1, \dots, n\}$ in random order into an initially empty binary search tree. Let $T_n^* \in \mathcal{G}^*$ be the equivalence class of T_n (forgetting keys and child orderings) and $\mu_{T_n^*}$ the corresponding distribution. Finally, let $\hat{T} \in \mathcal{G}^*$ be the infinite complete binary tree and $\delta_{\hat{T}}$ the corresponding Dirac measure. It is easy to see that $(\mu_{T_n^*})_{n\in\mathbb{N}} \rightsquigarrow \delta_{\hat{T}}$. We also write $T_n^* \rightsquigarrow \hat{T}$. Intuitively this just means that for any constant k the probability that the k-th level of T_n is full converges to 1.

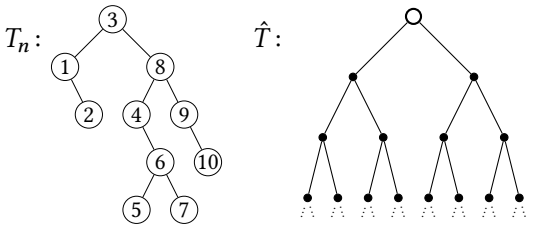


Figure 5.3 Random binary search tree T_n obtained by inserting the keys $1, \dots, n$ in random order, here possibly $3, 8, 9, 1, 4, 6, 7, 2, 10, 5$ with $n = 10$.
The corresponding random tree $T_n^* \in \mathcal{G}^*$ converges weakly to the complete binary tree $\hat{T}$.

Weak Convergence of Randomly Rooted Graphs. To capture the local characteristics of an *unrooted* graph G, we select $\circ \leftarrow V(G)$ uniformly at random and let $G(\circ) \in \mathcal{G}^*$ be the connected component of $\circ$ in G with $\circ$ as root. The distribution of $G(\circ)$ is denoted by $U(G)$. For a sequence $(G_n)_{n\in\mathbb{N}}$ of graphs, we often consider the weak limit of $U(G_n)$. We give a few examples that should be intuitively plausible.

Regular Graphs. If G_n is a random k-regular graph on vertex set $[2n]$ then we have $(G_n(\circ))_{n\in\mathbb{N}} \rightsquigarrow \hat{T}_k$ for the rooted k-regular tree $\hat{T}_k$. This is a way of asserting that G_n is unlikely to contain too many short cycles.

Erdős-Renyi Graphs. Let $G(n, \lambda/n)$ be the random graph with vertex set $[n]$ where each of the $\binom{n}{2}$ potential edges is independently included with probability λ/n. Then $(G(n, \lambda/n)(\circ))_{n\in\mathbb{N}} \rightsquigarrow \mathrm{GWT}_{\mathrm{Po}(\lambda)}$, see e.g. [Bor16]. The same holds if $G_{n,\lambda n/2}$ is a random graph with n vertices and exactly $\lambda n/2$ random edges.

Grimmett's Lemma [Gri80]. To give a non-obvious example, let G_n be a random tree on vertex set $[n]$.[1] Then $(G_n(\circ))_{n\in\mathbb{N}} \rightsquigarrow S$ where S is the *skeleton tree* consisting of an infinite path and an independent copy of the Galton-Watson tree GWT_1 at every vertex of the path, see Figure 5.4.

[1] To get the right distribution, it is important that G_n is a random *labelled* tree. The labels are forgotten when considering $G_n(\circ)$.

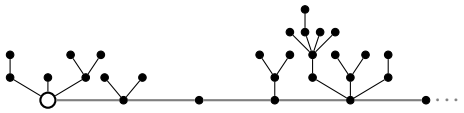


Figure 5.4 Sketch of the *skeleton tree* S obtained by sampling an infinite sequence $(T_i)_{i\in\mathbb{N}}$ of independent copies of GWT_1 (black) and connecting the roots on an infinite path (grey). The root of the skeleton tree is the root of T_1. The name "skeleton tree" is taken from [AS04].

Local Weak Limit. Making the transition from G_n to $G_n(\circ)$ implicit, we say a sequence of graphs $(G_n)_{n\in\mathbb{N}}$ has a distribution μ on $\mathcal{G}^*$ as *local weak limit* and write $(G_n)_{n\in\mathbb{N}} \xrightarrow{\text{LWL}} \mu$ for $(U(G_n))_{n\in\mathbb{N}} \rightsquigarrow \mu$. We also write $(G_n)_{n\in\mathbb{N}} \xrightarrow{\text{LWL}} G$ if $G \sim \mu$.
An important subtlety is that the notion treats $(G_n)_{n\in\mathbb{N}}$ as a fixed sequence. If $(G_n)_{n\in\mathbb{N}}$ is a random sequence then $\{G_n \xrightarrow{\text{LWL}} \mu\}$ is a random event. Due to Kolmogorov's zero-one law, it has probability 0 or 1 if $(G_n)_{n\in\mathbb{N}}$ is an independent family, and we could claim "$(G_n)_{n\in\mathbb{N}} \xrightarrow{\text{LWL}} \mu$ almost surely". In addition to the weak convergence of the *average* measures $(\mathbb{E}_{G_n}[U(G_n)])_{n\in\mathbb{N}} \rightsquigarrow \mu$ this implies that the random measures $U(G_n)$ (random over the choice of G_n) are tightly concentrated around their mean $\mathbb{E}_{G_n}[U(G_n)]$. This should be become clearer in Section 5.3.

Extension to Weighted Graphs. All notions effortlessly carry over to the case where graphs have vertex and edge weights in $\mathbb{N}$. Real weights can also be considered, see, e.g. [AS04], but natural weights suffice for our purposes.

5.2. The Objective Method and Lelarge's Theorem

To quote Aldous and Steele who coined the term "objective method":

> A characteristic element of the method is that it often calls for one to introduce a new, infinite, probabilistic *object* whose *local properties* inform us about the *limiting properties* of a sequence of finite problems. [AS04]

Lelarge's Theorem is an exemplification of the method, where the finite problems are the sizes $(M(G_n))_{n\in\mathbb{N}}$ of (generalised) maximum matchings in finite graphs $(G_n)_{n\in\mathbb{N}}$ with limiting property $\lim_{n\to\infty} M(G_n)/|E_n|$. The limiting object is the local weak limit of $(G_n)_{n\in\mathbb{N}}$, assuming it exists.

Before we can state the theorem, we have to import generalisations of matchings and Galton-Watson trees from [LLM13], namely *allocations* and *bipartite Galton-Watson trees*.

Allocations. Let $G = (A, B, E, \eta, \zeta)$ be a bipartite graph with vertex set $V = A \cup B$, edge set $E \subseteq A \times B$, vertex weights $\eta\colon V \to \mathbb{N}$ and edge weights $\zeta\colon E \to \mathbb{N}$. An *allocation* for G is a function $\sigma\colon E \to \mathbb{N}$ satisfying

$$\forall e \in E\colon \sigma(e) \le \zeta(e) \quad \text{and} \quad \forall v \in V\colon \sum_{e \ni v} \sigma(e) \le \eta(v).$$

The *size* of an allocation is $|\sigma| = \sum_{e\in E} \sigma(e)$ and $M(G)$ is the largest size of an allocation for G. Note that if all vertex weights are 1 an allocation corresponds to a matching $\{e \in E \mid \sigma(e) = 1\}$.

Bipartite Galton-Watson Trees. For the following discussion, a *star* $(D, W, \{C_i\}_{i\in[D]})$ is a centre vertex of degree D and weight W attached to D *dangling edges* with weights given by the multiset $\{C_i \in \mathbb{N}\}_{i\in[D]}$ as shown in Figure 5.5 **(a)**. A dangling edge can be joined with a dangling edge of the same weight attached to a different star to form a weighted edge between the star centres.

We consider two distributions Φ^A and Φ^B on stars and corresponding random variables

$$(D^A, W^A, \{C_i^A\}_{i\in[D^A]}) \sim \Phi^A \quad \text{and} \quad (D^B, W^B, \{C_i^B\}_{i\in[D^B]}) \sim \Phi^B.$$

To sample a weighted bipartite graph $G = (A, B, E, \eta, \zeta)$ that reflects the distributions Φ^A and Φ^B for vertices from A and B, one could proceed as follows. Sample stars from Φ^B and Φ^A at a ratio of $\mathbb{E}[D^A]/\mathbb{E}[D^B]$. Now randomly match dangling edges of the A-stars to dangling edges of the B-stars with the same weight, and join them, see Figure 5.5 **(b)** and **(c)**. Since this requires that the same number of dangling edges of each weight is generated on both sides, we demand that

$$\forall c \in \mathbb{N}\colon \tfrac{1}{\mathbb{E}[D^A]}\,\mathbb{E}\left[|\{i \in [D^A] \mid C_i^A = c\}|\right] = \tfrac{1}{\mathbb{E}[D^B]}\,\mathbb{E}\left[|\{i \in [D^B] \mid C_i^B = c\}|\right]$$

and (in the finite case) condition on the event that the actual numbers (not just the expectations) match.

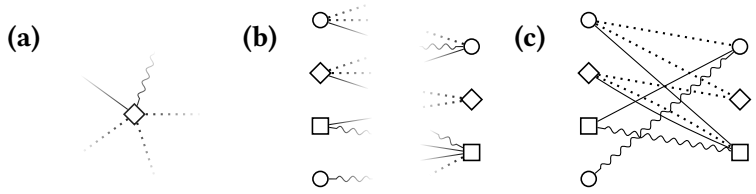


Figure 5.5 (a) A star with weights represented by drawing styles.
(b) Two collections of stars where each edge weight occurs equally often in both collections.
(c) Bipartite graph obtained from the collections from **(b)** by randomly joining dangling edges of equal weight.

The infinite analogue (and local weak limit) of G is the *bipartite Galton-Watson tree* $\mathrm{GWT}_{\Phi^A,\Phi^B}$ that we describe now. During construction, we keep track of whether we are on the A-side or B-side. With probability $\mathbb{E}[D^B]/(\mathbb{E}[D^A]+\mathbb{E}[D^B])$ the root of $\mathrm{GWT}_{\Phi^A,\Phi^B}$ is an A-vertex, otherwise a B-vertex, at the centre of a star sampled from Φ^A or Φ^B, respectively. We then grow a tree by joining random stars to dangling edges as long as dangling edges remain, possibly infinitely often.

When attaching a star $S = (d, w, \{c_1, \ldots, c_d\})$ to a dangling edge e of weight $c \in \mathbb{N}$ at an A-vertex, the selection of S is biased to each outcome in proportion to the number $|\{i \in [d] \mid c_i = c\}|$ of dangling edges of weight c occurring in that outcome. Note that S contains at least one dangling edge e' of weight c to be joined with e, and for convenience, we consider the distribution $\tilde{\Phi}^B(\cdot \mid c)$ describing only w and the $d-1$ *additional* dangling edges of S as follows ($\tilde{\Phi}^A(\cdot \mid c)$ is defined similarly):

$$\tilde{\Phi}^B(d-1, w, \{c_1, \ldots, c_{d-1}\} \mid c) = \tfrac{1}{Z}\Phi^B(d, w, \{c_1, \ldots, c_{d-1}, c\}) \cdot \Big(1 + \sum_{i=1}^{d-1} \mathbb{1}[c_i = c]\Big) \qquad (5.1)$$

where Z is the appropriate normalisation factor. We denote corresponding random variables by

$$(\tilde{D}^A, \tilde{W}^A, (\tilde{C}_i^A)_{i\in[\tilde{D}^A]}) \sim \tilde{\Phi}^A(\cdot \mid c) \quad \text{and} \quad (\tilde{D}^B, \tilde{W}^B, (\tilde{C}_i^B)_{i\in[\tilde{D}^B]}) \sim \tilde{\Phi}^B(\cdot \mid c).$$

Theorem Statement. With the additional notation $[z]_y^x := \max(\min(z,x),y)$ (where x and y can be omitted if they are $-\infty$ or ∞, respectively) we can now finally state Lelarge's Theorem in its general from.

▶ **Theorem 5.1** ([LLM13]). *Let $(G_n = (A_n, B_n, E_n, \eta_n, \zeta_n))_{n\in\mathbb{N}}$ be a sequence of weighted bipartite graphs with $(G_n)_{n\in\mathbb{N}} \xrightarrow{LWL} GWT_{\Phi^A,\Phi^B}$. If $\mathbb{E}[W^A]$ and $\mathbb{E}[W^B]$ are finite, then the limit $\mathcal{M}(\Phi^A,\Phi^B) := \lim_{n\to\infty} M(G_n)/|A_n|$ exists and equals*

$$\mathcal{M}(\Phi^A,\Phi^B) = \inf_{X,Y}\Bigg\{ \mathbb{E}\left[\min\{W^A, \sum_{i=1}^{D^A} X_i(C_i^A)\}\right] + \tfrac{\mathbb{E}[D^A]}{\mathbb{E}[D^B]}\,\mathbb{E}\left[\left[W^B - \sum_{i=1}^{D^B}\left[W^B - \sum_{j\neq i} Y_j(C_j^B)\right]_0^{C_i^B}\right]_0 \mathbb{1}[W^B < \sum_{i=1}^{D^B} C_i^B]\right]\Bigg\}$$

where for each $c\in\mathbb{N}$ the random variables $X_1(c), X_2(c), \dots$ and $Y_1(c), Y_2(c), \dots$ are independent copies of $X(c)$ and $Y(c)$, respectively, and the infimum is taken over all $(X(c),Y(c))_{c\in\mathbb{N}}$ satisfying the distributional equations:

$$Y(c) \overset{d}{=} \left[\tilde{W}^A - \sum_{i=1}^{\tilde{D}^A} X_i(\tilde{C}_i^A)\right]_0^c \qquad X(c) \overset{d}{=} \left[\tilde{W}^B - \sum_{i=1}^{\tilde{D}^B} Y_i(\tilde{C}_i^B)\right]_0^c.$$

Note that the right hand sides involve random variables drawn from $\tilde{\Phi}^A(\cdot \mid c)$ and $\tilde{\Phi}^B(\cdot \mid c)$ that depend on c.

The theorem looks quite intimidating, but with some intuition on a belief propagation process on $\mathrm{GWT}_{\Phi^A,\Phi^B}$ they emerge quite naturally, see [Lel12, Section 2] and Section 9.4 of this thesis.

Special Cases. We only need the feature of edge weights in the case of Theorem D and even there not as capacities restricting allocations but to distinguish two edge *types* for technical reasons. The numerical values of the weights will exceed all vertex weights. In this case, we can simplify the equation for $\mathcal{M}(\Phi^A,\Phi^B)$ to:

$$\mathcal{M}(\Phi^A,\Phi^B) = \inf_{X,Y}\Bigg\{ \mathbb{E}\left[\min\{W^A, \sum_{i=1}^{D^A} X_i(C_i^A)\}\right] + \tfrac{\mathbb{E}[D^A]}{\mathbb{E}[D^B]}\,\mathbb{E}\left[\left[W^B - \sum_{i=1}^{D^B}\left[W^B - \sum_{j\neq i} Y_j(C_j^B)\right]_0\right]_0 \mathbb{1}[D^B > 0]\right]\Bigg\} \tag{5.2}$$

$$\text{where} \quad Y(c) \overset{d}{=} \left[\tilde{W}^A - \sum_{i=1}^{\tilde{D}^A} X_i(\tilde{C}_i^A)\right]_0 \qquad X(c) \overset{d}{=} \left[\tilde{W}^B - \sum_{i=1}^{\tilde{D}^B} Y_i(\tilde{C}_i^B)\right]_0.$$

For Theorems A and F we do not need the edge weights at all, meaning all edge weights are also equal. Moreover, all vertices from A have weight 1 and degree $k \in \mathbb{N}$, and all vertices from B have weight $\ell \in \mathbb{N}$. In this case $\tilde{\Phi}^A$ and $\tilde{\Phi}^B$ have no parameter and the distributions X and Y on $[0,1]$ can simply be represented by probabilities $p = \Pr[X = 1]$ and $q = \Pr[Y = 1]$. We then obtain:

$$\mathcal{M}(\delta_{k,1,\tilde{1}}, \Phi^B) = \inf_{p,q\in[0,1]} \left\{ 1 - (1-p)^k + \tfrac{k}{\mathbb{E}[D^B]} \ell \Pr\left[\mathrm{Bin}(D^B, q) > \ell \right] \right\} \tag{5.3}$$
$$\text{where} \quad q = (1-p)^{k-1} \qquad p = \Pr[\mathrm{Bin}(\tilde{D}^B, q) < \ell].$$

5.3. Proving Local Weak Convergence

Our proofs require knowing the local weak limits of the incidence graphs $(G_n)_{n\in\mathbb{N}}$ of the hypergraph families $(H^k_{n,cn})_{n\in\mathbb{N}}$ and $(H^k_{n,n/(1+\theta),cn})_{n\in\mathbb{N}}$ as well as the family $(\hat{W}^{k,\ell}_{n,cn})_{n\in\mathbb{N}}$ from Chapter 9 and the family $(\widetilde{F}(n,k,c,z))_{n\in\mathbb{N}}$ from Chapter 6.

In all four cases, the claimed convergence is intuitively plausible, but a full-fledged proof would nevertheless have to go through lengthy arguments with only minor variations. A *general* theorem that handles all cases simultaneously would likely be annoyingly technical. Unhappy with either option, the author opted for an easy way out: **(i)** Provide a high-level view of the proof for the *simplest* case. **(ii)** For each of the other cases (considered in their respective chapters) outline why none of the key steps of the proof breaks.

Prototypical Case: The Fully Random Hypergraph. Let $G_n = (A_n, B_n, E_n)$ be the bipartite incidence graph of $H^k_{n,cn}$, where A_n and B_n correspond to hyperedges and vertices of $H^k_{n,cn}$, respectively. Moreover, let $\mathrm{GWT}_{\delta_k,\mathrm{Po}(ck)}$ be the unweighted bipartite Galton-Watson tree with constant degree k at A-vertices and Poisson distributed degree at B-vertices.

▶ **Theorem 5.2** ([Kim06]). *Let $k \in \mathbb{N}$ and $c \in \mathbb{R}^+$ be constants. Then almost surely over the randomness in $(H^k_{n,cn})_{n\in\mathbb{N}}$, we have $(G_n)_{n\in\mathbb{N}} \xrightarrow{LWL} GWT_{\delta_k,\mathrm{Po}(ck)}$.*

The proof is standard and partly implicit in [Kim06]. We heavily lean on the structure found in [Bor16, Section 3] and [Lec13], but gloss over many details. For a full account, we recommend [Bor16] where 7 pages are spent to prove a similar claim.

Proof. Let us write $T = \mathrm{GWT}_{\delta_k,\mathrm{Po}(ck)}$ and μ for the distribution of T. Then the claim can be decomposed into the claim $(\mathbb{E}_{G_n} U(G_n))_{n\in\mathbb{N}} \rightsquigarrow \mu$ that the measures $U(G_n)$ *averaged* over the choices for G_n have weak limit μ and the claim that $U(G_n)$ is highly concentrated around its mean $\mathbb{E}_{G_n} U(G_n)$. With generic arguments involving the Portmanteau Theorem, the proof of Theorem 5.2 is thus reduced to showing that for all $t \in \mathbb{N}$ and all trees $H \in \mathcal{G}^*$ with $(H)_t = H$ (in H all vertices have distance at most t from the root):

(i) $\lim_{n\to\infty} \Pr_{G_n,\circ}[(G_n(\circ))_t = H] = \Pr_T[(T)_t = H]$

(ii) $\lim_{n\to\infty} \left| \Pr_\circ[(G_n(\circ))_t = H] - \mathbb{E}_{G_n}\left[\Pr_\circ[(G_n(\circ))_t = H] \right] \right| = 0$ almost surely$_{(G_n)_{n\in\mathbb{N}}}$.

Indices at "Pr", "$\mathbb{E}$" and "almost surely" indicate the scope of the random variables.

To prove **(i)** we reveal a random coupling between $G_n(\circ)$ and T (i.e. embed them in a common probability space without affecting their individual distributions) such that when we perform breadth first searches (BFSs) on both graphs up to depth t, we will have explored

graphs isomorphic to H with asymptotically equal probability. First note that for both $G_n(\circ)$ and T, the root is a vertex from A with probability $\frac{c}{1+c}$ and from B with probability $\frac{1}{1+c}$, so the root types can be made to always agree in a coupling. From now on, the node types alternate from level to level in both $G_n(\circ)$ and T by construction. When the BFS visits a vertex v, we assume it first reveals the number and then the identities of the *successors* of v, defined as the neighbours of v except for the *parent vertex* of v from which v was discovered (assuming v is not the root). Note that the number of successors of A-vertices is deterministic. A root vertex from A has k successors, other vertices from A have $k-1$ successors (in the case of T one may check Equation (5.1) to see that $\Phi^A = \delta_k$ implies $\tilde{\Phi}^A = \delta_{k-1}$). A root vertex from B has, in the case of T, degree $X \sim \mathrm{Po}(ck)$ and in the case of $G_n(\circ)$ degree $Y \sim \mathrm{Bin}(ckn, \frac{1}{n})$ since cn vertices from A each choose k (not necessarily distinct) neighbours from the n vertices from B independently and uniformly at random. Identifying a coupling between X and Y with $\Pr[X \neq Y] = O(1/n)$ is easy. Concerning successors for a non-root vertex $v \in B$, their number has, in the case of T distribution $\tilde{X} \sim \mathrm{Po}(ck)$ (check that $\Phi^B = \mathrm{Po}(\lambda)$ implies $\tilde{\Phi}^B = \mathrm{Po}(\lambda)$ via Equation (5.1)). The case of $G_n(\circ)$ is slightly more complicated. We may assume that the BFS has so far produced results in agreement with H. In that case, let i be the number of edges from $G_n(\circ)$ that are already revealed and j the number of B-vertices in $G_n(\circ)$ that were already visited. The number of successors of v then has distribution $\tilde{Y} \sim \mathrm{Bin}(ckn - i, \frac{1}{n-j})$. Since i and j are both bounded by the constant $|V(H)|$, it is again quite easy to see that the coupling can be extended such that $\Pr[\tilde{X} \neq \tilde{Y}] = O(1/n)$. Concerning the *identities* of successors, for $G_n(\circ)$ we may find that a successor has already been discovered from a different vertex revealing a cycle. However, the probability for this is at most $|V(H)|/(\min(1,c)n) = O(1/n)$ in each case. Extending the coupling along the finite sequence of events that reveal $(G_n(\circ))_t = (T)_t = H$ we get $\Pr\left[\mathbb{1}[(G_n(\circ))_t = H] \neq \mathbb{1}[(T)_t = H]\right] = O(1/n)$ which proves **(i)**.

To prove **(ii)**, we consider $F(G_n) = |\{v \in V(G_n) \mid (G_n(v))_t = H\}|$. Intuitively, this is the number of copies of H in G_n. While **(i)** showed that $F(G_n)/n$ has the right *expectation*, namely $\Pr[(T)_t = H]$, we now show that $F(G_n)$ only strays by $o(n)$ from this expectation. Recall that G_n is obtained from $\Theta(n)$ independent random variables $Z_1, \ldots, Z_{ckn}$, namely the neighbours chosen by the cn vertices in A. We may therefore write $F(G_n) = F(Z_1, \ldots, Z_{ckn})$. Let d be the maximum degree of H. If v is a vertex with $(G_n(v))_t = H$ then for any edge e with distance at most t from v (i.e. for each edge contained in $(G_n(v))_t$) we say e *affects* v. Since this means there is a path from e to v traversing at most $t-1$ vertices each of degree at most d, any edge can affect at most $2d^{t-1}$ vertices. This shows that changing a single variable Z_i affects F by at most a constant c_H depending on H. From the Azuma-Hoeffding inequality, we obtain that $\Pr[F(G_n) - \mathbb{E}[F(G_n)] > s] \leq \exp(-O(s^2/nc_H^2))$. For $s = n^{2/3}$ we find that $\sum_{n\in\mathbb{N}} \Pr[F(G_n) - \mathbb{E}[F(G_n)] \geq n^{2/3}] = O(1)$ and the Borel-Cantelli Lemma implies that $\{F(G_n) - \mathbb{E}[F(G_n)] \leq n^{2/3}\}$ holds for all but a finite number of $n \in \mathbb{N}$ almost surely. This implies **(ii)** and concludes the proof of Theorem 5.2. ◀

Part III.

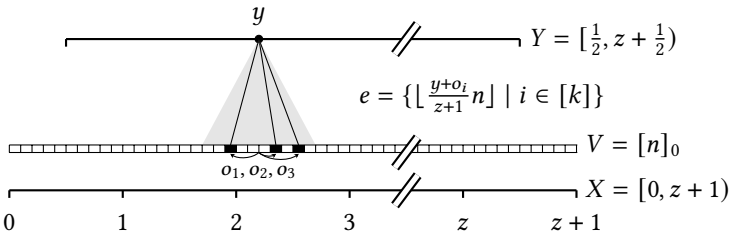
Proofs

Relevant Definition and Theorem for Chapter 6
(Originally on Pages 15 and 16)

► **Definition 2.4** (Spatially Coupled Hypergraph Family). *For $k \in \mathbb{N}$, $z, c \in \mathbb{R}_0^+$ and $n \in \mathbb{N}$, let $F_n = F(n,k,c,z)$ be the random k-uniform hypergraph with vertex set $V = [n]_0$ and edge set E of size $m = \lfloor cn\frac{z}{z+1} \rfloor$. Each edge $e \in E$ is independently obtained as*

$$e = \left\{ \left\lfloor \tfrac{y+o_i}{z+1} n \right\rfloor \mid i \in [k] \right\} \text{ where } y \leftarrow\!\!\boxed{\cdot\cdot} \ [\tfrac{1}{2}, z+\tfrac{1}{2}), \quad o_1, \dots, o_k \leftarrow\!\!\boxed{\cdot\cdot} \ [-\tfrac{1}{2}, \tfrac{1}{2}].$$

We call $y \in Y := [\frac{1}{2}, z+\frac{1}{2})$ the position of e and $\frac{v(z+1)}{n} \in X = [0, z+1)$ the position of $v \in V$.



► **Theorem A.** *Let $k, \ell \in \mathbb{N}$, with $k \geq 2$ and $k + \ell \geq 4$. Then we have:*
(i) $c < c^*_{k,\ell} \Rightarrow \forall z \in \mathbb{R}^+ \colon \Pr[F_n \text{ is } \ell\text{-peelable}] \xrightarrow{n\to\infty} 1$.
(ii) $c > c^*_{k,\ell} \Rightarrow \exists z^* \in \mathbb{R}^+ \colon \forall z \geq z^* \colon \Pr[F_n \text{ is } \ell\text{-orientable}] \xrightarrow{n\to\infty} 0$.

Corollaries (Originally on Pages 35, 40, 44 and 47)

► **Theorem A1.** *Let $k, \ell \in \mathbb{N}, k \geq 2, \ell \geq 1$, $(k,\ell) \neq (2,1)$ and $z \in \mathbb{R}^+$. Consider k-ary cuckoo hashing with buckets of size ℓ using spatially coupling with parameter $z \in \mathbb{R}^+$. There exists an $\varepsilon = \varepsilon(z)$ with $\varepsilon(z) \xrightarrow{z\to\infty} 0$ such that the load threshold of the scheme is at most $c^*_{k,\ell}/\ell - \varepsilon$. Moreover, at loads $c < c^*_{k,\ell}/\ell - \varepsilon$* **construct** *can be carried out successfully by a peeling algorithm in time $O(kn)$ whp.*

► **Theorem A2.** *Let $R = \mathbb{Z}_d$ for $d \in \mathbb{N}$, $k \geq 3$, $c < c^*_{k,1}$ and $\varepsilon := \frac{1-c}{c}$. When using $H = \hat{F}^{k,1}_{m/c,m}$ from Theorem A' to build a retrieval data structure for $f \colon S \to R$ with $|S| = m$ as in Lemma 3.1, then the following holds.*
(i) *One trial of* **construct** *succeeds whp and has expected running time $O(km)$.*
(ii) *The resulting data structure D_f needs space to store $z \in R^{(1+\varepsilon)m}$, plus $O(\log m)$ bits whp.*
(iii) *An* **eval** *takes time $O(k)$ and causes k cache misses.*

► **Theorem A3.** *For any $\alpha > 2/c^*_{3,1} \approx 2.18$ a simple algorithm computes for $S \subseteq \mathcal{U}$ of size m a* MPHF *$p \colon S \to [m]$ occupying $\alpha m + o(m)$ bits of memory in expected time $O(m)$. Evaluating p takes $O(1)$ time and causes 3 cache misses.*

► **Theorem A4.** *In the above setting, let $k \in \mathbb{N}$ and $c < c^*_{k,1}$. If $h_1, \dots, h_k$ are such that $H \stackrel{d}{=} \hat{F}^{k,1}_{n,cn}$, then* **listMembers** *succeeds in time $O(kn)$ whp.*

6. Peeling Close to the Orientability Threshold – Spatial Coupling in Hashing-Based Data Structures

The goal of this chapter is to prove Theorem A, restated on the facing page. A precursor to this result was published in [DW19b]. Thanks to a valuable comment by Djamal Belazzougui, we have since learned that the key underlying idea is actually well-known in coding theory (see also Section 4.5) and that the existing theorems can be applied in our data structure setting. The presentation here follows the correspondingly revised account from [Wal20b].

6.1. Threshold Saturation by Spatial Coupling

The hypergraph F_n arises by "spatial coupling" of $H^k_{n,cn}$ along the "coupling dimension" $X = [0, z+1)$. In the peeling process on F_n, vertices with a position close to the borders 0 or $z+1$ tend to be deleted early on, while many vertices in the denser, central parts remain. But gradually, deletions at the border "expose" vertices further on the inside and the whole hypergraph "erodes" from the outside in. The effect is shown in Figure 6.1. This does not happen in the more symmetric construction when X is glued into a circle, i.e. if for all $\varepsilon \in [0, 1)$ the positions ε and $z + \varepsilon$ are identified.

The authors of [Krz+12; KRU13] liken the phenomenon to water that is *super-cooled* to below 0°C in a smooth container. It will not freeze unless a *nucleus* for crystallization is introduced. Once this is done all water crystallizes quickly, starting from that nucleus. In this sense, a nucleus raises the temperature at which water freezes. In our construction, the borders play the role of a nucleus and get the peeling process (rather than the crystallization process) started.

When introducing a linear geometry in the way we did, the 1-peelability threshold of the resulting (*coupled*) hypergraph family approaches the solvability thresholds of the underlying *uncoupled* construction, in a wide range of cases. This phenomenon is known as *threshold saturation*.

We leave a summary of the field to the experts [KRU15; KRU13]. Put briefly, the phenomenon was discovered in the form of convolutional codes [FZ99], then rigorously explained, first in a special case [KRU10], then more generally [KRU13], later accounting for continuous coupling dimensions (and even multiple dimensions) [KRU15], a form we will exploit in this chapter.

6.2. The Peeling Process and Idealised Peeling Operators

This section mimics and expands upon similar discussions in the literature where the existence or sizes of cores of random hypergraphs are analysed, see for instance [PSW96;

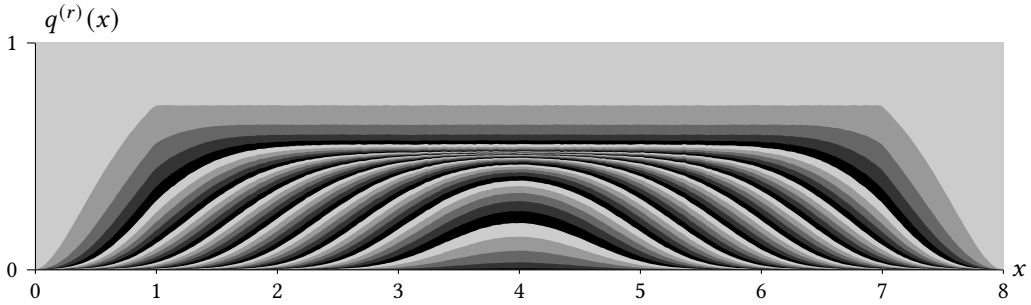


Figure 6.1 The "layers" of a hypergraph sampled according to Definition 2.4 with parameters $k = 3$, $c = 0.85$, $z = 7$ and $m = 10^8$.
It happens to be peelable in 48 rounds, that are assigned shades of gray in a round-robin fashion. For each round $r \in \{0, \dots, 48\}$ and each position $x \in X = [0, 8]$, the picture shows the fraction $q^{(r)}(x) \in [0, 1]$ of vertices close to x that "survive" r rounds of the parallel peeling process.

Coo04; Mol04; Mol05; Die+10] and [MM09, Ch. 18].

We examine how the probabilities for vertices of F_n to "survive" $r \in \mathbb{N}$ rounds of peeling change from one round to the next. In the classical setting, this could be described by a function, mapping the old survival probability to the new one [Mol05]. In our case, however, there are distinct survival probabilities $q(x)$ depending on the position x of the vertex. Thus we need a corresponding operator $\hat{\mathbf{P}}$ that acts on such functions q.

We almost always suppress k, ℓ, c, z in notation outside of definitions. Big-O notation refers to $n \to \infty$ while k, ℓ, c, z are constant.

Consider the parallel peeling process $\mathrm{peel}(F_n, \ell)$ on $F_n = F(n, k, c, z)$. In each *round* of $\mathrm{peel}(F_n)$, all vertices of degree at most ℓ are determined and then deleted simultaneously. Deleting a vertex implicitly deletes all incident hyperedges. We also define the *r-round rooted peeling process* $\mathrm{peel}_{v,r}(F_n, \ell)$ for any vertex $v \in V$ and $r \in \mathbb{N}$. In round $1 \le r' \le r-1$ of $\mathrm{peel}_{v,r}(F_n)$, only vertices with distance $r - r'$ from v are considered for deletion. Moreover, in round r, the root vertex v is only deleted if it has degree at most $\ell - 1$, not if it has degree ℓ.

For any vertex position $x \in X = [0, z + 1)$ and $r \in \mathbb{N}$ we let $q^{(r)}(x) = q^{(r)}(x, n, k, \ell, c, z)$ be the probability that the vertex $v = \lfloor \frac{x}{z+1} n \rfloor$ survives $\mathrm{peel}_{v,r}(F_n)$, i.e. is not deleted. It is convenient to define $q^{(0)}(x) = 1$ for all $x \in X$, i.e. every vertex survives the "0-round peeling process". To get an intuition for how $q^{(r)}(x)$ evolves with r, consider Figure 6.1 Even though $q^{(r)}$ is discrete in x by definition, we will later see that it has a continuous limit for $n \to \infty$.

Whether a vertex v at position x survives $\mathrm{peel}_{v,r}$ is a function of its r-neighbourhood $F_n(x, r)$, i.e. the subhypergraph of F_n that can be reached from v by traversing at most r hyperedges.

It is natural to consider the distributional limit of $F_n(x, r)$ to get a grip on $q^{(r)}(x)$. In the spirit of the objective method [AS04], we identify a (possibly infinite) random tree T_x that captures the local characteristics of $F_n(x, r)$ for $n \to \infty$. In the following $\mathrm{Po}(\lambda)$ refers to the Poisson distribution with mean $\lambda \in \mathbb{R}^+$.

▶ **Definition 6.1** (Limiting Tree). *Let $k \in \mathbb{N}$, $c, z \in \mathbb{R}^+$, $X = [0, z + 1)$, $Y = [\frac{1}{2}, z + \frac{1}{2})$ and $x \in X$. The random (possibly infinite) hypertree $T_x = T_x(k, c, z)$ is distributed as follows.*

T_x has a root vertex $\mathrm{root}(T_x)$ *at position x, which for $Y_x \coloneqq [x-\frac{1}{2}, x+\frac{1}{2}] \cap Y$ has $d_x \sim \mathrm{Po}(ck|Y_x|)$* child hyperedges *with positions uniformly distributed in Y_x.*[1] *Each child hyperedge at position y is incident to $k-1$ (fresh)* child vertices *of its own, each with a uniformly random position $x' \in [y-\frac{1}{2}, y+\frac{1}{2}]$. The sub-hypertree at such a child vertex at position x' is distributed recursively (and independently of its sibling-subtrees) according to $T_{x'}$.*

For $x \in X$ and $r \in \mathbb{N}$, let $F_n(x,r)$ and $T_x(r)$ denote the r-neighbourhoods of vertex $v = \lfloor \frac{x}{z+1} n \rfloor$ in F_n and $\mathrm{root}(T_x)$ in T_x, respectively. In the following, H is an arbitrary fixed rooted hypergraph and equality of hypergraphs indicates a root-preserving isomorphism.

▶ **Lemma 6.2.** $\forall x \in X, r \in \mathbb{N}, H\colon \lim_{n\to\infty} \Pr[F_n(x,r) = H] = \Pr[T_x(r) = H]$.

When put in terms of the notions from Chapter 5, this claims the weak convergence $(F_n(x,r))_{n\in\mathbb{N}} \rightsquigarrow T_x(r)$ for all $x \in X$ and $r \in \mathbb{N}$. We could even make the stronger claim that $(F_n)_{n\in\mathbb{N}} \xrightarrow{\text{LWL}} T_x$ where T_x is obtained by first sampling $x \leftarrow X$ and then sampling a corresponding random tree.

While the more light-weight statement is sufficient, the proof is similar to the first part of the proof of Theorem 5.2. We omit a few details because the arguments are standard.

Sketch of Proof. We construct for fixed r, x and H a random coupling[2] between $F_n(x,r)$ and $T_x(r)$ such that the symmetric difference between the events $\{F_n(x,r) = H\}$ and $\{T_x(r) = H\}$ has probability $o(1)$. We do so inductively, by following a sequence of events. The i-th event expresses, firstly, that $F_n(x,r)$ and $T_x(r)$ agree with H concerning the first i rounds of a breadth-first search traversal and, secondly, that corresponding active vertices in $F_n(x,r)$ and $T_x(r)$ have positions with distance $O(1/n)$.

For the first step, consider the root v of $F_n(x,r)$. By construction, any hyperedge containing v must have a position $y \in [x-\frac{1}{2}, x+\frac{1}{2}]$. For $x \in [0,1)$ or $x \in [z, z+1)$ the potential positions are further restricted by the upper and lower bounds on hyperedge positions, i.e. we have $y \in Y_x \coloneqq [x-\frac{1}{2}, x+\frac{1}{2}] \cap Y$. In order for a random hyperedge e to contain v, two things have to work out:

(1) The position of e must fall within Y_x. The probability for this is $|Y_x|/|Y| = |Y_x|/z$.

(2) At least one of the k incidences of e must turn out to be to v. The probability for this is $1-(1-\frac{z+1}{n})^k$.

With $cn\frac{z}{z+1}$ hyperedges in total, we obtain a binomial distribution $\deg(v) \sim \mathrm{Bin}\left(cn\frac{z}{z+1}, |Y_x|/z(1-(1-\frac{z+1}{n})^k)\right)$. This distribution converges, for $n \to \infty$, to $\mathrm{Po}(ck|Y_x|)$, which is the distribution of $\deg(\mathrm{root}(T_x))$. The positions of the neighbours of v are uniformly distributed in the discrete set $|Y_x| \cap (\frac{z+1}{n}\mathbb{Z})$, the positions of the neighbours of $\mathrm{root}(T_x)$ uniformly in the interval $|Y_x|$. It should now be easy to see how a coupling between $F_n(x,1)$ and $T_x(1)$ could look like.

---

[1] In other words: The positions of the child hyperedges are a Poisson point field on Y_x with intensity ck. By $|I|$ for an interval $I = [a,b]$ we mean $b-a$.
Note also that the position is now a property of a vertex, not an identifying feature. Possibly (though with probability 0) the tree T_x may contain several vertices with the same position.

[2] There is no relation to the term spatial coupling. We refer to the standard technique where several random variables are realised on the same probability space.

There are three complications when continuing the argument: **(i)** The discrepancies between vertex positions of $F_n(x,r)$ and $T_x(1)$ need to be kept in check. **(ii)** $F_n(x,r)$ may contain cycles[3]. **(iii)** There are slight dependencies between vertex degrees in $F_n(x,r)$. It should be intuitively plausible that these problems vanish in the limit in a similar way as in Theorem 5.2. ◀

We now consider the idealised peeling processes $(\mathrm{peel}_{\mathrm{root}(T_x),r}(T_x))_{x\in X}$. Their survival probabilities are easier to analyse than those of $\mathrm{peel}_{v,r}(F_n)$.

▶ **Lemma 6.3.** *Let $r \in \mathbb{N}_0$ be constant and $q_T^{(r)}(x) = q_T^{(r)}(x,k,\ell,c,z)$ be the probability that* $\mathrm{root}(T_x)$ *survives* $\mathrm{peel}_{\mathrm{root}(T_x),r}(T_x,\ell)$ *for $x \in X$. Then*

$$q_T^{(r+1)}(x) = Q\left(ck \int_{[x-\frac{1}{2},x+\frac{1}{2}]\cap Y} \left(\int_{y-\frac{1}{2}}^{y+\frac{1}{2}} q_T^{(r)}(x')dx'\right)^{k-1} dy,\ \ \ell\right) \qquad \text{for } x \in X.$$

where $Q(\lambda,\ell) = 1 - \sum_{i<\ell} \frac{\lambda^i}{i!} = \Pr[\mathrm{Po}(\lambda) \geq \ell]$, the latter term slightly abusing notation.

Proof. Let $x \in X$ and $v = \mathrm{root}(T_x)$. Assume $y \in [x-\frac{1}{2}, x+\frac{1}{2}] \cap Y$ is the type of some hyperedge e incident to v. Hyperedge e survives r rounds of $\mathrm{peel}_{v,r+1}(T_x)$ if and only if all of its incident vertices survive these r rounds. Since v itself may only be deleted in round $r+1$, the relevant vertices are the $k-1$ child vertices $w_1,\dots,w_{k-1}$ with positions uniformly distributed in $[y-\frac{1}{2}, y+\frac{1}{2}]$. Let W_i be the subtree rooted at w_i for $1 \leq i < k$. Consider the peeling process $\mathrm{peel}_{w_i,r}(W_i)$. Assume the process deletes w_i in round r, meaning w_i has degree at most $\ell-1$ at the start of round r. Then w_i has degree at most ℓ at the start of round r in $\mathrm{peel}_{v,r+1}(T_x)$, meaning $\mathrm{peel}_{v,r+1}(T_x)$ deletes e in round r. Conversely, if none of $\mathrm{peel}_{w_1,r}(W_1),\dots,\mathrm{peel}_{w_{k-1},r}(W_{k-1})$ delete their root vertex within r rounds, then $w_1,\dots,w_{k-1}$ have degree at least $\ell+1$ after round r of $\mathrm{peel}_{v,r+1}(T_x)$ and e survives round r of $\mathrm{peel}_{v,r+1}(T_x)$. Since the position of each w_i is independent and uniformly distributed in $[y-\frac{1}{2}, y+\frac{1}{2})$, the probability for e to survive is $p_y := (\int_{y-\frac{1}{2}}^{y+\frac{1}{2}} q_T^{(r)}(x')dx')^{k-1}$. Since the positions of the hyperedges incident to v are a Poisson point field on $[x-\frac{1}{2}, x+\frac{1}{2}] \cap Y$ with intensity ck, the number of incident hyperedges *surviving* round r of $\mathrm{peel}_{v,r+1}(T_x)$ has Poisson distribution with mean $\lambda := \int_{[x-\frac{1}{2},x+\frac{1}{2}]\cap Y} ckp_y dy$.

The claim now follows by observing that v survives $r+1$ rounds of $\mathrm{peel}_{v,r+1}(T_x)$ if it is incident to at least ℓ hyperedges surviving r rounds. The probability for this is $Q(\lambda,\ell)$. ◀

For convenience, we define the operator $\mathbf{P} = \mathbf{P}(k,\ell,c,z)$, which maps any (measurable[4]) $q\colon X \to [0,1]$ to $\mathbf{P}q\colon X \to [0,1]$ with

$$(\mathbf{P}q)(x) = Q\left(ck \int_{[x-\frac{1}{2},x+\frac{1}{2}]\cap Y} \left(\int_{y-\frac{1}{2}}^{y+\frac{1}{2}} q(x')dx'\right)^{k-1} dy,\ \ \ell\right) \quad \text{for } x \in X.$$

Together Lemmas 6.2 and 6.3 imply that $\mathbf{P}$ can be used to approximate survival probabilities.

---

[3] This can actually already occur for $r = 1$.

[4] All functions that play a role in our analysis are measurable. We refrain from pointing this out from now on.

▶ **Corollary 6.4.** *Let $r \in \mathbb{N}_0$ be constant. Then for all $x \in X$*

$$\mathbf{P}^r q^{(0)}(x) \overset{\text{def}}{=} \mathbf{P}^r q_T^{(0)}(x) \overset{\text{Lem 6.3}}{=} q_T^{(r)}(x) \overset{\text{Lem 6.2}}{=} q^{(r)}(x) \pm o(1).$$

To obtain *upper* bounds on survival probabilities, we may remove the awkward restriction "$\cap Y$" in the definition of $\mathbf{P}$. We define $\hat{\mathbf{P}} = \hat{\mathbf{P}}(k, \ell, c)$ as mapping any $q\colon \mathbb{R} \to [0,1]$ to $\hat{\mathbf{P}}q\colon \mathbb{R} \to [0,1]$ as follows (soon to be rewritten using convolutions)

$$(\hat{\mathbf{P}}q)(x) = Q\left(ck \int_{x-\frac{1}{2}}^{x+\frac{1}{2}} \left(\int_{y-\frac{1}{2}}^{y+\frac{1}{2}} q(x')dx' \right)^{k-1} dy, \quad \ell \right) \quad \text{for } x \in \mathbb{R}.$$

Note that $\hat{\mathbf{P}}$ does not depend on z or n. To simplify notation, we assume that the old operator $\mathbf{P}$ also acts on functions $q\colon \mathbb{R} \to [0,1]$, ignoring $q(x)$ for $x \notin X$, and producing $\mathbf{P}q\colon \mathbb{R} \to [0,1]$ with $\mathbf{P}q(x) = 0$ for $x \notin X$. We also extend $q^{(0)}$ to be $\mathbb{1}[x \in X]\colon \mathbb{R} \to [0,1]$, i.e. the characteristic function on X, essentially introducing vertices at positions $x \notin X$ which are, however, already deleted with probability 1 before the first round begins. Note that while $q^{(r)}(x)$ and $q_T^{(r)}(x)$ are by definition non-increasing in r, this is not the case for $(\hat{\mathbf{P}}^r q^{(0)})(x)$. For instance, $\hat{\mathbf{P}}^r q^{(0)}$ has support $(-r, z+1+r)$, which grows with r.[5] The following lemma lists a few easily verified properties of $\hat{\mathbf{P}}$. All inequalities between functions should be interpreted point-wise.

▶ **Lemma 6.5.** **(i)** $\forall q\colon \mathbb{R} \to [0,1]:\ \mathbf{P}q \le \hat{\mathbf{P}}q$.
(ii) *$\mathbf{P}$ and $\hat{\mathbf{P}}$ are monotonic, i.e.* $\forall q, q'\colon \mathbb{R} \to [0,1]:\ q \le q' \Rightarrow \mathbf{P}q \le \mathbf{P}q' \wedge \hat{\mathbf{P}}q \le \hat{\mathbf{P}}q'$.
(iii) *$\mathbf{P}$ and $\hat{\mathbf{P}}$ are continuous, i.e. pointwise convergence of $(q_i)_{i\in\mathbb{N}}$ to q^* implies pointwise convergence of $(\mathbf{P}q_i)_{i\in\mathbb{N}}$ and $(\hat{\mathbf{P}}q_i)_{i\in\mathbb{N}}$ to $\mathbf{P}q^*$ and $\hat{\mathbf{P}}q^*$, respectively.*

6.3. Analysis of Iterated Peeling

The goal of this section is to prove the following proposition.

▶ **Proposition 6.6.**
(i) *For $c < c^*_{k,\ell}$ and any $z \in \mathbb{R}^+$, we have $(\mathbf{P}^r q_0)(x) \overset{r\to\infty}{\longrightarrow} 0$ for all $x \in X$.*
(ii) *For $c > c^*_{k,\ell}$ and large z, we have $(\mathbf{P}^r q_0)(x) \overset{r\to\infty}{\longrightarrow} q^*(x)$ for all $x \in X$ and some $q^* \neq 0$.*

The intuition is that for $c > c^*_{k,\ell}$ the peeling process gets stuck, while for $c < c^*_{k,\ell}$ all vertices are eventually peeled.

Conveniently, iterations such as the one given by $\mathbf{P}$ and $\hat{\mathbf{P}}$ were extensively studied in a stunning paper by Kudekar, Richardson and Urbanke [KRU15]. For some initial function $f^{(0)}\colon \mathbb{R} \to [0,1]$ and non-decreasing functions $h_f, h_g\colon [0,1] \to [0,1]$ they study the sequence of functions

$$\begin{aligned} g^{(r)}(y) &\coloneqq h_g((f^{(r)} \otimes \omega)(y)) \\ f^{(r+1)}(x) &\coloneqq h_f((g^{(r)} \otimes \omega)(x)) \end{aligned} \tag{6.1}$$

[5] It is still possible to interpret $\hat{\mathbf{P}}^r q^{(0)}(x)$ as survival probabilities in more symmetric, extended versions $\hat{T}_x$ of the tree T_x, but we will not pursue this.

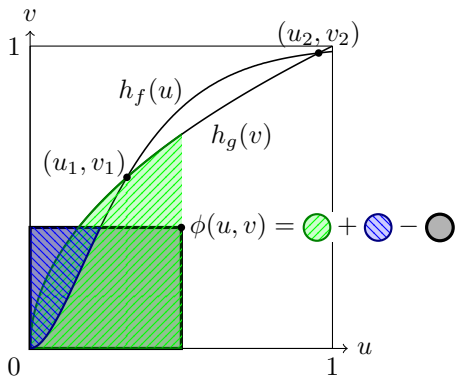


Figure 6.2 A plot of the curves $u \mapsto (u, h_f(u))$ and $v \mapsto (h_g(v), v)$ for $u, v \in [0,1]$ with $k = 3$, $\ell = 2$ and $c = c^*_{k,\ell}$. The three crossing points of the curves are the solutions $(0,0)$, (u_1, v_1) and (u_2, v_2) to Equation (6.3). The potential $\phi(u,v)$ can be visualised as the sum of three areas as shown. The significance of the threshold $c^*_{k,\ell}$ is that the two areas enclosed by the two curves have exactly the same size, or put differently, $\phi(u_2, v_2) = 0$.

where ω is an *averaging kernel*, i.e. an even non-negative function with integral 1 and $\otimes$ is the convolution operator. To apply the theory to our case, we use:

$$h_f(u) := Q(cku, \ell) \qquad h_g(v) := v^{k-1} \qquad \omega(x) = \mathbb{1}[|x| \le \frac{1}{2}] \tag{6.2}$$

With these substitutions the iteration (6.1) satisfies $\hat{\mathbf{P}} f^{(r)} = f^{(r+1)}$. If we force the functions $g^{(r)}$, $r \in \mathbb{N}$, to be zero outside of $Y = [\frac{1}{2}, z + \frac{1}{2})$ by replacing (6.1) with $g^{(r)}(y) := \min\{\mathbb{1}[y \in Y], h_g((f^{(r)} \otimes \omega)(y))\}$ we get the system with *two-sided termination*. In this case $\mathbf{P} f^{(r)} = f^{(r+1)}$. The system with *one-sided termination* is defined similarly with $Y = [\frac{1}{2}, \infty)$.

We remark that nothing in the following depends on the choice of ω.[6]

6.3.1. Unleashing Heavy Machinery from Coding Theory

We plan to delegate the proof of Proposition 6.6 to theorems from [KRU15]. For this, we need to examine the *potential* $\phi(u,v) = \phi(h_f, h_g, u, v)$ given as:

$$\phi(u,v) = \int_0^u h_g^{-1}(u')du' + \int_0^v h_f^{-1}(v')dv' - uv \qquad \text{for } 0 \le u \le h_g(1),\ 0 \le v \le h_f(1).$$

A visualisation is given in Figure 6.2. Consider the equation

$$(u,v) = (h_g(v), h_f(u)). \tag{6.3}$$

Clearly it has the *trivial solution* $(u,v) = (0,0)$. By monotonicity of h_g and h_f, any two solutions (u_1, v_1) and (u_2, v_2) are component-wise ordered. We write $(u_1, v_1) < (u_2, v_2)$ for $u_1 < u_2 \wedge v_1 < v_2$.

[6] There is a corresponding flexibility in Definition 2.4. Instead of a hyperedge at position y choosing its incident vertices uniformly at random from $[y - \frac{1}{2}, y + \frac{1}{2}]$, incidences can be chosen according to an almost arbitrary bounded density function that is symmetric around y. For details consider [KRU15, Definition 2].

▶ **Lemma 6.7.** **(i)** *Every local minimum (u,v) of ϕ is a solution to Equation* (6.3).
(ii) *If Equation* (6.3) *has at least one non-trivial solution, then the smallest non-trivial solution (u_1,v_1) has potential $\phi(u_1,v_1) > 0$.*
(iii) *Equation* (6.3) *has at most two non-trivial solutions.*
(iv) *For $c = c^*_{k,\ell}$ there is a non-trivial solution (u_2,v_2) of Equation* (6.3) *with $\phi(u_2,v_2) = 0$. In this case, $(0,0)$ and (u_2,v_2) are the only minima of ϕ.*
(v) *For $c < c^*_{k,\ell}$ we have $\phi(u,v) > 0$ for $(u,v) \neq (0,0)$.*
(vi) *For $c > c^*_{k,\ell}$ Equation (6.3) has two non-trivial solutions $(u_1,v_1) < (u_2,v_2)$. They satisfy $\phi(u_2,v_2) < \phi(0,0) = 0 < \phi(u_1,v_1)$.*

Proof. **(i)** The partial derivatives of ϕ are $\nabla\phi(u,v) = (h_g^{-1}(u) - v, h_f^{-1}(v) - u)$. Therefore, the only candidates for local minima of ϕ are the solutions to Equation (6.3) (it is easy to check that, except for $(u,v) = (0,0)$, there are no local minima at the borders).
(ii) Assume (u_1,v_1) is the smallest non-trivial solution to Equation (6.3). Considering Figure 6.2, we see that $|\phi(u_1,v_1)|$ is the area enclosed by $h_f(u)$ and $h_g^{-1}(u)$ for $u \in [0,u_1]$. To see that the sign of $\phi(u_1,v_1)$ is positive, observe that for small values of u we have $h_f(u) = Q(cku,\ell) = O(u^\ell)$ while $h_g^{-1}(u) = \Omega(u^{1/(k-1)})$ and thus $h_f(\varepsilon) < h_g^{-1}(\varepsilon)$ for $\varepsilon \in (0,u_1)$. This uses $\ell \geq 1$, $k \geq 2$ and $(k,\ell) \neq (2,1)$.
(iii) By expanding h_f and h_g and substituting $\xi = ckv^{k-1}$ we get for $v \neq 0$:

$$(u,v) = (h_g(v), h_f(u)) \Rightarrow v = Q(ckv^{k-1},\ell) \Leftrightarrow \tfrac{\xi}{ck} = Q(\xi,\ell)^{k-1} \Leftrightarrow \tfrac{\xi}{Q(\xi,\ell)^{k-1}} = ck.$$

To show that the right-most equation has at most two solutions it suffices to show that $\frac{\xi}{Q(\xi,\ell)^{k-1}}$ has at most one local extremum. If ξ is such an extremum, we get

$$\begin{aligned}
&\tfrac{d}{d\xi}\tfrac{\xi}{Q(\xi,\ell)^{k-1}} = 0 \Rightarrow Q(\xi,\ell)^{k-1} - \xi(k-1)Q(\xi,\ell)^{k-2}Q'(\xi,\ell) = 0\\
&\Rightarrow Q(\xi,\ell) - \xi(k-1)Q'(\xi,\ell) = 0 \Rightarrow \sum_{i\geq\ell}\tfrac{\xi^i}{i!} - (k-1)\xi^\ell(\ell-1)! = 0\\
&\Rightarrow \sum_{i\geq 0}\tfrac{\xi^i}{(i+\ell)!} = (k-1)(\ell-1)!
\end{aligned}$$

Since the left hand side is increasing in ξ for $\xi > 0$ while the right hand side is constant, there is exactly one solution ξ as claimed.
(iv) Recall that c occurs in the definition of h_f and note that ϕ is monotonically decreasing in c. It is easy to see that ϕ is nowhere negative for small values of c, and negative for some (u,v) if c is large. For continuity reasons and because $\phi(u,v) \geq 0$ for $u,v \in [0,\varepsilon]$ with $\varepsilon = \varepsilon(c)$ small enough (using similar arguments as in **(ii)**), there must be some intermediate value c where $\phi(u_2,v_2) = 0$ for a local minimum $(u_2,v_2) \neq (0,0)$ of ϕ. By **(i)**, (u_2,v_2) is a solution of Equation (6.3). By **(ii)** there must be a smaller solution (u_1,v_1) with $\phi(u_1,v_1) > 0$. Now by **(i)**, and **(iii)**, there cannot be minima of ϕ in addition to $(0,0)$ and (u_2,v_2). The only thing left to show is $c = c^*_{k,\ell}$.
We rewrite the potential at (u_2,v_2), using Equation (6.3)

$$\begin{aligned}
\phi(u_2,v_2) &= \int_0^{u_2} h_g^{-1}(u)du + \int_0^{v_2} h_f^{-1}(v)dv - u_2v_2\\
&= \left(u_2v_2 - \int_0^{v_2} h_g(v)dv\right) + \left(u_2v_2 - \int_0^{u_2} h_f(u)du\right) - u_2v_2\\
&= v_2h_g(v_2) - H_g(v_2) - H_f(h_g(v_2)),
\end{aligned}$$

where H_g and H_f are antiderivatives of h_g and h_f, i.e:

$$H_g(v) = \int h_g(v)dv = \tfrac{1}{k}v^k \qquad H_f(u) = \int h_f(u)du = u - \tfrac{1}{ck}\sum_{i=1}^{\ell} Q(cku, i).$$

The fact that $\int_0^\lambda Q(x,\ell)dx = \lambda - \sum_{i=1}^{\ell} Q(\lambda, i)$ can be seen by induction on ℓ. We now examine the implications of $\phi(u_2, v_2) = 0$. In the following calculation let $\xi \coloneqq ckv_2^{k-1}$ which implies $Q(\xi,\ell) = v_2$.

$$\begin{aligned}
0 = \phi(u_2,v_2) &= v_2 h_g(v_2) - H_g(v_2) - H_f(h_g(v_2)) \\
&= v_2^k - v_2^k/k - v_2^{k-1} + \tfrac{1}{ck}\sum_{i=1}^{\ell} Q(ckv_2^{k-1}, i) \\
\Rightarrow 0 = \xi v_2 - \xi v_2/k - \xi + \sum_{i=1}^{\ell} Q(\xi,i) &= \xi Q(\xi,\ell) - \xi Q(\xi,\ell)/k - \xi + \sum_{i=1}^{\ell} Q(\xi,i) \\
\Rightarrow \quad \xi Q(\xi,\ell)/k = \xi(Q(\xi,\ell)-1) + \sum_{i=1}^{\ell} Q(\xi,i) &= -\mathrm{e}^{-\xi}\sum_{j=0}^{\ell-1}\frac{\xi^{j+1}}{j!} + \sum_{i=1}^{\ell}\left(1 - \mathrm{e}^{-\xi}\sum_{j=0}^{i-1}\frac{\xi^j}{j!}\right) \\
&= \ell - e^{-\xi}\sum_{j=0}^{\ell-1}\left(\frac{\xi^{j+1}}{j!} + (\ell-j)\frac{\xi^j}{j!}\right) = \ell - e^{-\xi}\left(\frac{\xi^\ell}{(\ell-1)!} + \sum_{j=0}^{\ell-1}\ell\frac{\xi^j}{j!}\right) \\
&= \ell - \ell e^{-\xi}\sum_{j=0}^{\ell}\frac{\xi^j}{j!} = \ell Q(\xi,\ell+1) \qquad \Rightarrow \qquad k\ell = \frac{\xi Q(\xi,\ell)}{Q(\xi,\ell+1)}.
\end{aligned}$$

The last equation characterises the threshold $c^*_{k,\ell}$ for ℓ-orientability of random k-uniform hypergraphs, see for instance [FKP16]. Thus $c = c^*_{k,\ell}$ follows.

(v) We now make the dependence of $\phi_c(u,v)$ on c explicit. For monotonicity reasons we have $\phi_c(u,v) > \phi_{c'}(u,v)$ whenever $c < c'$ and $v \neq 0$. Since $\phi_{c^*_{k,\ell}}$ is positive except for its two roots at $(0,0)$ and (u_2,v_2), for $c < c^*_{k,\ell}$ the potential ϕ_c is positive except at $(0,0)$.

(vi) Since $\phi_{c^*_{k,\ell}}$ has a non-trivial root, ϕ_c attains negative values for monotonicity reasons. By **(i)**, the potential attains its (negative) minimum at a non-trivial solution to Equation (6.3), and by **(ii)** it attains a positive value at the smallest non-trivial solution. Thus, the claim follows.

◀

We are now ready to prove Proposition 6.6 by recruiting help from [KRU15].

Proof of Proposition 6.6. First note that we have $q_0 \geq \mathbf{P}q_0$ by definition, which implies $\mathbf{P}^r q_0 \geq \mathbf{P}^{r+1}q_0$ by monotonicity of $\mathbf{P}$ and induction on r. Thus, $\mathbf{P}^r q_0$ is pointwise bounded and decreasing and must converge to a limit q^*. As $\mathbf{P}$ is continuous (see Lemma 6.5) we have $\mathbf{P}q^* = q^*$.

(i) Let $\mathbb{1}\colon \mathbb{R} \to \{1\}$ be the 1-function. First note that for any $x \in X$ we have, using properties from Lemma 6.5 and monotonicity of h_f and h_g

$$(\mathbf{P}^r q_0)(x) \leq (\hat{\mathbf{P}}^r \mathbb{1})(x) = (h_f \circ h_g)^r(1) \xrightarrow{r\to\infty} \max\{u \in [0,1] \mid h_f(h_g(u)) = u\}. \quad (6.4)$$

So if the only solution of $h_f(h_g(u)) = u$ is $u = 0$, then we get $\mathbf{P}^r q_0(x) \overset{r\to\infty}{\longrightarrow} 0$ from this alone. Otherwise, by Lemma 6.7 **(iii)**, there are one or two non-trivial solutions, the larger of which we denote by (u_2, v_2).
We now apply [KRU15, Thm. 10][7]. It requires $\phi(u,v) > 0$ for $0 \neq (u,v) \in [0,u_2] \times [0,v_2]$, which we have shown in Lemma 6.7 **(v)**. The theorem asserts pointwise convergence of $f^{(r)}$ to zero for any $f^{(0)} : \mathbb{R} \to [0,u_2]$ in the case of one-sided termination. Clearly this implies convergence to zero in the case of two-sided termination as well, i.e. $\mathbf{P}^r f^{(0)} \overset{r\to\infty}{\longrightarrow} 0$. Choosing $f^{(0)} = \mathbb{1} \cdot u_2$ we get

$$\lim_{r\to\infty}(\mathbf{P}^r q_0) = \lim_{r\to\infty} \mathbf{P}^r \lim_{s\to\infty} \mathbf{P}^s q_0 \overset{(6.4)}{\leq} \lim_{r\to\infty} \mathbf{P}^r f^{(0)} = 0.$$

(ii) Using Lemma 6.7 **(vi)** and **(iii)**, we know there are exactly three solutions $(0,0) < (u_1,v_1) < (u_2,v_2)$ to Equation (6.3) and the signs of their potentials are zero, positive and negative, respectively. This is sufficient to apply [KRU15, Thm. 14][8]. The theorem asserts the existence of a solution $q^* : X \to [0,u_2]$ of $\mathbf{P}q^* = q^*$ with $q^*(\frac{z+1}{2}) = u_2 - \varepsilon$ for any $\varepsilon > 0$, assuming $z = z(\varepsilon)$ is large enough.
By monotonicity of $\mathbf{P}$ we have $\lim_{r\to\infty} \mathbf{P}^r q_0 \geq \lim_{r\to\infty} \mathbf{P}^r q^* = q^*$. ◀

6.4. Peelability of F_n below $c^*_{k,\ell}$

We now connect the behaviour of system (6.1) to the survival probabilities $q^{(R)}(x)$ we were originally interested in. For $c < c^*_{k,\ell}$ and any $z \in \mathbb{N}$, they can be made smaller than any $\delta > 0$ in $R = R(\delta, k, \ell, z, c)$ rounds.

► **Lemma 6.8.** *If $c < c^*_{k,\ell}$ then $\forall z \in \mathbb{R}^+, \delta > 0 : \ \exists R, N \in \mathbb{N} : \ \forall n \geq N, x \in X : q^{(R)}(x) < \delta$.*

Proof. Let $z \in \mathbb{R}^+$ and $\delta > 0$ be arbitrary constants. At first, Proposition 6.6 **(i)** implies only pointwise convergence $\mathbf{P}^r q^{(0)}(x) \overset{r\to\infty}{\longrightarrow} 0$ for all $x \in X$. However, since X is compact, since $\mathbf{P}^r q^{(0)}$ is continuous for $r > 0$ and since the all-zero limit is obviously continuous, basic calculus[9] implies uniform convergence, i.e. there is a constant R such that $\mathbf{P}^R q^{(0)}(x) \leq \delta/2$ for all $x \in X$. Therefore for $x \in X$:

$$q^{(R)}(x) \overset{\text{Cor 6.4}}{=} (\mathbf{P}^R q^{(0)})(x) + o(1) \leq \delta/2 + o(1) \leq \delta.$$

In the last step we simply choose $N \in \mathbb{N}$ large enough. ◀

Lemma 6.2 only allows us to track $q^{(R)}$ via $\mathbf{P}^R q_0$ for a *constant* number of rounds R. Therefore, we need to accompany Lemma 6.8 with the following combinatorial argument that shows that if all but a δ-fraction of the vertices are peeled, then with high probability (whp) the rest is peeled as well. Arguments such as these are standard, many similar ones can be found for instance in [FKP16; FP12; JL07; Lel12; Luc91; MPW18; Mol05].

---

[7] Strictly speaking, the theorem requires functions h_f and h_g with $h_f(0) = h_g(0) = 0$ and $h_f(1) = h_g(1) = 1$. As the authors of [KRU15] point out themselves, this is purely to simplify notation. We can apply the theorem to our $h_f : [0,u_2] \to [0,v_2]$ and $h_g : [0,v_2] \to [0,u_2]$ with $h_f(0) = h_g(0) = 0$ and $h_f(u_2) = v_2, h_g(v_2) = u_2$ after rescaling the axes so (u_2,v_2) becomes $(1,1)$. We will not do so explicitly.

[8] See previous footnote.

[9] Sometimes referred to as Dini's Theorem after Ulisse Dini (1848 – 1918).

▶ **Lemma 6.9.** *Let $c \in [0, \ell]$. There exists $\delta = \delta(k, \ell, z) > 0$ such that, whp, any subhypergraph of $F_n = F(n, k, c, z)$ induced by at most δn vertices has minimum degree at most ℓ.*

Proof. In the course of the proof, we will implicitly encounter positive upper bounds on δ in terms of k, ℓ and z. Any $\delta > 0$ small enough to respect these bounds is suitable. We consider the bad events $W_{s,t}$ that some small set $V' \subseteq [n]_0$ of size s induces t hyperedges for $1 \le s \le \delta n$, $\frac{(\ell+1)s}{k} \le t \le |E|$. If *none* of these events occur, then all such V' induce less than $(\ell+1)|V'|/k$ hyperedges and therefore induce hypergraphs with average degree less than $\ell+1$, so a vertex of degree at most ℓ exists in each of them.

We will show $\Pr[\bigcup_s \bigcup_t W_{s,t}] = O(1/n)$ using a first moment argument. It is convenient to assume that hyperedges are k-tuples, possibly with repetition. First note that F_n then contains three copies of the same hyperedge with probability at most $\binom{m}{3}(\frac{z+1}{n})^{-2k} = O(n^{-2k+3}) = O(n^{-1})$, so we restrict our attention to F_n with at most two copies of the same hyperedge. Given s and t there are $\binom{n}{s}$ ways to choose V'. Since there are s^k ways to form k-tuples from vertices of V' and each hyperedge occurs at most twice, there are at most $\binom{2s^k}{t}$ multisets of hyperedges that V' could induce. The probability that any given k-tuple occurs as a hyperedge is either zero if the k vertices are too far apart or at most $1 - (1 - (\frac{(z+1)}{n})^k)^{czn} \le \frac{(z+1)^k \ell}{n^{k-1}}$. Similarly, it occurs as a duplicate hyperedge with probability at most $(\frac{(z+1)^k \ell}{n^{k-1}})^2$. Since the presence of hyperedges is negatively correlated, we may obtain an upper bound on the probability of the event that a set of hyperedges are all simultaneously present by taking the product of the events for the presence of the individual hyperedges. Thus, using constants $C, C', C'' \in \mathbb{R}^+$ (that may depend on k, ℓ and z) where precise values do not matter, we get

$$\begin{aligned}
\Pr[\bigcup_{s=1}^{\delta n} \bigcup_{t=(\ell+1)s/k}^{|E|} W_{s,t}] &\le \sum_{s=1}^{\delta n} \sum_{t=(\ell+1)s/k}^{|E|} \Pr[W_{s,t}] \le \sum_{s=1}^{\delta n} \sum_{t=(\ell+1)s/k}^{|E|} \binom{n}{s}\binom{2s^k}{t}\left(\frac{(z+1)^k \ell}{n^{k-1}}\right)^t \\
&\le \sum_{s=1}^{\delta n} \sum_{t=(\ell+1)s/k}^{|E|} \left(\frac{en}{s}\right)^s \left(\frac{2e(z+1)^k \ell s^k}{t n^{k-1}}\right)^t \le \sum_{s=1}^{\delta n} \sum_{t=(\ell+1)s/k}^{|E|} \left(C\frac{n}{s}\right)^s \left(C'\frac{s^{k-1}}{n^{k-1}}\right)^t \\
&\le 2\sum_{s=1}^{\delta n} \left(C\frac{n}{s}\right)^s \left(C'\frac{s^{k-1}}{n^{k-1}}\right)^{\lceil(\ell+1)s/k\rceil} = 2\sum_{s=1}^{\delta n} \left(C''\tfrac{s}{n}\right)^{\lceil((k-1)(\ell+1)-k)\frac{s}{k}\rceil} \le 2\sum_{s=1}^{\delta n} \left(C''\tfrac{s}{n}\right)^{\lceil\frac{s}{k}\rceil}.
\end{aligned}$$

To get rid of the summation over t, we assumed $(s/n)^{k-1} \le \delta^{k-1} \le \frac{1}{2C'}$, in the last step we used $k \ge 2$, $\ell \ge 1$ and $(k, \ell) \ne (2, 1)$. Elementary arguments show that in the resulting bound, the contribution of summands for $s \in \{1, \dots, 2k\}$ is of order $O(\frac{1}{n})$, the contribution of the summands with $s \in \{2k+1, \dots, O(\log n)\}$ is of order $O(\frac{\log n}{n^2})$ (using $\frac{s}{n} \le \frac{\log n}{n}$) and the contribution of the remaining terms with $s \ge 3\log_2 n$ is of order $O(2^{-\log_2 n}) = O(\frac{1}{n})$ (using $C''\frac{s}{n} \le C''\delta \le \frac{1}{2}$). This gives $\Pr[\bigcup_{s,t} W_{s,t}] = O(n^{-1})$, proving the claim. ◀

We are now ready to prove the first half of Theorem A.

Proof of Theorem A (i). Let $c < c^*_{k,\ell}$ and $z \in \mathbb{R}^+$. We need to show that F_n is ℓ-peelable whp.

First, let $\delta = \delta(k, \ell, z)$ be the constant from Lemma 6.9 and $R = R(\delta/2)$ as well as N the corresponding constants from Lemma 6.8.

Assuming $n \geq N$ we have $q^{(R)}(x) \leq \delta/2$ for all $x \in X$, meaning any vertex v from F_n is *not* deleted within R rounds of $\mathrm{peel}_{v,R}(F_n)$ with probability at most $\delta/2$. Since $\mathrm{peel}(F_n)$ deletes in R rounds at least the vertices that any $\mathrm{peel}_{v,R}(F_n)$ for $v \in V$ deletes in R rounds, the expected number of vertices not deleted by $\mathrm{peel}(F_n)$ within R rounds is at most $\delta n/2$.

Now standard arguments using Azuma's inequality (see, e.g. [MU17, Thm. 13.7] but also Theorem 5.2 in this thesis) suffice to conclude that whp at most δn vertices are not deleted by $\mathrm{peel}(F_n)$ within R rounds.

By Lemma 6.9, whp, neither the remaining δn vertices, nor any subset induces a hypergraph of minimum degree $\ell + 1$. Therefore $\mathrm{peel}(F_n)$ deletes all vertices whp. ◀

6.5. Non-Orientability of F_n above $c^*_{k,\ell}$

To show that F_n is not ℓ-*peelable* whp for $c > c^*_{k,\ell}$, we argue that F_n is even not ℓ-*orientable* whp.[10] Our proof relies on the notion of local weak convergence and Lelarge's Theorem, as discussed in Chapter 5. There are three ingredients.

Ingredient 1: Identical weak limits. Let $H^k_{n,cn}$ be the fully random hypergraph (see Equation (2.1)) and let G^H_n be the incidence graph of $H^k_{n,cn}$. By Theorem 5.2 we have $(G^H_n)_{n\in\mathbb{N}} \xrightarrow{\text{LWL}} T$ almost surely for $T := \mathrm{GWT}_{\delta_k,\mathrm{Po}(ck)}$.

Now let $F_n = F(n,k,c,z)$ be the random hypergraph from Definition 2.4. We define $\widetilde{F}_n$ to be a "borderless" version of F_n where the vertices i and $i + \frac{nz}{z+1}$ for all $i \in [\frac{n}{z+1}]_0$ are merged, "glueing" the right-most $\frac{n}{z+1}$ vertices of F_n on top of the left-most $\frac{n}{z+1}$ vertices of F_n. Moreover, let $\widetilde{G}^F_n$ be the incidence graph of $\widetilde{F}_n$.

It is unsurprising and not hard to prove that $(\widetilde{G}^F_n)_{n\in\mathbb{N}}$ and $(G^H_n)_{n\in\mathbb{N}}$ have the same local weak limit, i.e. that $(\widetilde{G}^F_n)_{n\in\mathbb{N}} \xrightarrow{\text{LWL}} T$ holds almost surely as well. Indeed, the proof of Theorem 5.2 works with very few changes.

Ingredient 2: Lelarge's Theorem [Lel12]. The following is an immediate consequence of Lelarge's Theorem stated as Theorem 5.1 in Section 5.2. For a bipartite graph $G = (A, B, E)$ let the number $M(G)$ denote the maximum size of a set $E' \subseteq E$ with $\deg_{E'}(a) \leq 1$ for $a \in A$ and $\deg_{E'}(b) \leq \ell$ for $b \in B$.

▶ **Corollary 6.10** (To Lelarge's Theorem). *Let $(G^{(i)}_n = (A^{(i)}_n, B^{(i)}_n, E^{(i)}_n))_{n\in\mathbb{N}}$ for $i \in \{1,2\}$ be two sequences of bipartite graphs with $|E^{(i)}_n| = O(|A^{(i)}_n|)$. Let T is a bipartite Galton-Watson tree.*

If $(G^{(i)}_n)_{n\in\mathbb{N}} \xrightarrow{LWL} T$ holds almost surely for $i \in \{1,2\}$, then $\lim_{n\to\infty} \frac{M(G^{(1)}_n)}{|A_n|} = \lim_{n\to\infty} \frac{M(G^{(2)}_n)}{|A_n|}$ holds almost surely.

[10] Alternatively, one could try to base a proof on Proposition 6.6 **(ii)**, possibly by going through similar motions as [Mol05, Lemma 4]. If successful, this might give a detailed characterisation of the $(\ell+1)$-*core* of F_n – the largest subhypergraph of F_n with minimum degree $\ell + 1$. Presumably, the $(\ell+1)$-core contains roughly a $q^*(x)$-fraction of the vertices with position roughly at $x \in X$. We leave this aside. Our approach has the upside of establishing a connection between orientability thresholds and peelability thresholds.

Recall that if $H = (V, E)$ is a hypergraph and $G = (E, V, \text{“}\ni\text{”})$ its incidence graph, then $M(G)$ is the size of a largest set $E' \subseteq E$ such that the subhypergraph (V, E') of H is ℓ-orientable. In other words, $M(G)$ is the size of the largest *partial ℓ-orientation* of H.

Ingredient 3: Orientability-Gap above the threshold. Assume $c = c^*_{k,\ell} + \varepsilon$ for $\varepsilon > 0$. By definition, it is not the case that H_n is ℓ-orientable whp. More strongly however, it is known [FKP16; Lel12] that there exists a constant $\delta = \delta(\varepsilon) > 0$ such that the largest partial ℓ-orientation of H_n has size $(1-\delta)cn + o(n)$ whp. In the terms of Corollary 6.10, this means $\lim_{n\to\infty} M(G_n^H)/|cn| = 1 - \delta$ almost surely. We now put all three ingredients together.

Proof of Theorem A (ii). Let $c = c^*_{k,\ell} + \varepsilon$ and $\delta = \delta(\varepsilon)$ as above. We pick $z \geq z^* := \frac{2\ell}{\delta c}$.

Since $(G_n^H)_{n\in\mathbb{N}}$ and $(\widetilde{G}_n^F)_{n\in\mathbb{N}}$ almost surely share the random weak limit T, we conclude from Theorem 5.1 that the orientability gap carries over from H_n to $\widetilde{F}_n$, i.e. $\lim_{n\to\infty} M(\widetilde{G}_n^F)/m = 1 - \delta$ almost surely, where $m = \frac{czn}{z+1}$ is the number of hyperedges in F_n and $\widetilde{F}_n$.

In particular, the size of the largest partial ℓ-orientation of $\widetilde{F}_n$ is $(1-\delta)m + o(n)$ whp. Switching from $\widetilde{F}_n$ back to F_n splits $\frac{n}{z+1}$ vertices and can increase the size of a largest partial ℓ-orientation by at most $\frac{\ell n}{z+1}$ to $(1 - \delta + \frac{\ell}{cz})m + o(n) \leq (1 - \frac{\delta}{2})m + o(n)$ whp. Thus F_n is not ℓ-orientable whp. ◀

Relevant Definition and Theorems for Chapter 7

(Originally stated on Pages 18 and 39)

► **Definition 2.5** (Random Matrix with Two Aligned Blocks per Row). *Let $n, m, L \in \mathbb{N}$ with $m \leq n$ and L a divisor of n. For a* block index *$b \in [n/L]$ and a* pattern *$p \in \{0,1\}^L$ we let $B_{b,p} = 0^{bL-L} \circ p \circ 0^{n-bL} \in \{0,1\}^n$ where "$\circ$" denotes concatenation of bit strings. To obtain the random matrix $A^L_{m,n} \in \{0,1\}^{m \times n}$, each row is independently sampled as $B_{b_1,p_1} \oplus B_{b_2,p_2}$ where $b_1, b_2 \leftarrow [n/L]$ and $p_1, p_2 \leftarrow \{0,1\}^L$.*
We define $A^{L*}_{m,n}$ the same way, except for $p_1, p_2 \leftarrow \{0,1\}^L \setminus \{0^L\}$ the all-zero pattern is forbidden.

$$\begin{array}{cccc} \scriptstyle 1 & \scriptstyle 2 & \scriptstyle 3 & \scriptstyle 4 \\ \begin{pmatrix} 010 & & 110 & \\ 100 & & & 110 \\ 111 & & & 010 \\ 010 & 110 & & \\ & 111 & & 001 \\ & & 001 & \end{pmatrix} & & & \end{array}$$

Possible outcome for $A^3_{6,12}$.

► **Theorem B.** *Let $\beta = 39$, $\gamma = 1/4$ and $\delta > 0$ constants. Then we have:*

(i) *If $L = L(n) \geq \beta \log_2(n)$ then $A^L_{n,n}$ is regular with probability $\Theta(1)$.*

(ii) *If $2L = 2L(n) \geq (1+\delta)\log_2 n$ and $1 - c \geq \max\{2^{-\gamma L}, \beta \log_2(n)/n\}$ then $A^L_{cn,n}$ has independent rows with probability at least $1 - \widetilde{O}(n^{-\min(1,\delta)})$.*

(iii) *If $1 - c \geq \max\{2^{-\gamma L}, \beta \log_2(n)/n\}$ then $A^{L*}_{cn,n}$ has independent rows with probability at least $1 - O(\max(n^{-1}, L2^{-L}))$.*

► **Theorem B1.** *We aim to build a retrieval data structure for $f : S \to \{0,1\}^r$ where $|S| = m$.*

(i) *When using $A = A^L_{m,m}$ as in Theorem B **(i)** for suitable $L = O(\log m)$ in the retrieval framework from Section 3.4, then the following can be achieved:*

- *One trial of* **construct** *takes time $O(\frac{m^3}{w \log m})$ and succeeds with probability $\Theta(1)$. The number of trials s until the first trial succeeds has expectation $\mathbb{E}[s] = O(1)$.*
- *The data structure consists of $Z \in \{0,1\}^{m \times r}$, m and s, occupying rm bits, $O(\log m)$ bits and $O(\log s)$ bits, respectively.*
- *An eval-operation (using $\mathbf{eval}_2$) takes time $O(r)$ and causes two cache misses when Z is stored appropriately.*

(ii) *Using Wiedemann's algorithm (see Section 4.4.1), the expected construction time in **(i)** can be reduced to $\widetilde{O}(rm^2)$.*

(iii) *Using $A^L_{m,n}$ as in Theorem B **(ii)** for suitable n and L instead, increases the size of D_f by $O(r \log m)$ bits. The implementation is more convenient and each trial of* **construct** *succeeds whp.*

7. Constant - Time Retrieval with $O(\log m)$ Extra Bits

The purpose of this chapter is to prove Theorems B and B1, relating to the matrix $A^L_{m,n}$ from Definition 2.5 (see facing page).

We start with the proof of Theorem B, establishing lower bounds on the probability that $A^L_{cn,n}$ has full rank, which is the main ingredient for the proof of Theorem B1, provided afterwards. Note that variation **(iii)** of Theorem B concerning $A^{L*}_{cn,n}$ is not required for Theorem B1, but may be of independent interest.

Our practical implementation in Section 12.2 is in the spirit of Theorem B2, which combines Theorem B1 **(iii)** with input partitioning.

7.1. Proof of Theorem B

Before we start, we make a few high-level remarks about the theorem statement.

- For $2L = \log n$ the matrix $A^L_{cn,n}$ contains an all-zero row with constant probability, which explains why $\delta > 0$ is required in **(ii)**. One may interpret **(iii)** as a continuation of **(ii)** to $2L < \log n$, achieved by switching to $A^{L*}_{cn,n}$ where all-zero rows are far less likely.
- An all-zero row in $A^{L*}_{cn,n}$ requires an element x with hash values $b_1(x) = b_2(x)$ and $p_1(x) = p_2(x)$. The probability for this is $\Theta(L \cdot 2^{-L})$, explaining the term in the failure probability in **(iii)**.
- We have not tried to optimise the constants γ and β. We remark that in **(i)** a rough calculation concerning the event that each block occurs in at least L rows suggests a lower bound of $\beta \geq 1/(\log(e) - 1) \geq 2.25$. In particular, the requirement on L cannot be weakened to the one in **(ii)**.
- It is reasonable to suspect that for each $L \geq 2$ there is a threshold value $c^*_L \in (0, 1)$ such that for $c < c^*_L$ the matrix $A^{L*}_{cn,n}$ has independent rows with probability at least $1/2$ and for $c > c^*_L$ it has dependent rows whp.
  Assuming the values c^*_L are well-defined, approximations (obtained for $n = 10^4$) are given in table Table 7.1. To obtain each c^*_L, we generated 100 copies of $A^{L*}_{c\cdot 10^4, 10^4}$ for each c using the pseudo random number generator `std::mt19937` with distinct seeds, and counted the number $0 \leq s^L_c \leq 100$ of systems with independent rows. We reported $c^*_L \coloneqq \operatorname{argmin}_{c^*} \#\{c \in \{0, \frac{1}{n}, \ldots, 1\} \mid (c < c^* \wedge s^L_c < 50) \vee (c > c^* \wedge s^L_c > 50\}$.

7.1.1. General Considerations

Let $n \in \mathbb{N}$ be a multiple of L, and $m = cn$ for some $0 < c \leq 1$. Recall Definition 2.5 for $A = A^L_{m,n}$. We use $b_1(i), b_2(i) \in [n/L]$ and $p_1(i), p_2(i) \in \{0, 1\}^L$ to denote the random block indices and patterns selected for a row $i \in [m]$. If A does not have independent rows, then

this is *witnessed* by a non-trivial subset $W \subseteq [m]$ of elements such that the corresponding rows of A sum to zero. We use a first moment calculation to bound the probability for the existence of an *inclusion-minimal* witness W. We fix two parameters of candidate sets W: The number $s = |W|$ of rows, with $1 \le s \le m = cn$, and the number $t = |B| \in [n/L]$, where $B = \bigcup_{w \in W} \{b_1(w), b_2(w)\}$ is the set of variable blocks involved in at least one of the rows.

There are $\binom{m}{s}$ ways to choose W, and $\binom{n/L}{t}$ ways to choose B. The probability that the rows corresponding to W involve exactly the blocks from B is

$$\Pr[B = \bigcup_{w \in W} \{b_1(w), b_2(w)\}] \le \prod_{w \in W} \Pr[b_1(w) \in B \wedge b_2(w) \in B] = (\tfrac{t}{n/L})^{2s}.$$

The event that the rows corresponding to W sum to zero is the intersection of the independent events that the rows sum to zero within each block $b \in B$. Its probability is therefore

$$\prod_{b \in B} \Pr\Big[\bigoplus_{\substack{w \in W, i \in \{0,1\} \\ b_i(w) = b}} p_i(w) = 0^L \;\Big|\; \exists w \in W, i \in \{0,1\} \colon b_i(w) = b \Big] = \prod_{b \in B} 2^{-L} = 2^{-Lt}.$$

In the following, it is often convenient to deal with the fraction $\sigma = s/m$ of rows and the fraction $\tau = t/(n/L) = Lt/n$ of blocks involved in a witness. Accordingly, we define $O(n^2/L)$ values $p_{\sigma,\tau}$, where $p_{\sigma,\tau}$ is the probability that some set of equations involving σm rows and $\tau n/\ell$ blocks is a minimal witness. This gives

$$p_{\sigma,\tau} \le \binom{m}{s}\binom{n/L}{t}(\tfrac{t}{n/L})^{2s}2^{-Lt} = \binom{m}{\sigma m}\binom{n/L}{\tau n/L}\tau^{2\sigma m}2^{-\tau n}. \tag{7.1}$$

We now list a few bounds that will be useful later. Throughout, $\sigma \in \{\frac{1}{m}, \ldots, \frac{m}{m} = 1\}$, $\tau \in \{\frac{1}{n/\ell}, \ldots, \frac{n/\ell}{n/\ell} = 1\}$, logarithms have base 2 and $\bar{x}$ is a shorthand for $1 - x$.

► **Lemma 7.1.** *Let $H(x) = -x \log x - \bar{x} \log \bar{x}$ be the binary entropy function. Then*

(a) $\frac{L}{n} \log(p_{\sigma,\tau}) \le cLH(\sigma) + H(\tau) + 2\sigma cL \log \tau - L\tau = cL(H(\sigma) + \sigma \log \tau^2) + H(\tau) - L\tau.$

(b) $\frac{L}{n} \log(p_{\sigma,\tau}) \le L(c \log(1 + \tau^2) - \tau) + H(\tau).$

(c) *All minimal witnesses satisfy $t \le s + 1$.*

(d) $$\log(1 + \tau^2) \le \begin{cases} \tau \cdot 2 \log \frac{5}{4} \le \frac{2}{3}\tau & \text{if } 0 \le \tau \le \frac{1}{2}, \\ 1 - 2\bar{\tau} \cdot (1 - \log \frac{5}{4}) \le 1 - \frac{4}{3}\bar{\tau} & \text{if } \frac{1}{2} \le \tau \le 1, \\ \tau & \text{if } 0 \le \tau \le 1. \end{cases}$$

(e) $-\log \tau \le 2\bar{\tau}$ *if* $\frac{1}{2} \le \tau \le 1$.

L	1	2	3	4	5	6	7	8
c_L^*	0.4712	0.6943	0.8531	0.9223	0.9564	0.9752	0.9834	0.9902
L	9	10	11	12	13	14	15	16
c_L^*	0.9942	0.9963	0.9976	0.9985	0.9991	0.9994	0.9996	0.9998

Table 7.1 Approximations of the conjectured thresholds c_L^* for full rank of $A_{cn,n}^{L*}$.

(f) $-\tau_1 \log \tau_1 \le -\tau_2 \log \tau_2$ *for* $0 < \tau_1 < \tau_2 < \frac{1}{4}$.
(g) $H(\tau) \le -\tau \log \tau + 2\tau$ *if* $0 < \tau \le \frac{1}{2}$.
(h) $H(\tau_1) < H(\tau_2)$ *for* $0 < \tau_1 < \tau_2 \le \frac{1}{2}$ *and* $H(\tau) = H(\bar{\tau})$ *for* $0 < \tau \le 1$.
(j) *Without loss of generality, we may assume* $L \ge L_0$ *for any absolute constant* $L_0 \in \mathbb{N}$.
(k) *If* $s \ge t$ *and* $0 < \tau < 1/L$ *then* $\frac{L}{n} \log(p_{\sigma,\tau}) \le -\tau L/2$.

The claims of Lemma 7.1 can be verified with simple calculations, given in Section 7.1.5.

We now consider various ranges of τ and derive bounds for the sum of all $p_{\sigma,\tau}$ that fall within these ranges. The proofs in Sections 7.1.2 to 7.1.4 will profit from these preparations. We refer to parts of Lemma 7.1 by their labels.

Case 1: $\frac{1}{2} \le c \le \tau \le 1$ assuming $\bar{c} \ge \max\{2^{-L/4}, 15\log(n)/n\}$.

$$\begin{aligned}\tfrac{L}{n}\log(p_{\sigma,\tau}) &\overset{\text{(b,d)}}{\le} L(c\tau - \tau) + H(\tau) \overset{\text{(h)}}{\le} -L\tau\bar{c} + H(\bar{c}) \overset{\text{(g)}}{\le} -L\bar{c}/2 - \bar{c}\log\bar{c} + 2\bar{c} \\ &\le -L\bar{c}/2 + \bar{c}L/4 + 2\bar{c} = -\bar{c}(L/2 - L/4 - 2) \overset{\text{(j)}}{\le} -\bar{c}L/5.\end{aligned}$$

This gives a bound of:

$$p_{\sigma,\tau} \le 2^{-\bar{c}n/5} \le n^{-3}.$$

Multiplying with $O(n^2)$ choices for σ and τ, this is still $O(n^{-1})$.

Case 2: $1/2 \le \tau, \bar{\tau} \ge \max\{2^{-L/4}, 39\log(n)/n\}$.

$$\begin{aligned}\tfrac{L}{n}\log(p_{\sigma,\tau}) &\overset{\text{(b,d)}}{\le} L(c(1 - \tfrac{4}{3}\bar{\tau}) - \tau) + H(\tau) \overset{\text{(g,h)}}{\le} L(\bar{\tau} - \tfrac{4}{3}\bar{\tau}) - \bar{\tau}\log\bar{\tau} + 2\bar{\tau} \\ &\le \bar{\tau}(-L/3 - \log\bar{\tau} + 2) \le \bar{\tau}(-L/3 + L/4 + 2) \overset{\text{(j)}}{\le} -\bar{\tau}L/13.\end{aligned}$$

From this we obtain a bound $p_{\sigma,\tau} = n^{-39/13}$ for an aggregated $O(n^{-1})$ as in Case 1.

Case 3: $\tau \le 1/2, \tau \ge \max\{2^{-L/4}, 39\log(n)/n\}$.

$$\tfrac{L}{n}\log(p_{\sigma,\tau}) \overset{\text{(b,d)}}{\le} L(\tfrac{2}{3}\tau - \tau) + H(\tau) \overset{\text{(g)}}{\le} -\tau L/3 - \tau\log\tau + 2\tau = \tau(-L/3 - \log\tau + 2)$$

We may proceed as in Case 2 with τ instead of $\bar{\tau}$.

Case 4: $8\log(n)/n < \tau < 1/L$. Assuming $s \ge t$ for the moment, we may apply **(k)** to obtain $\frac{L}{n}\log(p_{\sigma,\tau}) \le -\tau L/2$. This gives $p_{\sigma,\tau} = 2^{-\tau n/2} \le 2^{-4\log n} \le n^{-4}$. Summing over all admissible σ and τ gives a bound of $O(n^{-2})$

Inconveniently, **(c)** only gives $s \ge t-1$ instead of $s \ge t$ and the $O(n)$ cases with $s = t-1$ are not yet handled. Luckily, decreasing s by 1 (or equivalently σ by $1/m$) raises the upper bound in Equation (7.1) by at most $O(n^2)$ and the combined contribution of the cases in question is bounded by $O(n) \cdot O(n^2) \cdot n^{-4} = O(n^{-1})$.

Case 5: $2 \le t$ and $L^2t^2 \le \frac{n}{2e}$. We refine Equation (7.1) to get (recall $\sigma = \frac{s}{m}$, $\tau = \frac{Lt}{n}$)

$$\begin{aligned}p_{\sigma,\tau} &\le \binom{m}{s}\binom{n/L}{t}\left(\tfrac{t}{n/L}\right)^{2s} 2^{-Lt} \le \left(\frac{me}{s}\right)^s \left(\frac{en/L}{t}\right)^t \left(\frac{t}{n/L}\right)^{2s} 2^{-Lt} \\ &= \left(\frac{ceL^2t^2}{sn}\right)^s \left(\frac{en}{tL2^L}\right)^t = \left(\frac{ceL^2t^2}{sn}\right)^{1+(s-t+1)+(t-2)} \left(\frac{en}{tL2^L}\right)^{2+(t-2)} \\ &= \left(\frac{ceL^2t^2}{sn}\right)\left(\frac{en}{tL2^L}\right)^2 \left(\frac{ce^2Lt}{s2^L}\right)^{t-2} \left(\frac{ceL^2t^2}{sn}\right)^{s-t+1} \\ &\le \frac{ce^3n}{2^{2L}}(\tfrac{1}{2})^{t-2}(\tfrac{1}{2})^{s-t+1} = \frac{ce^3n}{2^{2L}}(\tfrac{1}{2})^{s-1}\end{aligned}$$

where the last inequality used $L/2^L \le ce^2/4$ (due to (j)), $t/s \le 2$ and the upper bound on L^2t^2. It is crucial that the exponents $t-2$ and $s-t+1$ are nonnegative; for the latter exponent this is because of (c). The sum over all applicable s and t is dominated by the contribution for $t=2$ and $s=1$, i.e.

$$\sum_{s\ge1}\sum_{2\le t\le s+1} p_{\sigma,\tau} \le \sum_{s\ge1}\sum_{2\le t\le s+1} \frac{ce^3n}{2^{2L}}(\tfrac12)^{s-1} = \frac{ce^3n}{2^{2L}}\sum_{s\ge1} s\cdot(\tfrac12)^{s-1} = \frac{4ce^3n}{2^{2L}} = O(n2^{-2L}).$$

Case 6: $1 = t$ and $L^2 = o(n)$. The argument from Case 5 essentially works, but the trivial bound $s \ge t-1 = 0$ needs to be replaced with $s \ge 1$. We get

$$\begin{aligned}\sum_{s\ge1} p_{\sigma,\tau} &\le \sum_{s\ge1}\binom{m}{s}\tfrac{n}{L}\left(\tfrac{L}{n}\right)^{2s}2^{-L} \le \sum_{s\ge1}\tfrac{n}{L\cdot2^L}\left(\frac{meL^2}{sn^2}\right)^s\\ &= \tfrac{ceL}{2^L}\sum_{s\ge1}\left(\frac{ceL^2}{sn}\right)^{s-1} \le O(L2^{-L})\sum_{s\ge1}(\tfrac12)^{s-1} = O(L2^{-L}).\end{aligned}$$

7.1.2. Proof of Theorem B (i)

For the cases with $c=1$ and $\tau=1$, we need the following lemma.

▶ **Lemma 7.2.** *In the situation of Section 7.1.1 with $c=1$, let w be the number of witness sets W with parameter $\tau=1$ (and arbitrary σ). There is an absolute constant $\varepsilon>0$ such that $\Pr[w>0] \le 1-\varepsilon$.*

Proof. Simplifying Equation (7.1) using $n=m$ and $t=n/L$ we get $p_{\sigma,1} \le \binom{n}{s}2^{-n}$. Clearly only contributions for s close to $\frac{n}{2}$ are substantial, in particular $\sum_{0<\sigma<\frac25} p_{\sigma,1} + \sum_{\frac35<\sigma\le1} p_{\sigma,1} = o(1/n)$.

We call a witness with parameters $\tau=1$ and $\frac25 \le \sigma \le \frac35$ *bad* and denote by w^* the number of bad witnesses. We have just seen $\Pr[w>0] = \Pr[w^*>0]+o(1)$. We aim for an ε-improvement of the union bound $\Pr[w^*>0] \le \mathbb{E}[w^*] \le \sum_{\frac25\le\sigma\le\frac35}\binom{n}{s}2^{-n} \le 1$. Now let w_1^* and w_2^* be the number of bad witnesses W with $W\cap\{1,2\}=\{1\}$ and $W\cap\{1,2\}=\{2\}$, respectively. Due to the symmetry between all rows of $A^L_{n,n}$ and the size restriction on bad witnesses, we have

$$\Pr[w_1^*>0] = \Pr[w_2^*>0] \ge (\tfrac25)^2\Pr[w^*>0].$$

Note that once all rows with indices $\{3,\dots,n\}$ are fixed, the values of w_1^* and w_2^* are stochastically independent. Let $\mathcal{Z}$ be the set of all possibilities for the rows with indices $\{3,\dots,n\}$, or more precisely, the corresponding events. Then

$$\begin{aligned}\Pr[w_1^*>0\wedge w_2^*>0] &= \sum_{z\in\mathcal{Z}}\Pr[w_1^*>0\wedge w_2^*>0 \mid z]\cdot\Pr[z]\\ &= \sum_{z\in\mathcal{Z}}\Pr[w_1^*>0\mid z]^2\cdot\Pr[z] \ge \Pr[w_1^*>0]^2\end{aligned}$$

where the last inequality uses the convexity of $x\mapsto x^2$. Putting both together we obtain

$$\Pr[w^*>1] \ge \Pr[w_1^*>0\wedge w_2^*>0] \ge \Pr[w_1^*>0]^2 \ge (\tfrac25)^4\Pr[w^*>0]^2.$$

We can now conclude

$$\begin{aligned} 1 \geq \mathbb{E}[w^*] &= \sum_{i \in \mathbb{N}_0} \Pr[w^* > i] \geq \Pr[w^* > 0] + \Pr[w^* > 1] \\ &\geq \Pr[w^* > 0](1 + (\tfrac{2}{5})^4 \Pr[w^* > 0]). \end{aligned}$$

In particular there is an absolute constant ε (roughly $\varepsilon = (\frac{2}{5})^4$) such that $\Pr[w^* > 0] \leq 1 - \varepsilon$ which implies the claim. ◀

The proof of Theorem B **(i)** is now immediate. By Lemma 7.2, $A^L_{n,n}$ admits a witness with parameter $\tau = 1$ with probability at most $1 - \varepsilon$. Since $L \geq \beta \log n$, whenever $\tau \neq 1$ we have $\tau \in [\beta \log(n)/n, 1 - \beta \log(n)/n]$. By Cases 2 and 3 from Section 7.1.1, witnesses with $\tau \neq 1$ therefore occur with probability at most $O(n^{-1})$. The probability for no witness to exist, and thus for $A^L_{n,n}$ to be regular, is therefore at least $\varepsilon - O(n^{-1}) = \Theta(1)$.

7.1.3. Proof of Theorem B (ii)

Let $\xi := \max\{2^{-\gamma L}, \beta \log(n)/n\}$. We have $\bar{c} \geq \xi$ by assumption. To bound $p_{\sigma,\tau}$ for $\tau \in [1-\xi, 1]$, for $\tau \in [1/2, 1-\xi]$ and for $\tau \in [\xi, 1/2]$ we may apply Cases 1,2 and 3, respectively. If $L = \omega(\log n)$ then $t = 1$ corresponds to $\tau = L/n \geq \beta \log(n)/n = \xi$ meaning the entire range of τ is already covered. For more interesting values of $L = \Theta(\log n)$, we require Cases 4,5 and 6. Note that $1/L \geq \xi$, so the domains of applicability of Cases 3 and 4 overlap. The domains of Cases 4 and 5 overlap since $L^2 t^2 \leq \frac{n}{2\mathrm{e}}$ corresponds to $\tau = Lt/n = O(n^{-1/2})$.

The probability for the existence of a witness summed over all six cases is

$$O(n^{-1}) + O(n^{-1}) + O(n^{-1}) + O(n^{-1}) + O(n2^{-2L}) + O(L2^{-L})).$$

Using the assumption $2L \geq (1+\delta)\log n$, and thus $2^{2L} \geq n^{1+\delta}$ this is $\widetilde{O}(n^{-\min(1,\delta)})$ as required. The '$\sim$' accounts for the log-factor in the special case $\delta = 1$, where the bound is $O(\log(n)/n)$.

7.1.4. Proof of Theorem B (iii)

We only outline the required adjustments to previous arguments instead of giving a stand-alone proof. Recall that $A^{L*}_{cn,n}$ differs from $A^L_{cn,n}$ only in that all-zero patterns are forbidden. Inconveniently, the probability for uniformly random patterns $p_1, \ldots, p_k \in \{0,1\}^L - \{0^L\}$ to sum to zero is now

$$\begin{aligned} &\Pr[p_1 \oplus \ldots \oplus p_k = \vec{0}] \\ =&\Pr[p_k = p_1 \oplus \ldots \oplus p_{k-1} \mid p_1 \oplus \ldots \oplus p_{k-1} \neq \vec{0}] \cdot \Pr[p_1 \oplus \ldots \oplus p_{k-1} \neq \vec{0}] \\ \leq&\Pr[p_k = p_1 \oplus \ldots \oplus p_{k-1} \mid p_1 \oplus \ldots \oplus p_{k-1} \neq \vec{0}] = 1/(2^L - 1) \end{aligned}$$

instead of 2^{-L}. It is however easy to check that this change is too insignificant to make a qualitative difference in the calculations from Section 7.1.1.

The big upside of $A^{L*}_{cn,n}$ is that **(c)** can be strengthened from $t \leq s+1$ to $t \leq s$. Essentially, since a minimal witness must involve each block at least twice, the extremal cases are then not tree-like with $s = t - 1$, but cycle-like with $s = t$. The dominating term in Case 5 is then not the one for $s = 1$ and $t = 2$ but the one for $s = t = 2$ and bounded by $\frac{2c^2\mathrm{e}^4 L^2}{(2^L-1)^2}$.

Otherwise, we use Cases 1,2,3,4 and 6 essentially unchanged. The probability for the existence of a witness summed over all six cases is

$$O(n^{-1}) + O(n^{-1}) + O(n^{-1}) + O(n^{-1}) + O(L^2 2^{-2L}) + O(L2^{-L})$$

which implies the claim.

7.1.5. Proof of Lemma 7.1

Proof. **(a)** This follows from Equation (7.1) after taking logarithms and multiplying by L/n on both sides, using the standard approximation $\log \binom{n}{k} \leq nH(\frac{k}{n})$.

(b) This is obtained from **(a)** by observing that $H(\sigma) + \sigma \log \tau^2$ is concave as a function of σ and assumes its unique maximum value at $\sigma^* = \sigma^*(\tau) = \frac{\tau^2}{1+\tau^2}$.

(c) Consider the graph $G = (B, \{\{b_1(w), b_2(w)\} \mid w \in W\})$ with t vertices and s edges. If $t > s+1$ then G has (at least) two connected components (B_1, W_1) and (B_2, W_2). Clearly, if the rows corresponding to W sum to zero then the rows corresponding to W_1 and W_2 do so individually, as they involve disjoint sets of variable blocks. In that case, W is not a minimal witness.

(d) Since $g(\tau) = \log(1+\tau^2)$ is convex on $[0,1]$, we may the obtain upper bounds on g by linearly interpolating between the values $g(0) = 0$, $g(\frac{1}{2}) = \log \frac{5}{4}$ and $g(1) = 1$.

(e) Using that $g(\tau) = -\log \tau$ is convex we may obtain bounds on g by linearly interpolating between the values $g(1/2) = 1$, $g(1) = 0$.

(f) The function $g(\tau) = -\tau \log \tau$ is clearly continuous and its unique maximum is easily determined to be at $\tau = 1/\mathrm{e} > 1/4$, which implies the claim.

(g) $H(\tau) = -\tau \log \tau - \bar{\tau} \log \bar{\tau} \leq -\tau \log \tau - \log \bar{\tau} \overset{\text{(e)}}{\leq} -\tau \log \tau + 2\tau$.

(h) These properties of the entropy function are well known and easily checked.

(j) We are in the process of proving Theorem B. The claims **(i)** and **(ii)** require $L = \Omega(\log n)$. In claim **(iii)** the upper bound $O(L \cdot 2^{-L})$ on the failure probability is trivial for constant L.

(k) From $\sigma cn = s \geq t = \tau n/L$ we get $\sigma \geq \frac{\tau}{cL}$. Using the upper bound on τ we continue with $\sigma \geq \frac{\tau^2}{c} \geq \frac{\tau^2}{1+\tau^2}$. This means that all values permitted for σ exceed the argument $\sigma^* = \frac{\tau^2}{1+\tau^2}$ from **(b)** that maximises $H(\sigma) + \sigma \log \tau^2$. Again by concavity of this function we may refine the upper bound from **(a)** by substituting the smallest admissible value $\sigma = \frac{\tau}{cL}$. This yields:

$$\begin{aligned}
\tfrac{L}{n} \log(p_{\sigma,\tau}) &\leq cLH(\tfrac{\tau}{cL}) + 2\tau \log \tau + H(\tau) - L\tau \\
&\overset{\text{(g)}}{\leq} -\tau \log(\tfrac{\tau}{cL}) + 2\tau + \tau \log \tau + 2\tau - L\tau \\
&= \tau \log(cL) + 4\tau - L\tau = \tau(\log(cL) - L + 4) \overset{\text{(j)}}{\leq} -\tau L/2.
\end{aligned}$$

◀

7.2. Proof of Theorem B1

To derive Theorem B1 from Theorem B, we follow the general strategy from Section 3.4.

Proof of Theorem B1. **(i)** Let $S \subseteq \mathcal{U}$ of size $m = |S|$ and $f\colon S \to \{0,1\}^r$ be given. We pick $L := \lceil \beta \log m \rceil$, $n = m$ and fully random hash functions $b_1, b_2 \leftarrow [n/L]^{\mathcal{U}}$

and $p_1, p_2 \leftarrow\!\!\boxed{\$}\!\!- (\{0,1\}^L)^{\mathcal{U}}$ which defines another hash function $\vec{e}\colon \mathcal{U} \to \{0,1\}^n$ via $\vec{e}(x) = B_{b_1(x),p_1(x)} \oplus B_{b_2(x),p_2(x)}$ using notation from Definition 2.5. The matrix A with rows $\vec{e}(x)$ for $x \in S$ has distribution $A \stackrel{\mathrm{d}}{=} A^L_{n,n}$. By Theorem B **(i)** A has full rank with constant probability. (At first, this requires that L divides n. If L does not divide n, we may mix blocks of sizes L and $L+1$. It is intuitively clear that Theorem B **(i)** still applies.) If A has full rank, we can solve the linear system arising in **construct** (see Figure 3.5) for $Z \in \{0,1\}^{n \times r}$. If A does not have full rank, construction may fail and may have to be restarted with fresh hash functions an expected constant number of times. Let s be the number of failed trials.

We store Z, the number $n = m$ and s. Note that n and s cannot be omitted as they are implicit parameters of the hash function $\vec{e}$.

The time for a construction attempt is dominated by the time to solve the linear system. We use the Method of Four Russians described in Section 4.4.3 which takes time $O(\frac{m^3}{w \log m})$. An $\mathbf{eval}_2$ (see Figure 3.5) has to evaluate $\vec{e}$ and access two blocks of size $L \times r$ in Z. Two cache misses per **eval** suffice if blocks of Z of size $L \times r$ are stored contiguously.

A sequence of L bits fits into $\lceil L/w \rceil = O(1)$ memory words. Each bit in the output can be computed by $\mathbf{eval}_2$ with $O(1)$ bit-wise AND operations as well as a PARITY operation, assuming that the $L \times r$ blocks of Z are stored column-wise. The running time of $\mathbf{eval}_2$ is thus $O(r)$.

(ii) We proceed as in **(i)**, but use Wiedemann's algorithm (see Section 4.4.1) to solve the system $A\vec{Z} = (f(x))_{x \in S}$. For $r = 1$ the running time is $\widetilde{O}(m^2)$ by Theorem 4.2. For $r > 1$, the algorithm must be repeated r times.

(iii) We proceed as in **(i)**, but use $L := 4\lceil \log m \rceil$ and let n be the least multiple of L exceeding $m + (\beta+1)\log m$. We obtain a matrix distributed like $A^L_{m,n}$ which has full rank whp by Theorem B **(ii)**.

This introduces an overhead of $r(n-m) \le r(\beta+5)\log m$ bits. On the upside, all blocks have the same size, the block size is smaller (by a constant factor), and construction does not have to be restarted whp. ◀

Relevant Definition and Theorems for Chapter 8
(Originally on Pages 19, 39 and 41)

► **Definition 2.6** (Random Matrix with One Unaligned Block per Row). *Let $n, m, L \in \mathbb{N}$ with $m \leq n$. For a* starting position *$s \in [n]$ and a* pattern *$p \in \{0,1\}^L$ we let $B_{s,p} = 0^{s-1} \circ p \circ 0^{n-s} \in \{0,1\}^{n+L-1}$. To obtain the random matrix $M^L_{m,n} \in \{0,1\}^{m\times(n+L-1)}$, each row is independently sampled as $B_{s,p}$ where $s \leftarrow [n]$ and $p \leftarrow \{0,1\}^L$.*

$$\begin{array}{c} \begin{array}{ccccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \end{array} \\ \begin{pmatrix} & 0 & 1 & 1 & 0 & 0 & & & & & & & \\ 1 & 0 & 1 & 0 & 0 & & & & & & & & \\ & & & 1 & 0 & 0 & 0 & 1 & & & & & \\ & & & & & & & & 1 & 0 & 1 & 0 & 1 \\ & 0 & 1 & 0 & 1 & 1 & & & & & & & \\ & & & & & 1 & 1 & 1 & 0 & 0 & & & \end{pmatrix} \end{array}$$

Possible outcome for $M^5_{6,9}$.

► **Theorem C.** *Let $0 < \varepsilon < \frac{1}{2}$ be a constant, $n \in \mathbb{N}$ and $m = (1-\varepsilon)n$. For some $L = O((\log n)/\varepsilon)$ the matrix $M^L_{m,n}$ has independent rows with probability $1 - O(1/n)$.*
Moreover, a simple algorithm computes a solution $\vec{z}$ to $M^L_{m,n}\vec{z} = \vec{b}$ for an arbitrary right hand side $\vec{b} \in \{0,1\}^m$. Its expected running time is dominated by $O(n/\varepsilon)$ row additions. Each row is always zero outside of a block of length L.

► **Theorem C1.** *We aim to build a retrieval data structure for $f : S \to \{0,1\}^r$ where $|S| = m$. Let $\varepsilon > 0$. When using $A = M^L_{m,n}$ from Theorem C for suitable n and $L = O(\frac{\log m}{\varepsilon})$ in the retrieval framework from Section 3.4, then the following holds.*

- **(i)** *One trial of* **construct** *succeeds whp and has expected running time $O(m/\varepsilon^2 + rm/\varepsilon)$.*
- **(ii)** *The resulting data structure occupies at most $(1+\varepsilon)rm$ bits whp.*
- **(iii)** *An eval-operation (using* $\mathbf{eval}_2$*) takes time $O(r/\varepsilon)$ and causes one cache miss when Z is stored appropriately.*

► **Theorem C2.** *We aim to build a retrieval data structure for $f : S \to \{0,1\}^r$ where $|S| = m$. For $\varepsilon > 0$ and chunks of size $C = m^\varepsilon$, the following performance characteristics can be achieved.*

- **(i)** *One trial of* **construct** *succeeds whp and has expected running time $O(m/\varepsilon + rm)$.*
- **(ii)** *The resulting data structure occupies at most $(1+\varepsilon)rm$ bits whp.*
- **(iii)** *An* **eval** *takes time $O(r)$ and causes one cache miss.*

8. Efficient Gauss Elimination for Near-Quadratic Matrices with One Short Random Block per Row, with Applications

The purpose of this chapter is to prove Theorems C, C1 and C2 (see facing page).

Throughout this section $0 < \varepsilon < \frac{1}{2}$ is a constant, $m, n \in \mathbb{N}$ satisfy $m = (1-\varepsilon)n$ and $L \in \mathbb{N}$ is a number later chosen to be $O((\log m)/\varepsilon)$. The random matrix $M^L_{m,n}$ from Definition 2.6 is denoted simply by $A = (a_{ij})_{i\in[m],\, j\in[n+L-1]}$. The random *starting position* of the i-th row is denoted by $s_i \in [n]$. The entries a_{ij}, $s_i \le j < s_i + L$ form a *block* of fully random bits, all other entries in row i are 0.

In asymptotic considerations, ε is constant and m and n tend to ∞, but O-notation only hides absolute constants that do not depend on ε.

Most of this chapter is spent proving Theorem C, i.e. that for a proper choice of L, the matrix A has full row rank whp and a corresponding systems $A\vec{z} = \vec{b}$ is solvable very efficiently whp. Theorems C1 and C2 are fairly simple consequences. Before giving a sketch of the main ideas of the proof, we review a simple but useful technique.

Technique: Coupling of random variables. A standard notion frequently used in this chapter (and sporadically used previously) is that of a *coupling* of random variables X and Y (or of processes $(X_i)_{i\ge1}$ and $(Y_i)_{i\ge1}$). A coupling is a single probability space on which X and Y are defined, so that there are interesting *pointwise* relations between them, like $X \le Y$. Sometimes these relations hold only conditioned on some (large) part of the probability space. We will make use of the following observation. If we have random variables $U_0, \dots, U_k$ and for each $\ell \in [k]$ a coupling, i.e. a joint distribution, of $U_{\ell-1}$ and U_ℓ, then there is a common probability space on which all these random variables are defined and all couplings are simultaneously realised. [1]

8.1. Proof Sketch for Theorem C

We start by formulating a special version of Gaussian elimination for solving linear systems $A\vec{z} = \vec{b}$ as just described. The algorithm sorts the rows of A by the starting position of their

[1] We do not prove this formally since arguments like this belong to basic probability theory or measure theory. The principle used is that the pairwise couplings give rise to conditional expectations $\mathbb{E}[U_\ell \mid U_{\ell-1}]$. Arguing inductively, given a common probability space for $U_1, \dots, U_{\ell-1}$ and $\mathbb{E}[U_\ell \mid U_{\ell-1}]$, one can obtain a common probability space for $U_1, \dots, U_\ell$ so that $(U_1, \dots, U_{\ell-1})$ is distributed as before and $\mathbb{E}[U_\ell \mid U_1, \dots, U_{\ell-1}] = \mathbb{E}[U_\ell \mid U_{\ell-1}]$. – This is practically the same as the standard argument that shows that a sequence of conditional expectations gives rise to a corresponding Markov chain on a joint probability space.

block. The resulting matrix resembles a band matrix, and standard Gaussian elimination is applied to it, treating the rows in order of their starting position. Conveniently, there is no "proliferation of 1's", i.e. we never produce a 1-entry outside of any row's original block. In the round for row i, the entries a_{ij} for $j = s_i, \dots, s_i + L - 1$ are scanned. If column j has been previously chosen as pivot then $a_{ij} = 0$. Otherwise, a_{ij} is a random bit. While this bit may depend in a complex way on the original entries of rows $1, \dots, i$ (apart from position (i, j)), for the analysis we may simply imagine that a_{ij} is only chosen now by flipping a fair coin. This means that we consider eligible columns from left to right, and the first j for which the coin flip turns up 1 becomes the pivot column for row i. This view makes it possible to regard choosing pivot columns for the rows as probabilistically equivalent to a slightly twisted version of Robin Hood hashing. Here this means that m keys $x_1, \dots, x_m$ with random hash values in $[n]$ are given and, in order of increasing hash values, are inserted in a linear probing fashion into a table, meaning that for x_i cells $s_i, s_{i+1}, \dots$ are inspected. The twist is that, whenever a key probes an empty table cell, flipping a fair coin decides whether it is placed in the cell or moves on to the next. The resulting position of key x_i is the same as the position of the pivot for row i. As is standard in the precise analysis of linear probing hashing, we switch perspective and look at the process from the point of view of cells $1, 2, \dots, n + L - 1$. Associated with position ("time") j is the set of keys that probe cell j (the "queue"), and the quantity to study is the length of this queue. It turns out that the average queue length determines the overall cost of the row additions of the Gaussian elimination process, and that the probability for the maximum queue length to become too large is decisive for bounding the success probability. The first and routine step in the analysis of the queue length is to "Poissonise" arrivals such that the evolution of the queue length becomes a Markov chain. A second step is needed to deal with the somewhat annoying possibility that in a cell, all keys that are eligible for this cell reject it because of their coin flips. We end up with a standard queue (an "M/D/1 queue" in Kendall notation) and can use existing results from queuing theory to obtain bounds regarding the queue length, which are needed to complete the analysis.

The following sections give the details.

8.2. A Simple Gaussian Solver

We now describe the algorithm to solve linear systems involving the random matrices described above. This is done by a variant or Gauss elimination, which will bring the matrix into echelon form (up to column exchanges) and then apply back substitution.

Given $A = (a_{ij})_{i \in [m],\, j \in [n+L-1]}$ as defined above, with blocks of length L starting at positions s_i, for $i \in [m]$, as well as some $\vec{b} \in \{0,1\}^m$, we wish to find a solution $\vec{z}$ to the system $A\vec{z} = \vec{b}$. Consider algorithm SGAUSS (Figure 8.1). If A has linearly independent rows, SGAUSS returns a solution $\vec{z}$ and produces intermediate values $(\mathrm{piv}_i)_{i \in [m]}$, which will be important in the analysis. If the rows of A are linearly dependent, the algorithm fails.

Algorithm SGAUSS starts by sorting the rows of the system $(A, \vec{b})$ by their starting positions s_i in linear time, e.g. using counting sort [Cor+09, Chapter 8.2]. We suppress the resulting permutation in the notation, assuming $s_1 \le s_2 \le \cdots \le s_m$. Rows are then processed sequentially. When row i is treated, its leftmost 1-entry is determined, if it exists, and the corresponding column index is called the *pivot* piv_i of row i. Row additions are used to eliminate 1-entries from column piv_i in subsequent rows. Note that this operation never

```
1  Algorithm SGAUSS(A = (a_ij)_{i∈[m], j∈[n+L−1]}, (s_i)_{i∈[m]}, b⃗ ∈ {0,1}^m):
2      sort the rows of the system (A, b⃗) by s_i (in time O(m))
3      relabel such that s_1 ≤ s_2 ≤ ··· ≤ s_m
4      piv_1, piv_2, ..., piv_m ← 0
5      for i = 1, ..., m do
6          for j = s_i, ..., s_i + L − 1 do      } // search for leftmost 1 in row i. Can be done
7              if a_ij = 1 then                  } // in time O(L/log m) on a word RAM.
8                  piv_i ← j
9                  for i' with i' > i ∧ s_i' ≤ piv_i do
10                     if a_{i',piv_i} = 1 then
11                         a_i' ← a_i' ⊕ a_i   // row addition (= subtraction)
12                         b_i' ← b_i' ⊕ b_i
13                 break
14         if piv_i = 0 then // row i is 0
15             return FAILURE
       // back substitution:
16     z⃗ ← 0⃗
17     for i = m, ..., 1 do
18         z⃗_{piv_i} ← ⟨z⃗, a_i⟩ ⊕ b_i   // note: a_ij = 0 for j outside of {s_i, ..., s_i + L − 1}
19     return z⃗ // solution to Az⃗ = b⃗
```

Figure 8.1 A simple Gaussian solver for the One-Block construction.

produces non-zero entries outside of any row's original block, i.e. for no row i are there ever any 1's outside of the positions $\{s_i, \dots, s_i + L - 1\}$. To see this, we argue inductively on the number of additions performed. Assume $1 \le i \le i' \le m$ and row i is added to row i'. By choice of piv_i and the induction hypothesis, non-zero entries of row i can reside only in positions $\text{piv}_i, \dots, s_i + L - 1$. Again by induction and since row i' contains a 1 in position piv_i, we have $s_{i'} \le \text{piv}_i$. Due to sorting, we have $s_i + L - 1 \le s_{i'} + L - 1$. Thus, row i contains no 1's outside of the block of row i', and the row addition maintains the invariant.

If an all-zero row is encountered, the algorithm fails (and returns FAILURE). This happens if and only if the rows of A are linearly dependent[2]. Otherwise, we say that the algorithm succeeds. In this case a solution $\vec{z}$ to $A\vec{z} = \vec{b}$ is obtained by back-substitution. It is not hard to see that the expected running time of SGAUSS is dominated by the expected cost of row additions.

8.3. Coin-Flipping Robin Hood Hashing

Let $\{x_1, \dots, x_m\} \subseteq \mathcal{U}$ be some set of *keys* to be stored in a hash table T. Each key x_i has a uniformly random *hash value* $h_i \leftarrow [n]$. An (injective) placement of the keys in T fulfils

[2] Depending on $\vec{b}$, the system $A\vec{z} = \vec{b}$ may still be solvable. We will not pursue this.

the *linear probing* requirement if each x_i is stored in a cell $T[\mathrm{pos}_i]$ with $\mathrm{pos}_i \geq h_i$ and all cells $T[j]$ for $h_i \leq j < \mathrm{pos}_i$ are non-empty. In *Robin Hood hashing* there is the additional requirement that $h_i > h_{i'}$ implies $\mathrm{pos}_i > \mathrm{pos}_{i'}$. Robin Hood hashing is interesting because it minimises the variance of the displacements $\mathrm{pos}_i - h_i$. It has been studied in detail in several papers [CLM85; DMV04; Jan05; JV16; Vio05].

Given the hash values $(h_i)_{i\in[m]}$, a placement of the keys obeying the Robin Hood linear probing conditions can be obtained as follows: Insert the keys in the order of increasing hash values, by the usual linear probing insertion procedure, which probes (i.e. inspects) cells $T[h_i], T[h_i+1], \ldots$ until the first empty cell is found, and places x_i in this cell. We consider a slightly "broken" variation of this method, which sometimes delays placements. In the placing procedure for x_i, when an empty cell $T[j]$ is encountered, it is decided by flipping a fair coin whether to place x_i in cell $T[j]$ or move on to the next cell. Note that the resulting placement may violate the Robin Hood requirement and even the linear probing requirement. For this insertion method we assume we have an (idealised) unbounded array $T[1, 2, \ldots]$. The position in which key x_i is placed is called pos_i. In the end, the algorithm checks whether any of the displacements $\mathrm{pos}_i - h_i$ is larger than L, in which case it reports FAILURE.[3] Figure 8.2 gives a precise description of this algorithm, which we term CFRH.

```
1  Algorithm CFRH ({x_1, ..., x_m} ⊆ U):
2      sort x_1, ..., x_m by hash value h_1, ..., h_m
3      relabel such that h_1 ≤ ··· ≤ h_m
4      T ← [⊥, ⊥, ...] // infinite array of empty cells
5      pos_1, ..., pos_m ← 0
6      for i = 1, ..., m do
7          for j = h_i, h_i + 1, ... do
8              if T[j] = ⊥ ∧ coinFlip() = 1 then // empty cell and "HEADS"
9                  pos_i ← j
10                 T[j] ← x_i
11                 break
12     if ∃i ∈ [m] : pos_i − h_i ≥ L then return FAILURE
13     return T
```

Figure 8.2 The *Coin-Flipping Robin Hood hashing* algorithm. Without the condition "coinFlip() = 1", it would compute a Robin Hood placement with maximum displacement L, if one exists.

8.4. Connection between SGAUSS and CFRH

We now establish a close connection between the behaviour of the algorithms SGAUSS and CFRH, thus reducing the analysis of SGAUSS to that of CFRH. The algorithms have been formulated in such a way that some structural similarity is immediate. A run of

[3] The reason we postpone checking for FAILURE until the very end of the execution is that it is technically convenient to have the values $(\mathrm{pos}_i)_{i\in[m]}$ even if a failure occurs.

SGAUSS on a matrix with random starting positions $(s_i)_{i\in[m]}$ and random entries yields a sequence of pivots $(\mathrm{piv}_i)_{i\in[m]}$; a run of CFRH on a key set with random hash values $(h_i)_{i\in[m]}$ performing random coin flips yields a sequence of positions $(\mathrm{pos}_i)_{i\in[m]}$. We will see that the distributions of $(\mathrm{piv}_i)_{i\in[m]}$ and $(\mathrm{pos}_i)_{i\in[m]}$ are essentially the same and that moreover two not so obvious parameters of the two random processes are closely connected. For this, we will show that outside the FAILURE events, we can use the probability space underlying SGAUSS to describe the behaviour of CFRH. This yields a coupling of the underlying random processes.

The first step is to identify $s_i = h_i$ for $i \in [m]$ (both sequences are assumed to be sorted and then renamed). The connection between pos_i and piv_i is achieved by connecting the coin flips of CFRH to certain events in applying SGAUSS to matrix A. We construct this correspondence by induction on i. Assume rows $1,\dots,i-1$ have been treated, $x_1,\dots,x_{i-1}$ have been placed, and $\mathrm{piv}_{i'} = \mathrm{pos}_{i'}$ for all $1 \le i' < i$.

Now row a_i (transformed by previous row additions) is treated. It contains a 0 in columns that were previously chosen as pivots, so possible candidates for piv_i are only indices from $J_i := \{s_i,\dots,s_i{+}L{-}1\}\setminus\{\mathrm{piv}_1,\dots,\mathrm{piv}_{i-1}\}$. For each $j \in J_i$, the initial value of a_{ij} was a random bit. The bits added to a_{ij} in rounds $1,\dots,i-1$ are determined by the original entries of rows $1,\dots,i-1$ alone. We order the entries of J_i as $j^{(1)} < j^{(2)} < \cdots < j^{(|J_i|)}$. Then, conditioned on all random choices in rows $1,\dots,i-1$ of A, the current values $a_{i,j^{(1)}},\dots,a_{i,j^{(k)}}$ still form a sequence of fully random bits. We use these random bits to run round i of CFRH, in which x_i is placed. Since each cell can only hold one key, and by excluding runs where finally FAILURE is declared, we may focus on the empty cells with indices in $\{h_i,\dots,h_i{+}L{-}1\}\setminus\{\mathrm{pos}_1,\dots,\mathrm{pos}_{i-1}\} = \{s_1,\dots,s_i{+}L{-}1\}\setminus\{\mathrm{piv}_1,\dots,\mathrm{piv}_{i-1}\} = J_i$. We use (the current value) a_{ij} as the value of the coin flip for cell j, for $j = j^{(1)}, j^{(2)},\dots,j^{(|J_i|)}$. The minimal j in this sequence (if any) with $a_{ij} = 1$ equals piv_i and pos_i. If all these bits are 0, algorithm SGAUSS will fail immediately, and key x_i will be placed in a cell $T[j]$ with $j \ge h_i + L$, so CFRH will eventually fail as well.

Thus we have established that the random variables needed to run algorithm CFRH (outside of FAILURE) can be taken to belong to the probability space defined by $(s_i)_{i\in[m]}$ and the entries in the blocks of A for algorithm SGAUSS, so that (outside of FAILURE) the random variables pos_i and piv_i are the same. In the following lemma, we state this connection as Claim **(i)**. In addition, we consider other random variables central to the analysis to follow. First, we define the *height* of position $j \in [n+L-1]$ in the hash table as

$$H_j := \#\{i \in [m] \mid h_i \le j < \mathrm{pos}_i\}.$$

This is the number of keys probing table cell j without being placed in it, either because the cell is occupied or because it is rejected by the coin flip. Claim **(ii)** in the next lemma shows that $\sum_{j\in[n+L-1]} H_j$ essentially determines the running time of SGAUSS, so that we can focus on bounding $(H_j)_{j\in\mathbb{N}}$ from here on. Finally, with Claim **(iii)**, we get a handle on the question of how large we have to choose L in order to keep the failure probability small.

► **Lemma 8.1.** *With the coupling just described, we get*

(i) SGAUSS *succeeds iff* CFRH *succeeds. On success we have* $\mathrm{piv}_i = \mathrm{pos}_i$ *for all* $i \in [m]$.

(ii) *A successful run of* SGAUSS *performs at most* $\sum_{j\in[n+L-1]} H_j$ *row additions.*

(iii) *Conditioned on the event* $\max_{j\in[n]} H_j \le L - 2\log m$, *the algorithms succeed with probability* $1 - O(1/m)$.

Proof. **(ii)** (Note that a similar statement with a different proof can be found in [Jan08, Lemma 2.1].) Consider the sets $\mathrm{Add} := \{(i,i') \in [m]^2 \mid \text{SGAUSS adds row } i \text{ to row } i'\}$ and $\mathrm{Displ} := \{(i,j) \in [m] \times [n+L-1] \mid h_i \le j < \mathrm{pos}_i\}$. Since H_j simply counts the pairs $(i,j) \in \mathrm{Displ}$ with $i \in [m]$, we have $|\mathrm{Displ}| = \sum_{j\in[n+L-1]} H_j$. To prove the claim, we exhibit an injection from Add into Displ.

Assume $(i,i') \in \mathrm{Add}$. If $\mathrm{pos}_i < \mathrm{pos}_{i'}$, we map (i,i') to (i', pos_i). This is indeed an element of Displ, since $h_{i'} = s_{i'} \le \mathrm{piv}_i = \mathrm{pos}_i < \mathrm{pos}_{i'}$ (if piv_i were smaller than $s_{i'}$, row i would not be added to row i'). On the other hand, if $\mathrm{pos}_i > \mathrm{pos}_{i'}$, we map (i,i') to $(i, \mathrm{pos}_{i'})$. This is in Displ since $h_i = s_i \le s'_i \le \mathrm{pos}_{i'} < \mathrm{pos}_i$ (recall that rows are sorted by starting position).

The mapping is injective since from the image of $(i,i') \in \mathrm{Add}$ we can recover $\{i,i'\}$ with the help of the injective mapping $i \mapsto \mathrm{pos}_i$, $i \in [m]$. The fact that $i < i'$ fixes the ordering in the pair.

(iii) In CFRH, for an arbitrary $i \in [m]$, consider the state before key x_i probes its first position $j := h_i$. Any previous key $x_{i'}$ with $i' < i$ has a hash value $h_{i'} \le h_i$. Hence, it either was inserted in a cell $j' < j$ or it has probed cell j. Since at most H_j keys have probed cell j, at most H_j positions in $T[j, \dots, j+L-1]$ are occupied and at least $2 \log m$ are free. The probability that x_i is not placed in this region is therefore at most $2^{-2\log m} = m^{-2}$. By the union bound we obtain a failure probability of $O(1/m)$. ◀

8.5. Bounding Heights in CFRH by a Markov Chain

Lemma 8.1 tells us that we must analyse the heights in the hashing process CFRH. In this subsection, we use "Poissonisation" of the hashing positions to majorise the heights in CFRH by a Markov chain, i.e. a process that is oblivious to the past, apart from the current height. Poissonisation is a common step in the analysis of linear probing hashing, see e.g. [Vio05]. Further, we wish to replace randomized placement by deterministic placement: Whenever a key is available for a position, one is put there (instead of flipping coins for all available keys). By this, the heights may decrease, but only by a bounded amount whp. The details of these steps are given in this subsection.

In analysing CFRH (without regard for the event Failure), it is inconvenient that the starting positions h_i are determined by random choices with subsequent sorting. Position j is hit by a number of keys given by a binomial distribution $\mathrm{Bin}(m, \frac{1}{n})$ with expectation $\frac{m}{n} = 1-\varepsilon$, but there are dependencies. We approximate this situation by "Poissonisation" [MU05, Sect. 5.4]. Here this means that we assume that cell $j \in [n]$ is hit by k_j keys, independently for $j = 1, \dots, m$, where $k_j \sim \mathrm{Po}(1-\varepsilon')$ is Poisson distributed, for $\varepsilon' = \varepsilon/2$. Then the total number $m' = \sum_{j\in[n]} k_j$ of keys is distributed as $m' \sim \mathrm{Po}((1-\varepsilon')n)$. Given $k_1, \dots, k_n$, we can imagine we have m' keys with nondecreasing hash values $(h_i)_{i\in[m']}$, and we can apply algorithm CFRH to obtain key positions $(\mathrm{pos}'_i)_{i\in[m']}$ and cell heights $(H'_j)_{j\ge1}$.

Conveniently, with Poissonisation, the heights $(H'_j)_{j\in[n]}$ turn out to form a Markov chain. This can be seen as follows. Recall that H'_{j-1} is the number of keys probing cell $j-1$ without being placed there. Hence the number of keys probing cell j is $H'_{j-1} + k_j$. One of these keys will be placed in cell j, unless $H'_{j-1} + k_j$ coin flips all yield 0, so if $g_j \sim \mathrm{Geom}(\frac{1}{2})$ is a random variable with geometric distribution with parameter $\frac{1}{2}$ (number of fair coin flips needed until the first 1 appears) and $b_j = \mathbb{1}[g_j > H'_{j-1} + k_j]$ is an indicator random variable, we have $H'_j = H'_{j-1} + k_j - 1 + b_j$. (Note that the case $H'_{j-1} + k_j = 0$ is treated

correctly by this description. Conditioned on $H'_{j-1} + k_j$, the value b_j is a Bernoulli variable.) The Markov property holds since H'_j depends only on H'_{j-1} and the two "fresh" random variables k_j and g_j.

The following lemma allows us to shift our attention from $(H_j)_{j\in[n+L-1]}$ to $(H'_j)_{j\in[n+L-1]}$.

► **Lemma 8.2.** *Let $m = (1-\varepsilon)n$ and $m' \sim \mathrm{Po}((1-\varepsilon')m)$ for $\varepsilon' = \varepsilon/2$. There is a coupling between an ordinary run of* CFRH *(with m, n and H_j) and a Poissonised run (with m', n and H'_j) such that conditioned on the high probability event $E_{\geq m} = \{m' \geq m\}$ we have $H'_j \geq H_j$ for all $j \in [n+L-1]$.*

Proof. Because ε and $\varepsilon' = \varepsilon/2$ are constants, the event $E_{\geq m}$ has high probability, as can be seen by well-known concentration bounds for the Poisson distribution (e.g. [MU05, Th. 5.4]). For $m_0 \geq m$ fixed, the distribution of the number of hits in the cells in $T[1,\dots,n]$ conditioned on $\{m' = m_0\}$ is the same as what we get by throwing m_0 balls randomly into n bins [MU05, Th. 5.6]. Thus, we may assume the Poissonised run has to deal with the m keys of the ordinary run plus $m' - m$ additional keys with random hash values in $[n]$. We apply algorithm CFRH to both inputs. After sorting, the new keys are inserted in some interleaved way with the ordinary keys. Now if one of the ordinary keys x probes an empty cell $T[j]$, we use the same coin flip in both runs to decide whether to place it there; for the probing of the additional keys, we use new, independent coin flips. With this coupling, it is clear that for all ordinary keys x the displacement "(position of x) − (hash value of x)" in the Poissonized run is at least as big as in the ordinary run. As the additional keys can only increase heights, $H'_j \geq H_j$ follows. ◄

It is natural and convenient to prepend $H'_0 = 0$ to the sequence of heights and remove the "termination" at n, making $(H'_j)_{j\geq 0}$ an infinite Markov chain (we refrain from changing the symbol). As a further simplification, we eliminate the geometrically distributed variable g_j and the derived variable b_j. For this, let $(X_j)_{j\geq 0}$ be the Markov chain defined as

$$X_0 \coloneqq 0 \quad \text{and} \quad X_j \coloneqq \max(0, X_{j-1} + d_j - 1) \quad \text{for } j \geq 1, \tag{8.1}$$

where $d_j \sim \mathrm{Po}(1-\varepsilon'/2)$ are independent random variables.

► **Lemma 8.3.** *There is a coupling between $(X_j)_{j\geq 0}$ and $(H'_j)_{j\geq 0}$ such that $X_j + \log(4/\varepsilon') \geq H'_j$ for all $j \geq 0$.*

Proof. Assume wlog that $\log(1/\varepsilon')$ is an integer. Let $b'_j \sim \mathrm{Po}(\varepsilon'/2)$ be a random variable on the same probability space as g_j such that $g_j > \log(4/\varepsilon')$ implies $b'_j \geq 1$. This is possible because

$$\Pr[g_j > \log(4/\varepsilon')] = 2^{-\log(4/\varepsilon')} = \varepsilon'/4 \leq 1 - e^{-\varepsilon'/2} = \Pr[b'_j \geq 1].$$

We then define $d_j \coloneqq k_j + b'_j$ which gives $d_j \sim \mathrm{Po}(1-\varepsilon'/2)$. Proceeding by induction, and using (8.1), we can define $(X_j)_{j\geq 0}$ and $(H'_j)_{j\geq 0}$ on a common probability space. Then we check $X_j + \log(4/\varepsilon') \geq H'_j$, also by induction: In the case $H'_{j-1} + k_j \leq \log(4/\varepsilon')$ we simply get

$$X_j + \log(4/\varepsilon') \geq \log(4/\varepsilon') \geq H'_{j-1} + k_j \geq H'_{j-1} + k_j + b_j - 1 = H'_j.$$

Otherwise we can use the inequality $b_j = \mathbb{1}[g_j > H'_{j-1} + k_j] \le \mathbb{1}[g_j > \log(4/\varepsilon')] \le b'_j$ to obtain

$$\begin{aligned} X_j + \log(4/\varepsilon') &\ge X_{j-1} + d_j - 1 + \log(4/\varepsilon') \overset{\text{(Ind.Hyp.)}}{\ge} H'_{j-1} + d_j - 1 \\ &= H'_{j-1} + k_j + b'_j - 1 \ge H'_{j-1} + k_j + b_j - 1 = H'_j. \end{aligned}$$ ◀

8.6. Enter Queuing Theory

It turns out that, in essence, the behaviour of the Markov chain $(X_j)_{j\ge 0}$ has been studied in the literature under the name "M/D/1 queue", which is Kendall notation [Ken53] for queues with "**M**arkovian arrivals, **D**eterministic service times and **1** server". We will exploit what is known about this simple queuing situation in order to finish our analysis.

Formally, an M/D/1 queue is a Markov process $(Z_t)_{t\in\mathbb{R}_{\ge 0}}$ in continuous time and discrete space $\mathbb{N}_0 = \{0, 1, 2, \dots\}$. The random variable Z_t is usually interpreted as the number of *customers* waiting in a FIFO queue at time $t \in \mathbb{R}_{\ge 0}$. Initially the queue is empty ($Z_0 = 0$). Customers arrive independently, i.e. arrivals are determined by a Poisson process with a rate we set to $\rho = 1 - \varepsilon'/2$ (which implies that the number of customers arriving in a time interval of length 1 is $\mathrm{Po}(\rho)$-distributed). The *server* requires one time unit to process a customer which means that if $t \in \mathbb{R}_{\ge 0}$ is the time of the first arrival, then customers will leave the queue at times $t + 1, t + 2, \dots$ until the queue is empty again.

Now consider the discretisation $(Z_j)_{j\in\mathbb{N}_0}$ of the M/D/1 queue. For $j \ge 1$, the number d_j of arrivals in between two observations Z_{j-1} and Z_j has distribution $d_j \sim \mathrm{Po}(\rho)$, and one customer was served in the meantime if and only if $Z_{j-1} > 0$. We can therefore write

$$Z_j = \begin{cases} d_j & \text{if } Z_{j-1} = 0, \\ Z_{j-1} + d_j - 1 & \text{if } Z_{j-1} > 0. \end{cases}$$

By reusing the variables $(d_j)_{j\ge 1}$ that previously occurred in the definition of $(X_j)_{j\ge 0}$, we already established a coupling between the processes $(X_j)_{j\ge 0}$ and $(Z_j)_{j\ge 0}$. A simple induction shows

$$X_j = \max(0, Z_j - 1), \quad \text{for all } j \ge 0. \tag{8.2}$$

Intuitively, the server in the X-process is ahead by one customer because customers are processed at integer times "just in time for the observation".

The following results are known in queuing theory:

▷ Fact 8.4. **(i)** The average number of customers in the Z-queue at time $t \in \mathbb{R}_{\ge 0}$ is

$$\mathbb{E}[Z_t] \le \lim_{\tau\to\infty} \mathbb{E}[Z_\tau] = \rho + \tfrac{1}{2}\left(\tfrac{\rho^2}{1-\rho}\right) = \Theta(1/\varepsilon).$$

(Precise values are known even for general service-time distributions, see [Coo90, Ch. 5.4].)

(ii) [EZB06, Prop 3.4] We have the following tail bound for the event $\{Z_t > k\}$ for any $k \in \mathbb{N}$:

$$\Pr[Z_t > k] \le \lim_{\tau\to\infty} \Pr[Z_\tau > k] = \mathrm{e}^{-k\cdot\Theta(\varepsilon)}, \text{ for all } t \ge 0.$$

8.7. Putting the Pieces Together – Proof of Theorem C

We now have everything in place to prove Theorem C.

Proof of Theorem C. For $0 < \varepsilon < \frac{1}{2}$, $m = (1-\varepsilon)n$, $\vec{b} \in \{0,1\}^m$ and L chosen later, consider an application of SGAUSS to a linear system $A \cdot \vec{z} = \vec{b}$ where $A \stackrel{\mathrm{d}}{=} M^L_{m,n}$.

By the observation made at the start of this chapter, we may assume that the random variables $(\mathrm{piv}_i)_{i\in[m]}$, $(\mathrm{pos}_i)_{i\in[m]}$, $(H_j)_{j\in[n+L-1]}$, $(H'_j)_{j\in[n+L-1]}$, $(X_j)_{j\geq 0}$ and $(Z_j)_{j\geq 0}$ and the corresponding couplings are realized on one common probability space.

By Fact 8.4 **(ii)** it is possible to choose $L = \Theta((\log n)/\varepsilon)$ while guaranteeing $\Pr[Z_j > L/2] = O(n^{-2})$ for all $j \geq 0$. By the union bound, the event $E_{\max Z} = \{\forall j \in [n+L-1] : \ Z_j \leq L/2\}$ then occurs with probability $1-O(n^{-1})$. Conditioned on $E_{\max Z}$ and the high probability event $E_{\geq m}$ from Lemma 8.2 we have for $j \in [n+L-1]$

$$H_j \overset{\text{Lem. 8.2}}{\leq} H'_j \overset{\text{Lem. 8.3}}{\leq} X_j + \log(4/\varepsilon') \overset{\text{Eq. 8.2}}{\leq} Z_j + \log(4/\varepsilon') \overset{E_{\max Z}}{\leq} L/2 + \log(4/\varepsilon') \leq L - 2\log m.$$

By using Lemma 8.1 **(iii)** we conclude that SGAUSS succeeds with probability $1 - O(m^{-1})$. Since SGAUSS only succeeds if A has independent rows, this establishes the first claim of Theorem C.

Along similar lines we get, for each $j \in [n+L-1]$:

$$\begin{aligned} E[H_j] &\overset{\text{Lem. 8.2}}{\leq} \mathbb{E}[H'_j \mid E_{\geq m}] \leq \tfrac{1}{\Pr[E_{\geq m}]}\,\mathbb{E}[H'_j] \overset{\text{Lem. 8.3}}{\leq} \tfrac{1}{\Pr[E_{\geq m}]}\,\mathbb{E}[X_j + \log(4/\varepsilon')] \\ &\overset{\text{Eq. 8.2}}{\leq} \tfrac{1}{\Pr[E_{\geq m}]}\,\mathbb{E}[Z_j + \log(4/\varepsilon')] \overset{\text{Fact 8.4 (i)}}{\leq} \tfrac{1}{\Pr[E_{\geq m}]}\,(O(1/\varepsilon) + \log(4/\varepsilon')) = O(1/\varepsilon). \end{aligned}$$

By Lemma 8.1 **(ii)** the expected number of row additions performed by a successful run of SGAUSS is therefore at most $\mathbb{E}[\sum_{j\in[n+L-1]} H_j] = O(m/\varepsilon)$. Since unsuccessful runs happen with probability $O(1/m)$ and can perform at most mL additions (each row can only be the target of L row additions), the overall expected number of additions is not skewed by unsuccessful runs, hence is also in $O(m/\varepsilon)$. This finishes the proof of Theorem C. ◀

▶ Remark. The analysis described in this section works in exactly the same way if instead of $\mathbb{F}_2$ a larger finite field $\mathbb{F}$ is used. A row in the random matrix is determined by a random starting position and a block of L random elements from $\mathbb{F}$. A row operation during Gaussian elimination consists of a division, a multiplication of a block with a scalar and a row addition. The running time of the algorithm will increase at least by a factor of $\log(|\mathbb{F}|)$ (the bitlength of a field element), and further increases depend on how well word parallelism can be utilized for operations like row additions and scalar multiplications. (In [GOV16], efficient methods are described for $\mathbb{F}_3$.) The queue length will become a little smaller, but not significantly since even the M/D/1 queue with arrivals with a Poisson$(1-\varepsilon)$ distribution will lead to average queue length $\Theta(1/\varepsilon)$.

8.8. A New Retrieval Data Structure – Proof of Theorem C1

With Theorem C in place we are ready to carry out the analysis of the retrieval data structure following the general strategy from Section 3.4.

Proof of Theorem C1. Let $0 < \varepsilon < \frac{1}{2}$, $S \subseteq \mathcal{U}$ of size $m = |S|$ and $f\colon S \to \{0,1\}^r$ be given and let $L = \Theta(\frac{\log m}{\varepsilon})$ be the number from Theorem C. We pick fully random hash functions $s \leftarrow [n]^{\mathcal{U}}$ and $p \leftarrow (\{0,1\}^L)^{\mathcal{U}}$ which define another hash function $\vec{e}\colon \mathcal{U} \to \{0,1\}^{n+L-1}$ via $\vec{e}(x) = B_{s(x),p(x)}$ using notation from Definition 2.6. The matrix A with rows $\vec{e}(x)$ for $x \in S$ has distribution $A \stackrel{\mathrm{d}}{=} M^L_{m,n}$.

Assume $r = 1$ for now. By Theorem C, A has full rank with probability $1 - O(1/n)$ and, in that case, SGAUSS solves the linear system arising in **construct** (see Figure 3.5) for $\vec{z} \in \{0,1\}^{n+L-1}$ by performing $O(m/\varepsilon)$ row additions. Since additions affect only $L = O(\frac{\log m}{\varepsilon})$ consecutive bits, and since a word RAM can deal with $O(\log m)$ bits at once, a single row addition takes time $O(1/\varepsilon)$. Back substitution has to evalute one inner product of vectors of length L for each element of $\vec{z}$, which takes time $O(m/\varepsilon)$. The total expected running time is then $O(m/\varepsilon^2)$.

For $r \geq 2$ we make the obvious changes to SGAUSS to apply all row transformations simultaneously to r right-hand sides, which incurs no additional cost using bit parallelism. The back substitution to obtain $Z \in \{0,1\}^{(n+L-1)\times r}$ is done separately for each column, for a total construction time of $O(m/\varepsilon^2 + rm/\varepsilon)$, which establishes **(i)**.

The data structure D_f stores Z, which takes $(\frac{1}{1-\varepsilon}m + L - 1)r$ bits, as well as a seed, which takes $O(1)$ bits whp, giving at most $(1+3\varepsilon)rm$ bits whp. Replacing ε with $\varepsilon/3$ yields the literal result **(ii)**.

We store $Z \in \{0,1\}^{(n+L-1)\times r}$ by dividing it into submatrices of dimension $w \times r$ (where w is the word size) that are stored consecutively. Each submatrix is stored column-wise using r consecutive memory words. An $\mathbf{eval}_2$ operation accesses a submatrix of Z of size $L \times r$. It is easy to see that the required data is contained in at most $(L/w+1)r = O(r/\varepsilon)$ consecutive memory words and the computation in $\mathbf{eval}_2$ can be carried out using $O(r/\varepsilon)$ word operations, including bit-shift, AND and PARITY. This establishes **(iii)**. ◀

8.9. Input Partitioning – Proof of Theorem C2

We now apply input partitioning to Theorem C1, using a desired chunk size of $C = m^\varepsilon$. This reduces the time bounds for **construct** and **eval** by a factor of $1/\varepsilon$. The reason is that we can use smaller block sizes L, which makes row additions and inner products cheaper. We remark that $C = m^\varepsilon$ is unlikely to be a good choice in practice. In the argument, we make use of $\frac{\log m}{m^\varepsilon} \ll \varepsilon$ which holds for sufficiently large m. While the left term is indeed $o(1)$ and the right a constant, even for moderate values of $\varepsilon = 0.05$ implausibly large values of m are needed to satisfy $\frac{\log m}{m^\varepsilon} < \varepsilon$. In this sense, Theorem C2 taken literally is of theoretical value only. Still, the general idea is sound, and it can give improvements in practice when partitioning less aggressively, say with $C \approx \sqrt{m}$. Indeed, the running times reported in Section 12.2 are achieved with input partitioning.

Proof of Theorem C2. We use flexible input partitioning from Section 3.4.2 with $C = m^\varepsilon$ and construct an individual retrieval data structure for each chunk with $L = O(\frac{\log C}{\varepsilon}) = O(\log m)$. Arguing similarly as in Theorem C1, such a construction succeeds in expected time $O(C/\varepsilon + rC)$ with probability $1 - O(1/C)$. In case the construction fails for a chunk, it is repeated with a different seed. In the end, we save the concatenation of all m/C retrieval data structures and an auxiliary array containing for each chunk the offset of the corresponding retrieval data structure in the concatenation and the seed used for the chunk. It is easy

to check that the auxiliary array occupies $O((\log m)/m^{1-\varepsilon})$ bits which is asymptotically negligible. The total expected construction time is $O((m/C) \cdot (C/\varepsilon + rC) = O(m/\varepsilon + rm)$. Since $L = O(\log m)$, an **eval** can be carried out in time $O(r)$. It causes one cache miss, not counting the access to the auxiliary array (see Section 3.2). ◄

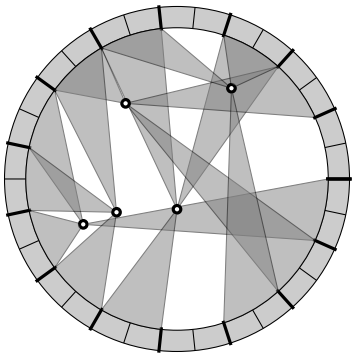
Relevant Definitions and Theorems for Chapter 9
(Originally on Pages 20, 21 and 35)

► **Definition 2.7** (Random Hypergraph with Aligned Blocks). *Let $n, m, k, \ell \in \mathbb{N}$ with n a multiple of ℓ. The random hypergraph with k* aligned blocks of size ℓ *is given as*

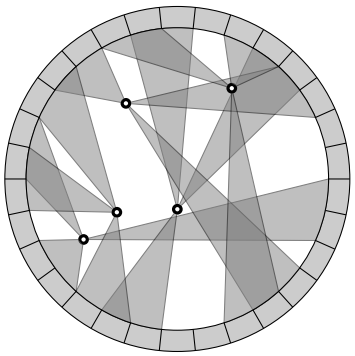
$$B^{k,\ell}_{n,m} := ([n]_0, \{e'_1, \dots, e'_m\}), \text{ where } e_i \leftarrow \binom{[n/\ell]_0}{k}$$
$$\text{and } e'_i = \bigcup_{j \in e_i} \{j\ell, \dots, (j+1)\ell - 1\} \text{ for } i \in [m].$$

► **Definition 2.8** (Random Hypergraph with Unaligned Blocks). *Let $n, m, k, \ell \in \mathbb{N}$. The random hypergraph with k* unaligned blocks of size ℓ *is given as*

$$W^{k,\ell}_{n,m} := (\mathbb{Z}_n, \{e'_1, \dots, e'_m\}), \text{ where } e_i \leftarrow \binom{\mathbb{Z}_n}{k} \text{ and } e'_i = \bigcup_{j \in e_i} \{j, \dots, j + \ell - 1\} \text{ for } i \in [m].$$



possible outcome of $B^{3,2}_{30,5}$



possible outcome of $W^{3,2}_{30,5}$

► **Theorem D.** *Let $k, \ell \geq 2$. The 1-orientability threshold $\gamma_{k,\ell}$ of $(W^{k,\ell}_{n,cn})_{c \in \mathbb{R}^+, n \in \mathbb{N}}$ is characterised by Equation (9.8) in Chapter 9. It exceeds the threshold of $(B^{k,\ell}_{n,cn})_{c \in \mathbb{R}^+, n \in \mathbb{N}}$, at least for $(k, \ell) \in \{2, \dots 7\} \times \{2, \dots, 10\}$.*

► **Theorem D1.** *Let $k, \ell \geq 2$. Consider k-ary cuckoo hashing with blocks of size ℓ. When using unaligned blocks rather than aligned blocks, the load threshold changes from $c^*_{k,\ell}/\ell$ to $\gamma_{k,\ell}$. This is an improvement at least for the values $(k, \ell) \in \{2, \dots 7\} \times \{2, \dots, 10\}$. The worst-case number of cache misses and key comparisons per lookup remain unaffected.*

9. Load Thresholds for Cuckoo Hashing with Unaligned Blocks

The purpose of this chapter is to prove Theorem D (restated on the facing page) and compute the corresponding thresholds $\gamma_{k,\ell}$. Theorem D1 is then an immediate consequence.

9.1. Outline of the Proof

Step 1: A tidier problem. The elements of a hyperedge e of $B_n = B^{k,\ell}_{n,m}$ and $W_n = W^{k,\ell}_{n,m}$ are not independent, as e is the union of k intervals of size ℓ. This poorly reflects the actual tidiness of the probabilistic object. We may obtain a model with independent elements in hyperedges, by switching to a more general notion of what it means to orient a hypergraph.

Formally, given a weighted hypergraph $H = (V, E, \eta)$ with weight function $\eta\colon V \cup E \to \mathbb{N}$, an *$\eta$-orientation* σ of H assigns to each pair (e, v) of a hyperedge and an incident vertex a number $\sigma(e,v) \in \mathbb{N}_0$ such that

$$\forall e \in E: \sum_{v \in e} \sigma(e,v) = \eta(e), \text{ and } \forall v \in V: \sum_{e \ni v} \sigma(e,v) \le \eta(v). \tag{9.1}$$

We will still say that a hyperedge e *is oriented to* a vertex v (possibly several times) if $\sigma(e,v) > 0$. One may be inclined to call $\eta(v)$ a *capacity* for $v \in V$ and $\eta(e)$ a *demand* for $e \in E$, but we use the same letter in both cases as the distinction is dropped later anyway. If all hyperedge weights are 1 and all vertex weights are ℓ, we recover the notion of an ℓ-orientation.

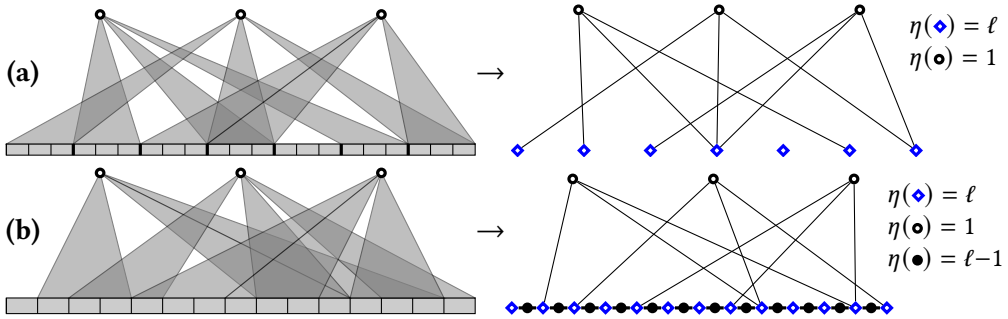


Figure 9.1 **(a)** In k-ary cuckoo hashing with aligned blocks of size ℓ (here $k = \ell = 3$), we can contract each block into a single vertex of weight ℓ to obtain a simpler but equivalent representation of the orientation problem.
(b) In k-ary cuckoo hashing with windows of size ℓ, a similar idea works, but additional helper hyperedges (drawn as ●) of weight $\ell - 1$ are needed (see Proposition 9.1).

A simplified representation of $B^{k,\ell}_{n,m}$ is readily obtained as $H^k_{n/\ell,m}$, as explained in Section 2.4.1 and illustrated in Figures 2.5 and 9.1 **(a)**. In $H^k_{n/\ell,m}$, each group of ℓ vertices of

$B^{k,\ell}_{n,m}$ representing one aligned block is now contracted into a single vertex of weight ℓ and hyperedges contain k independent vertices representing blocks instead of $k\ell$ dependent vertices. It is clear that $B^{k,\ell}_{n,m}$ is 1-orientable if and only if $H^k_{n/\ell,m}$ is ℓ-orientable.

In a similar spirit we identify a transformed version $\hat{W}_n$ for W_n. However, this time the details are more complicated as the vertex set has an intrinsic linear geometry, whereas B_n featured essentially an unordered collection of internally unordered blocks. The *ordinary hyperedges* in $\hat{W}_n$ also have size k instead of size $k\ell$, but we need to introduce additional *helper hyperedges* that capture the linear geometry of $\mathbb{Z}_n$, see Figure 9.1 **(b)**. We define:

$$\begin{aligned}&\hat{W}_n \coloneqq \hat{W}^{k,\ell}_{n,m} \coloneqq (\mathbb{Z}_n, C_n \cup \{e_1, \dots, e_m\}, \eta) \\ &\text{with ordinary hyperedges } e_i \leftarrow \binom{\mathbb{Z}_n}{k}, \\ &\text{helper hyperedges } C_n = \{h_i \coloneqq (i, i+1) \mid i \in \mathbb{Z}_n\}, \\ &\text{vertex weight } \eta(v) = \ell \text{ for } v \in \mathbb{Z}_n, \\ &\text{and hyperedge weights } \eta(h) = \ell - 1,\ \eta(e) = 1 \text{ for } h \in C_n, e \in \{e_1, \dots, e_m\}.\end{aligned} \tag{9.2}$$

Note that formally the hypergraphs W_n and $\hat{W}_n$ are random variables on a common probability space. An outcome $\omega = (e_i)_{i\in[m]}$ from this space determines both hypergraphs.

The following proposition justifies the definition. It is proved in Section 9.2.

▶ **Proposition 9.1.** *$\hat{W}_n$ is orientable if and only if W_n is orientable.*

An important merit of $\hat{W}_n$ that will be useful in Step 3 is that it is *locally tree-like*, meaning each vertex has a probability of $o(1)$ to be involved in a constant-length cycle. Here, by a cycle in a hypergraph we mean a sequence of distinct hyperedges $e_1, e_2, \dots, e_j$ such that two consecutive hyperedges share a vertex and e_j and e_1 share a vertex.

Note the interesting special case $\hat{W}^{2,2}_{n,m}$, which is a cycle of length n with m random chords, unit edge weights and vertices of weight 2. Understanding the orientability thresholds for this graph seems interesting in its own right, not just as a means to understand $W^{2,2}_{n,m}$.

Step 2: Incidence Graph and Allocations. The next step is by no means difficult or creative. We merely perform the necessary preparations needed to apply Lelarge's Theorem [LLM13], recalling their concept of an allocation in the process (see also Chapter 5).

This will effectively get rid of the asymmetry between the roles of vertices and hyperedges in the problem of orienting $\hat{W}_n$, by switching perspective in two simple ways. The first is to consider the incidence graph G_n of $\hat{W}_n$ instead of $\hat{W}_n$ itself, i.e. the weighted bipartite graph

$$G_n = G^{k,\ell}_{n,m} = (\underbrace{C_n}_{A_C} \cup \underbrace{\{e_1, \dots, e_m\}}_{A_R}, \underbrace{\mathbb{Z}_n}_{B}, \underbrace{\text{“}\ni\text{”}}_{E(G_n)}, \ \eta\ ). \tag{9.3}$$

We use $A = A_C \cup A_R$ to denote those vertices of G_n that were hyperedges in $\hat{W}_n$, and B for those vertices of G_n that were vertices in $\hat{W}_n$. Vertices $a \in A$ and $b \in B$ are adjacent in G_n if $b \in a$ in $\hat{W}_n$. The weights η on vertices and hyperedges in $\hat{W}_n$ are now vertex weights with $\eta(a_C) = \ell - 1$, $\eta(a_R) = 1$, $\eta(b) = \ell$ for $a_C \in A_C$, $a_R \in A_R$, $b \in B$. The notion of σ being an η-orientation translates to σ being a map $\sigma : E(G_n) \to \mathbb{N}_0$ such that $\sum_{b\in N(a)} \sigma(a,b) = \eta(a)$ for all $a \in A$ and $\sum_{a\in N(b)} \sigma(a,b) \le \eta(b)$ for all $b \in B$. Note that

vertices from A need to be *saturated* ("$= \eta(a)$" for $a \in A$) while vertices from B need not be ("$\leq \eta(b)$" for $b \in B$). This leads to the second switch in perspective.

Dropping the saturation requirement for A, we say σ is an *allocation* (with respect to η) if $\sum_{u \in N(v)} \sigma(u,v) \leq \eta(v)$ for all $v \in A \cup B$.

Clearly, any η-orientation is an allocation, but not vice versa; for instance, the trivial map $\sigma \equiv 0$ is an allocation. Let $|\sigma|$ denote the size of an allocation, i.e. $|\sigma| = \sum_{e \in E} \sigma(e)$. By bipartiteness, no allocation can have a size larger than the total weight of A, i.e.

$$\text{for all allocations } \sigma: \ |\sigma| \leq \eta(A) = \sum_{a \in A} \eta(a) = |A_C| \cdot (\ell - 1) + |A_R| \cdot 1 = (\ell - 1)n + m$$

and the η-orientations of G_n are precisely the allocations of size $\eta(A)$. From now on let $m = cn$. We conclude:

▶ **Proposition 9.2.** *Let $M(G_n)$ denote the maximal size of an allocation of G_n. Then*

$$\frac{M(G_n)}{n} = \ell - 1 + c \quad \textit{if and only if} \quad \hat{W}_n \textit{ is orientable.}$$

Step 3: The Limit T of G_n. Reaping the benefits of step 1, we find that $G_n = G^{k,\ell}_{n,cn}$ has $O(1)$ cycles of length $O(1)$ whp. To capture the local appearance of G_n even more precisely, let the *r-ball* around a vertex v in a graph be the subgraph induced by the vertices of distance at most r from v. Then the r-ball around a random vertex of G_n is distributed, as n gets large, more and more like the r-ball around the root of a random infinite rooted tree $T = T^{k,\ell}_c$. The tree T is distributed as follows, with weighted nodes of types A_C, A_R or B.

- The root of T is of type A_C, A_R or B with probability $\frac{1}{2+c}$, $\frac{c}{2+c}$ and $\frac{1}{2+c}$, respectively.
- If the root is of type A_C, it has two children of type B. If it is of type A_R, it has k children of type B. If it is of type B, it has two children of type A_C and a random number X of children of type A_R, where $X \sim \mathrm{Po}(kc)$. Here $\mathrm{Po}(\lambda)$ denotes the Poisson distribution with parameter λ.
- A vertex of type A_C that is not the root has one child of type B. A vertex of type A_R that is not the root has $k-1$ children of type B.
- A vertex of type B that is not the root has a random number X of children of type A_R, where $X \sim \mathrm{Po}(kc)$. If its parent is of type A_C, then it has one child of type A_C. Otherwise, it has two children of type A_C.
- Vertices of type A_C, A_R and B have weight $\ell-1$, 1 and ℓ, respectively.

All random decisions should be understood to be independent. A type is also treated as a set containing all vertices of that type. In Section 9.3 we argue that:

▶ **Proposition 9.3.** *Almost surely, $(G_n)_{n \in \mathbb{N}} = (G^{k,\ell}_{n,cn})_{n \in \mathbb{N}}$ has local weak limit $T = T^{k,\ell}_c$.*

The concept of local weak limit is explained in Section 5.1. With notation from Section 5.2 and distributions given in Figure 9.2, T is a bipartite Galton-Watson tree $T \overset{\mathrm{d}}{=} \mathrm{GWT}_{\Phi^A, \Phi^B}$.

Step 4: The Method of Lelarge. We are now in a position to apply Lelarge's Theorem [LLM13] that characterises $\lim_{n \to \infty} \frac{M(G_n)}{n}$ in terms of solutions to belief propagation equations for T. Put abstractly: The numerical limit of a function of G_n is expressed as a function of the graph limit of G_n. We elaborate on details and deal with the equations in Section 9.4. After condensing the results into a characterisation of $\gamma_{k,\ell} \in (0,1)$ in terms of "well-behaved" functions we obtain:

Φ^A:

Sketch	Formal Outcome $(D^A, W^A, \{C_i^A\}_{i\in[D^A]})$	Probability $\Phi^A(D^A, W^A, \{C_i^A\}_{i\in[D^A]})$
	$(k, 1, \{R, R, \dots, R\})$	$\frac{c}{1+c}$
	$(2, \ell-1, \{C, C\})$	$\frac{1}{1+c}$

Φ^B:

Sketch	Formal Outcome $(D^B, W^B, \{C_i^B\}_{i\in[D^B]})$	Probability $\Phi^B(D^B, W^B, \{C_i^B\}_{i\in[D^B]})$
	$(2, \ell, \{C, C\})$	$\Pr[\mathrm{Po}(ck) = 0]$
	$(3, \ell, \{C, C, R\})$	$\Pr[\mathrm{Po}(ck) = 1]$
⋮	⋮	⋮
	$(d, \ell, \{C, C, R, \dots, R\})$	$\Pr[\mathrm{Po}(ck) = d-2]$
⋮	⋮	⋮

Figure 9.2 Distributions $\Phi^A = \Phi^A_{c,k,\ell}$ and $\Phi^B = \Phi^B_{c,k,\ell}$ on stars (in the sense of Section 5.2) such that $\mathrm{GWT}_{\Phi^A,\Phi^B} \stackrel{d}{=} T$. This slightly more technical terminology is closer to Lelarge's Theorem [LLM13] and we use distinct constants $R, C \geq \ell$ as edge weights to distinguish edges incident to A_R and A_C.

▶ **Proposition 9.4.**

$$\lim_{n\to\infty} \frac{M(G_{n,cn})}{n} \begin{cases} = \ell - 1 + c & \text{almost surely} \quad \text{if } c < \gamma_{k,\ell} \\ < \ell - 1 + c & \text{almost surely} \quad \text{if } c > \gamma_{k,\ell}. \end{cases}$$

Step 5: Closing the Gap. It is important to note that we are not done, as

$$\lim_{n\to\infty} \frac{M(G_{n,cn})}{n} = \ell - 1 + c \text{ a.s.} \quad \text{does not imply} \quad M(G_{n,cn}) = n \cdot (\ell - 1 + c) \text{ whp.} \tag{9.4}$$

We still have to exclude the possibility of a gap of size $o(n)$ on the right hand side; imagine for instance $M(G_{n,cn}) = (\ell - 1 + c)n - \sqrt{n}$ to appreciate the difference. In the setting of cuckoo hashing with double hashing in Chapter 10, the analogue of this pesky distinction requires a somewhat lengthy case analysis. We should therefore treat this carefully.

Luckily the line of reasoning by Lelarge [Lel12] can be adapted to our more general setting. The key is to prove that if no orientation exists, then the configuration causing this problem has size $\Theta(n)$ and those large overfull structures do not go unnoticed on the left side of (9.4).

▶ **Lemma 9.5.** *There is a constant $\delta > 0$ such that whp no set of $0 < t < \delta n$ vertices in $\hat{W}_n$ (of weight ℓt) induces hyperedges of total weight ℓt or more, provided $c \leq 1$.*

The proof of this Lemma (using first moment methods) and the final steps towards Theorem D are found in Section 9.5.

The following sections provide the details of the technical argument.

9.2. Equivalence of W_n and $\hat{W}_n$ with respect to orientability

In this section we prove Proposition 9.1, i.e. we show that W_n is orientable if and only if $\hat{W}_n$ is orientable. Recall the relevant notions from Definition 2.8 and Equations (9.1) and (9.2). Some ideas are illustrated in Figure 9.3. We start with the hypergraph W_n in **(a)**. From this, **(b)** is obtained by introducing one "broker-vertex" (◈) for each interval of size ℓ in the table, through which the incidences of the hyperedges (⊙) are "routed" as shown. The purpose of each broker-vertex is to "claim" part of its interval on behalf of incident hyperedges. To manage these claims, we imagine a "separator" (●) between each pair of adjacent broker-vertices that, by pointing between two table cells, indicates where the claim of one broker-vertex ends and the claim of the next broker-vertex begins, see **(c)**. There are ℓ possible "settings" for each separator. The separators can be modelled as (hyper-)edges of size 2 and weight $\ell - 1$ with ℓ possible ways to distribute this weight among the two incident broker-vertices that have weight ℓ. The table is then fully implicit, which gives $\hat{W}_n$ in **(d)**.

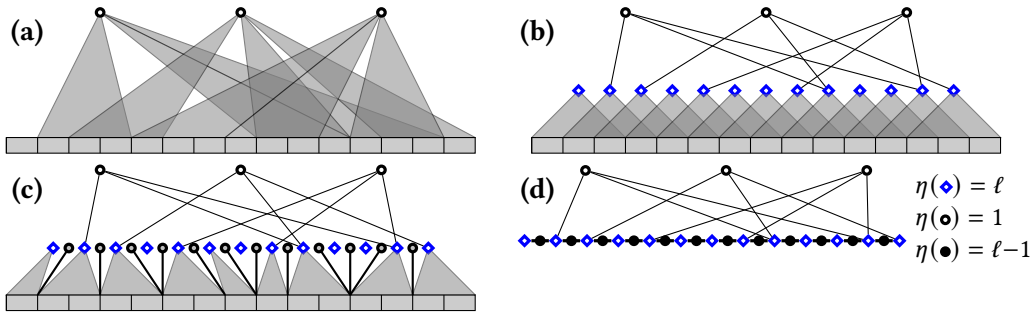


Figure 9.3 Drawing of (a part of) W_n in **(a)** and $\hat{W}_n$ in **(d)** with two intermediate ideas implicit in the proof of Proposition 9.1.

Proof of Proposition 9.1. For this proof, let $\mathfrak{w}_i := \{i, \dots, i + \ell - 1\}$.

⇒ Let σ be a 1-orientation of W_n. We will define an η-orientation $\hat{\sigma}$ of $\hat{W}_n$. Recall from Definition 2.8 and Equation (9.2) how a hyperedge $e' = \bigcup_{j \in e} \mathfrak{w}_j$ of W_n is defined in terms of a hyperedge e of $\hat{W}_n$. If σ assigns e' to $x \in \mathbb{Z}_n$, we pick $j \in e$ with $x \in \mathfrak{w}_j$. We let $\hat{\sigma}$ assign e to j and also assign to x the *label* $\mathfrak{w}_j$.
Note that, since σ is a 1-orientation, each $x \in \mathbb{Z}_n$ receives at most one label this way, and that the label stems from $\{\mathfrak{w}_{x-\ell+1}, \dots, \mathfrak{w}_x\}$.
We still have to orient the helper hyperedges $h_i = (i, i+1)$ of weight $\ell{-}1$. For this, we count the number r_i of elements in $\mathfrak{w}_i \cap \mathfrak{w}_{i+1}$ with a label that is *to the right*, i.e. stems from $\{\mathfrak{w}_{i+1}, \mathfrak{w}_{i+2}, \dots, \mathfrak{w}_{i+\ell-1}\}$. We then set $\hat{\sigma}(h_i, i) = r_i$ and $\hat{\sigma}(h_i, i+1) = \ell{-}1 - r_i$.
We now check that the weight of any vertex $i \in \mathbb{Z}_n$ is respected, i.e. we check that

$$\hat{\sigma}(h_i, i) + \hat{\sigma}(h_{i-1}, i) + \sum_{\substack{\text{ordinary hyperedge } e \\ i \in e}} \hat{\sigma}(e, i) \overset{!}{\leq} \ell.$$

From ordinary hyperedges, the contribution is 1 for each $x \in \mathfrak{w}_i$ with label $\mathfrak{w}_i$. From h_i the contribution is the number of $x \in \mathfrak{w}_i \cap \mathfrak{w}_{i+1}$ with label in $\{\mathfrak{w}_{i+1}, \dots, \mathfrak{w}_{i+\ell-1}\}$ and from h_{i-1}, the contribution is the number of $x \in \mathfrak{w}_{i-1} \cap \mathfrak{w}_i$ *not* having a label from

$\{\mathfrak{w}_i, \dots, \mathfrak{w}_{i+\ell-2}\}$. The three conditions are clearly mutually exclusive, so each $x \in \mathfrak{w}_i$ can contribute at most 1, giving a total contribution of at most $|\mathfrak{w}_i| = \ell$ as required.

⇐ Let $\hat{\sigma}$ be an η-orientation of $\hat{W}_n$. Define $s_i := \hat{\sigma}(h_{i-1}, i)$ and $t_i := \hat{\sigma}(h_i, i)$. Let further $\bar{\mathfrak{w}}_i := \{i + s_i, \dots, i + \ell - t_i - 1\} \subseteq \mathfrak{w}_i$.
Crucially, $\{\bar{\mathfrak{w}}_i \mid i \in \mathbb{Z}_n\}$ forms a partition of $\mathbb{Z}_n$. This follows from the following properties:

$$\max(\bar{\mathfrak{w}}_i) = i + \ell - t_i - 1, \quad \min(\bar{\mathfrak{w}}_{i+1}) = i + 1 + s_{i+1}$$
$$\text{and} \quad t_i + s_{i+1} = \hat{\sigma}(h_i, i) + \hat{\sigma}(h_i, i+1) = \eta(h_i) = \ell - 1.$$

Here, if cyclic intervals span the "seam" of the cycle, max and min should be reinterpreted in the natural way. Now let $e_1^{(i)}, \dots, e_{\rho_i}^{(i)}$ be the ordinary hyperedges directed to i by $\hat{\sigma}$. Since $\hat{\sigma}$ respects $\eta(i) = \ell$ we have $\rho_i + s_i + t_i \le \ell$, so $\rho_i \le \ell - s_i - t_i = |\bar{\mathfrak{w}}_i|$. We can now define the 1-orientation σ of W_n to direct each $e_j^{(i)}$ to $i + s_i + j - 1 \in \mathbb{Z}_n$ for $j \in [\rho_i]$ and $i \in \mathbb{Z}_n$. ◂

9.3. Local weak convergence of G_n to T

Recall the definitions of the finite graph G_n in Equation (9.3) and the infinite rooted tree T in Step 3 of Section 9.1. As in Chapter 5, we obtain the rooted graph $G_n(\circ)$ from G_n by distinguishing one vertex—the root—uniformly at random. For any rooted graph R and $d \in \mathbb{N}$, let $(R)_d$ denote the rooted subgraph of R induced by the vertices at distance at most d from the root. We treat two rooted graphs as equal if there is an isomorphism between them that preserves root and vertex types. Refer to Figure 9.4 for a possible outcome of $(T)_3$ and $(G_n(\circ))_3$.

As in Theorem 5.2, the local weak convergence claim of Proposition 9.3 can be simplified using the Portmanteau Theorem. What was stated separately as **(i)** and **(ii)** in Theorem 5.2 can also be stated together as:

$$\forall d \in \mathbb{N} : \ \forall \text{ rooted graph } H : \ \lim_{n\to\infty} \Pr_{\circ}[(G_n(\circ))_d = H] = \Pr_{T}[(T)_d = H] \text{ almost surely}_{(G_n)_{n\in\mathbb{N}}}. \tag{9.5}$$

Here "almost surely" refers to the randomness in choosing the sequence $(G_n)_{n\in\mathbb{N}}$ while $\Pr_\circ$ refers to the randomness in choosing $\circ \leftarrow V(G_n)$.

The concentration aspect of (9.5) is proved in the same way as in Theorem 5.2. We will take the time to sketch a proof of the other ingredient, namely the weak convergence of the "average" measures, which was **(i)** in Theorem 5.2:

$$\forall d \in \mathbb{N} : \ \forall \text{ rooted graph } H : \ \lim_{n\to\infty} \Pr_{G_n,\circ}[(G_n(\circ))_d = H] = \Pr_{T}[(T)_d = H]. \tag{9.5'}$$

Note that in (9.5) the probability in the limit is a random variable (depending on $(G_n)_{n\in\mathbb{N}}$), while the probability in (9.5') is not.

Proof of Equation (9.5'). For simplicity, we shall assume that child vertices of the same type are ordered (for $G_n(\circ)$ and T just fix an ordering at random). Correspondingly, we prove the more fine-grained version of (9.5') where equality indicates the existence of an isomorphism preserving roots, types and child orderings.

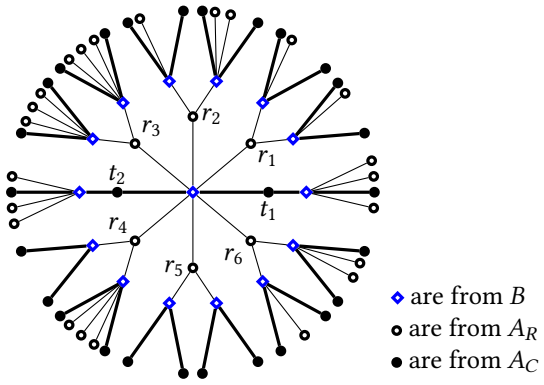


Figure 9.4 One possibility of what $(T)_3$ may look like for $k = 3$. Since the infinite random tree T is designed to reflect the local characteristics of G_n, it is also a possibility for $(G_n(\circ))_3$. Actually, the distributions of $(G_n(\circ))_3$ and $(T)_3$ are asymptotically equal.

Let H be a possible outcome of $(T)_d$, let $t \in \{A_C, A_R, B\}$ be the type of the root of H and let $v_1, \dots, v_s$ be the vertices of type B in H, except for leaves, in breadth-first-search ordering. Let further $x_i \in \mathbb{N}_0$ denote the number of v_i's children of type A_R for $i \in [s]$. Checking the definition of T, the sequence $(t, x_1, \dots, x_s)$ contains all random decisions that T has to "get right" in order for $(T)_d$ to coincide with H, i.e.

$$\Pr[(T)_d = H] = \Pr[\text{root}(T) \in t] \cdot \prod_{i=1}^{s} \Pr[X_i = x_i]$$

where $X_i \sim \text{Po}(ck)$ for $i \in \mathbb{N}$ are the independent random variables used in the construction of T, in breadth-first-search order.

To compare this to $\Pr_{G_n, \circ \in G_n}[(G_n(\circ))_d = H]$, we shall reveal the type of $\circ$ and then the neighbourhoods of vertices from $(G_n(\circ))_d$ one by one in breadth-first-search order and check equality with H. Let $w_1, w_2, \dots$ be the vertices of type B in breadth-first-search ordering and $C(w_i) := N(w_i) \setminus \text{parent}(w_i)$ the successors w_i, i.e. w_i's neighbours except for the vertex from which w_i was discovered (if $w_i \neq \circ$). Recall that the cn vertices from A_R each choose k among the n vertices of B uniformly at random, so for $v \in B$ we have $|N(v) \cap A_R| \sim \text{Bin}(ckn, \frac{1}{n})$. However, when revealing $Y_i := |C(w_i) \cap A_R|$, then a constant number α_i of incidences of vertices from A_R are already revealed and the full neighbourhoods of a constant number β_i of vertices of type B are already revealed. Thus, conditioned on $G_n(\circ)$ matching with H until before $N(w_i)$ is revealed, we have $Y_i \sim \text{Bin}(ckn - \alpha_i, \frac{1}{n-\beta_i})$. A second complication is that $G_n(\circ)$ can contain cycles. Therefore, whenever we reveal the identity of a vertex of type A_R we shall assume that it is not one of the vertices already seen and when we reveal the identity of a vertex of type B found as a child of a vertex of type A_R, we shall assume that its position in $\mathbb{Z}_n$ is not within distance d of a vertex of type B already seen. The probability of these events is clearly $1 - o(1)$ since only a constant number of vertices are forbidden. It is important to note that, since H is a possible outcome for $(T)_d$, all remaining aspects of H and $G_n(\circ)$ coincide by construction (e.g. the degree of vertices of types A_R and A_C). We get

$$\Pr_{G_n, \circ \in V(G_n)}[(G_n(\circ))_d = H] = (1 - o(1)) \Pr[\circ \in t] \cdot \prod_{i=1}^{s} \Pr[Y_i = x_i \mid G_n(\circ) \text{ matches with } H \text{ up to } v_i].$$

Since $\Pr[\circ \in t] = \Pr[\mathrm{root}(T) \in t]$ by construction and due to the convergence of Binomial to Poisson random variables, we get $\lim_{n\to\infty} \Pr_{G_n,\circ}[(G_n(\circ))_d = H] = \Pr[(T)_d = H]$ as desired. ◀

9.4. Belief Propagation on the Limiting Tree T

Before applying Lelarge's Theorem, we try to give some intuition on the underlying belief propagation equations.

Recall the definition and relevance of large allocations from Proposition 9.2 and consider the task of finding a large allocation σ for G_n. Imagine the vertices as agents in a parallel endeavour that proceeds in rounds and that is designed to yield information useful for constructing σ. In each round, every vertex sends a message to each of its neighbours. Since two messages are sent between two adjacent vertices u and v—one in each direction—it is convenient to distinguish the directed edges (u,v) and (v,u), the message from u to v being sent along (u,v) and vice versa. Along $e = (u,v)$ the message is a number $I_e \in [0, \min(\eta(u), \eta(v))]$. We interpret this as the vertex u suggesting that $\sigma(\{u,v\})$ be set to I_e. To determine I_e, the vertex u sums up the messages it received from its *other* neighbours in the previous round, obtaining a value ξ. If $\xi < \eta(u)$, then, assuming the suggestions of the neighbours of u were all followed, u would want $\sigma(e) = \eta(u) - \xi$ in order to fully utilise its weight $\eta(u)$. Taking into account the weight of v, u sends $I_e = [\eta(u) - \xi]_0^{\eta(v)}$ where $[x]_i^j := \max(i, \min(j, x))$ is our shorthand for the "clamp function" (which we also occasionally use for one-sided clamping, leaving out the upper or lower index). Let P be the operator that takes an assignment $I : \vec{E}(G_n) \to \mathbb{N}_0$ of messages to directed edges and computes the messages $P(I) : \vec{E}(G_n) \to \mathbb{N}_0$ of the next round.

On finite trees, iterated application of P can easily be seen to converge to a unique fixed point I^* of P, regardless of the initial assignment of messages. From I^*, the size of the maximum allocation can be obtained by local computations. It is plausible but non-trivial that the asymptotic behaviour of a largest allocation of G_n is similarly linked to fixed points of P on the random weak limit T of G_n.

Let (u,v) be an edge of T where v is closer to the root than u and let T_u be the subtree of T containing v, u and all descendants of u. If we apply P to T repeatedly (starting with, say, the all-zero message assignment $I \equiv 0$), the message $I_{(u,v)}$ in later rounds will depend on ever larger parts of T_u but nothing else. Assume there was a magical (measurable) function f that finds, by looking at all of T_u, the message $f(T_u)$ that is sent along (u,v) in some fixed point of P. In particular, if $u_1, \ldots, u_i$ are the children of u, locally, the fixed point equation is

$$f(T_u) = [\eta(u) - \sum_{j=1}^{i} f(T_{u_j})]_0^{\eta(v)}.$$

Assume we have yet to reveal anything about T_u and only know the types t_u and t_v of u and v. Then the random variable $I_{t_u \to t_v} := f(T_u)$ has a well-defined distribution. The four possible combinations of types yield random variables $I_{A_C \to B}, I_{A_R \to B}, I_{B \to A_C}, I_{B \to A_R}$, which must fulfil certain distributional equations.

Consider for instance $e = (u,v)$ with $u \in B$ and $v \in A_C$. On the one hand the message $f(T_u)$ is distributed like $I_{B \to A_C}$. On the other hand, looking one layer deeper, u has children $u_1, \ldots, u_X$ of type A_R, with $X \sim \mathrm{Po}(ck)$ as well as one child $a' \neq a$ of type A_C. The messages

$f(T_{u_1}), \dots, f(T_{u_X})$ and $f(T_{a'})$ are independent (since the subtrees are independent) and distributed like $I_{A_R \to B}$ or $I_{A_C \to B}$, respectively, implying:

$$I_{B \to A_C} \stackrel{\mathrm{d}}{=} \big[\ell - I_{A_C \to B} - \sum_{j=1}^{X} I^{(j)}_{A_R \to B}\big]_0^{\ell-1}. \tag{⋆3}$$

where a superscript in parentheses indicates an independent copy of a random variable and "$\stackrel{\mathrm{d}}{=}$" denotes equality in distribution.

Leconte, Lelarge, and Massoulié [LLM13] show that, remarkably, the solutions to a system of such equations are essentially all we require to capture the asymptotics of maximum allocations. Readers deterred by the measure theory may find Section 2 of [Lel12] illuminating, which gives a high-level description of the argument for a simpler case.

We now apply Lelarge's Theorem to our family $(G_n)_{n\in\mathbb{N}}$, which is a fairly straightforward matter with one twist: In [LLM13] allocations were restricted not only by vertex constraints (our $\eta : V(G_n) \to \mathbb{N}$) but also by edge constraints giving an upper bound on $\sigma(e)$ for every edge e. We do not require them in this sense and make all edge constraints large enough to never get in the way. We repurpose them *for something else*, however, namely to tell apart the subtypes A_C and A_R within the vertex set A. This is because the distribution of the children of $u \in A$ depends on this distinction and while [LLM13] knows no subtypes of A out of the box, the constraint on the edge to the parent may influence the child distribution.

Concretely, we use Lelarge's Theorem in the specialised form of Equation (5.2), the distributions Φ^A and Φ^B from Figure 9.2 and the substitutions

$$X(C) = I_{A_C \to B}, \quad X(R) = I_{A_R \to B}, \quad Y(C) = I_{B \to A_C}, \quad Y(R) = I_{B \to A_R}.$$

► **Lemma 9.6** (Special case of [LLM13, Theorem 2.1])**.**

$$\lim_{n\to\infty} \frac{M(G_{n,cn})}{n} = \inf\left\{F(I_{A_C \to B}, I_{B \to A_C}, I_{A_R \to B}, I_{B \to A_R}, c)\right\} \quad \textit{almost surely, where}$$

$$F(I_{A_C \to B}, I_{B \to A_C}, I_{A_R \to B}, I_{B \to A_R}, c) = \mathbb{E}\Bigg[[I^{(1)}_{B \to A_C} + I^{(2)}_{B \to A_C}]_0^{\ell-1} + c \cdot [\sum_{j=1}^{k} I^{(j)}_{B \to A_R}]_0^1$$

$$+\Big[\ell - \sum_{i=1,2} [\ell - I^{(i)}_{A_C \to B} - \sum_{j=1}^{X} I^{(j)}_{A_R \to B}]_0 - \sum_{i=1}^{X} [\ell - I^{(1)}_{A_C \to B} - I^{(2)}_{A_C \to B} - \sum_{j\neq i} I^{(j)}_{A_R \to B}]_0\Big]_0\Bigg]$$

and the infimum is taken over distributions of $I_{A_C \to B}, I_{A_R \to B}, I_{B \to A_C}, I_{B \to A_R}$ *fulfilling*

$$I_{A_C \to B} \stackrel{d}{=} \ell - 1 - I_{B \to A_C}, \tag{⋆1}$$

$$I_{A_R \to B} \stackrel{d}{=} \big[1 - \sum_{j=1}^{k-1} I^{(j)}_{B \to A_R}\big]_0, \tag{⋆2}$$

$$I_{B \to A_C} \stackrel{d}{=} \big[\ell - I_{A_C \to B} - \sum_{j=1}^{X} I^{(j)}_{A_R \to B}\big]_0^{\ell-1}, \tag{⋆3}$$

$$I_{B \to A_R} \stackrel{d}{=} \big[\ell - I^{(1)}_{A_C \to B} - I^{(2)}_{A_C \to B} - \sum_{j=1}^{X} I^{(j)}_{A_R \to B}\big]_0^1, \tag{⋆4}$$

where $X \sim \mathrm{Po}(kc)$ *and superscripts in parentheses indicate independent copies.*

To appreciate the usefulness of Lemma 9.6, understanding its form is more important than understanding the significance of the individual terms.

If X is a random variable on a finite set D, then the distribution of X is captured by real numbers $(\Pr[X = i])_{i\in D} \in [0,1]^{|D|}$ that sum to 1. In this sense, the four distributions of $I_{A_C\to B}, I_{A_R\to B}, I_{B\to A_C}, I_{B\to A_R}$ are given by numbers $\vec{\rho} \in [0,1]^{\ell-1}\times[0,1]^2\times[0,1]^{\ell-1}\times[0,1]^2 = [0,1]^{2\ell+2}$. We say $\vec{\rho} \in [0,1]^{2\ell+2}$ is a *solution* to the system $(\star)$ if the four groups of numbers belonging to the same distribution each sum to 1 and if setting up the four random variables according to $\vec{\rho}$ satisfies $(\star1)$,$(\star2)$,$(\star3)$ and $(\star4)$.

If we treat c as a variable instead of as a constant, we obtain the *relaxed system* $(\star c)$ where solutions are pairs $(\vec{\rho}, c) \in [0,1]^{2\ell+2} \times (0,\infty)$. The value $\vec{\rho}_{\text{triv}}$ that corresponds to

$$1 = \Pr[I_{A_C\to B} = 0] = \Pr[I_{A_R\to B} = 0] = \Pr[I_{B\to A_C} = \ell - 1] = \Pr[I_{B\to A_R} = 1]$$

is easily checked to give rise to a solution $(\vec{\rho}_{\text{triv}}, c)$ of the relaxed system for any $c > 0$, we call such a solution *trivial*. Evaluating F for a trivial solution yields $\ell - 1 + c$ so Lemma 9.6 implies the trivial assertion $\lim \frac{M(G_n)}{n} \le \ell - 1 + c$ for all $c > 0$.

We now give a "nice" characterisation of the space of non-trivial solutions for $(\star c)$.

▶ **Lemma 9.7.** *For any $k, \ell \ge 2$, there is a bijective map $\lambda \mapsto (\vec{\rho}_\lambda, c_\lambda)$ from $(0,\infty)$ to the set of non-trivial solutions for $(\star c)$.*

Moreover, (each component of) this map is an explicit real analytic function.

Proof. Note that $Y \coloneqq \sum_{j=1}^{X} I^{(j)}_{A_R\to B}$ is the sum of X independent indicator random variables, where $X \sim \text{Po}(kc)$ and $\Pr[I^{(j)}_{A_R\to B} = 1] = q$ for some $q \in [0,1]$ and all $1 \le j \le X$, with $q = 0$ only occurring in trivial solutions. It is well known that such a "thinned out" Poisson distribution is again Poisson distributed and we have $Y \sim \text{Po}(\lambda)$ for $\lambda = kcq$. Thus, each non-trivial solution to $(\star c)$ has such a parameter $\lambda > 0$. We will now show that, conversely, λ uniquely determines this solution. From $(\star1)$ and $(\star3)$ we obtain:

$$I_{B\to A_C} \overset{\text{d}}{=} \left[\ell - (\ell - 1 - I_{B\to A_C}) - Y\right]_0^{\ell-1} = \left[I_{B\to A_C} + 1 - Y\right]_0^{\ell-1}.$$

With $p_i \coloneqq \Pr[I_{B\to A_C} = i]$ for $0 \le i \le \ell - 1$ we can write this equation in matrix form as

$$\begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_{\ell-1} \end{pmatrix} = e^{-\lambda} \cdot \begin{pmatrix} * & * & * & \dots & * \\ 1 & \lambda & \lambda^2/2 & \dots & \frac{\lambda^{\ell-1}}{(\ell-1)!} \\ & \ddots & \ddots & & \vdots \\ & & 1 & \lambda & \lambda^2/2 \\ & & & 1 & \lambda+1 \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_{\ell-1} \end{pmatrix} \tag{9.6}$$

This uses $\Pr[Y = j] = e^{-\lambda}\lambda^j/j!$ for $j \in \mathbb{N}_0$. Each "$*$" is such that the columns of the matrix sum to $(e^\lambda, \dots, e^\lambda)$, which is implicit in the fact that we deal with distributions. The unique solution for a fixed λ can be obtained by using the equations from bottom to top to express $p_{\ell-2}, p_{\ell-3}, \dots, p_0$ in terms of $p_{\ell-1}$ and then choosing $p_{\ell-1}$ such that the probabilities sum to 1. This yields a closed expression $p_j = p_j(\lambda)$ for $0 \le i \le \ell - 1$.

Using first $(\star1)$, then $(\star4)$ (with the definition of Y) and finally $(\star2)$ the distributions of $I_{A_C\to B}$, $I_{B\to A_R}$ and $I_{A_R\to B}$ fall into place, completing the unique solution candidate $\vec{\rho}_\lambda$. The only loose end is the definition of λ, which gives a final equation: If $q(\lambda) = \Pr[I_{A_R\to B} = 1]$ is

the value we computed after choosing λ, we need $\lambda = kc \cdot q(\lambda)$, which uniquely determines a value $c_\lambda = \frac{\lambda}{k \cdot q(\lambda)}$ (it is easy to check that $\lambda > 0$ guarantees $q(\lambda) > 0$). Thus $(\vec{\rho}_\lambda, c_\lambda)$ is the unique solution with parameter λ. Retracing our steps it is easy to verify that we only composed real analytic functions. ◀

With the parametrisation of the solutions of $(\star c)$, we can, with a slight stretch of notation, rewrite Lemma 9.6. For any $c > 0$ we have

$$\lim_{n\to\infty} \frac{M(G_{n,cn})}{n} = \inf(\{F(\vec{\rho}_\lambda, c_\lambda) \mid \lambda \in (0,\infty), c_\lambda = c\} \cup \{\ell - 1 + c\}) \quad \text{almost surely.} \tag{9.7}$$

We now define the value $\gamma_{k,\ell}$ and by proving Proposition 9.4 demonstrate its significance.

$$\gamma_{k,\ell} \coloneqq \inf_{\lambda>0}\{c_\lambda \mid F(\vec{\rho}_\lambda, c_\lambda) < \ell - 1 + c_\lambda\}. \tag{9.8}$$

Proof of Proposition 9.4. Case $c < \gamma_{k,\ell}$. By definition of $\gamma_{k,\ell}$ there is no parameter λ with $c_\lambda = c$ and $F(\vec{\rho}_\lambda, c_\lambda) < \ell - 1 + c$. Thus, Equation (9.7) implies $\lim_{n\to\infty} \frac{1}{n} M(G_{n,cn}) = \ell - 1 + c$ almost surely.

Case $c > \gamma_{k,\ell}$. By definition of $\gamma_{k,\ell}$, for $c = \gamma_{k,\ell} + \varepsilon$ there is some λ with $c_\lambda \in [\gamma_{k,\ell}, c)$ and $F(\vec{\rho}_\lambda, c_\lambda) \leq \ell - 1 + c_\lambda - \varepsilon'$ for some $\varepsilon' > 0$. This implies $\lim_{n\to\infty} \frac{1}{n} M(G_{n,c_\lambda n}) \leq \ell - 1 + c_\lambda - \varepsilon'$ almost surely.

Since $G_{n,cn}$ can be obtained from $G_{n,c_\lambda n}$ by adding $(c - c_\lambda)n$ vertices of weight 1 with random connections, and this can increase the size of a maximum allocation by at most $(c - c_\lambda)n$, we also have $\lim_{n\to\infty} \frac{1}{n} M(G_{n,cn}) \leq \ell - 1 + c - \varepsilon'$ almost surely. ◀

9.5. Closing the Gap – Proof of the Main Theorem

The key ingredient still missing to prove Theorem D is Lemma 9.5, stated on page 112.

Proof of Lemma 9.5. Call a set $X \subset \mathbb{Z}_n = V(\hat{W}_n)$ a *bad set* if it induces hyperedges of total weight $\ell|X|$ or more. We now consider each possible size t of X and each possible number αt ($0 < \alpha \leq 1$) of contiguous segments of X separately, using the first-moment method to bound the probability that a bad set X with such parameters exists, later summing over all t and α.

For now, let t and α be fixed and write X as the disjoint union of non-empty, non-touching[1] intervals $X = X_1 \cup X_2 \cup \ldots \cup X_{\alpha t}$ arranged on the cycle $\mathbb{Z}_n$ in canonical ordering and with X_1 being the interval containing $\min X$. We write the complement $\mathbb{Z}_n - X = Y_1 \cup Y_2 \cup \ldots \cup Y_{\alpha t}$ in a similar way. It is almost possible to reconstruct X from the sets $\{x_1, \ldots, x_{\alpha t}\}$ and $\{y_1, \ldots, y_{\alpha t}\}$ where $x_i \coloneqq |X_1 \cup \ldots \cup X_i|$ and $y_i \coloneqq |Y_1 \cup \ldots \cup Y_i|$, we just do not know where X_1 starts. To fix this, we exploit that $x_{\alpha t}$ and $y_{\alpha t}$ are always t and $n - t$, respectively, and do not really encode information. In the case $0 \in X$, we set $x'_{\alpha t} \coloneqq \max X_1 \in [0, \ldots, x_1)$ and $y'_{\alpha t} \coloneqq y_{\alpha t}$; if $0 \notin X$ we set $y'_{\alpha t} \coloneqq \max Y_1 \in [0, \ldots, y_1)$ and $x'_{\alpha t} \coloneqq x_{\alpha t}$. The sets $\{x_1, \ldots, x_{\alpha t - 1}, x'_{\alpha t}\} \subseteq \{0, \ldots, t\}$ and $\{y_1, \ldots, y_{\alpha t - 1}, y'_{\alpha t}\} \subseteq \{0, \ldots, n - t\}$ now uniquely identify X, meaning there are at most $\binom{t+1}{\alpha t}\binom{n-t+1}{\alpha t}$ choices for X. No matter the choice, X induces helper hyperedges of weight precisely $(\ell - 1) \cdot (t - \alpha t)$, since for

[1] Two intervals touch if their union is an interval, i.e. if there is no gap in between them.

each $x \in X$ the hyperedge $(x, x+1)$ is induced, except if x is the right endpoint of one of the αt intervals. In order for X to induce a total weight of ℓt or more another

$$\ell t - (\ell - 1)(t - \alpha t) = t + (\ell - 1)\alpha t \geq t + \alpha t$$

ordinary hyperedges (of weight 1) need to be induced. There are $\binom{cn}{t+\alpha t}$ ways to choose such a set of hyperedges and each hyperedge has all endpoints in X with probability $(\frac{t}{n})^k \leq (\frac{t}{n})^2$.

Together we obtain the following upper bound on the probability that a bad set of size t with αt contiguous regions exists:

$$\underbrace{\binom{t+1}{\alpha t}\binom{n-t+1}{\alpha t}}_{\text{choices for } X} \underbrace{\binom{cn}{t+\alpha t}}_{\text{choices for hyperedges}} \left(\frac{t}{n}\right)^{2(t+\alpha t)} \leq 2^{t+1}\binom{n}{\alpha t}\binom{n}{t+\alpha t}\left(\frac{t}{n}\right)^{2(t+\alpha t)}$$

$$\leq 2^{t+1}\left(\frac{ne}{\alpha t}\right)^{\alpha t}\left(\frac{ne}{t+\alpha t}\right)^{t+\alpha t}\left(\frac{t}{n}\right)^{2(t+\alpha t)} \leq 2\left(2\left(\frac{ne}{\alpha t}\right)^{\alpha}\left(\frac{ne}{t+\alpha t}\right)^{1+\alpha}\left(\frac{t}{n}\right)^{2(1+\alpha)}\right)^{t}$$

$$\leq 2\Bigg(\underbrace{2\left(\frac{e}{\alpha}\right)^{\alpha}\left(\frac{e}{1+\alpha}\right)^{1+\alpha}}_{\leq C \text{ for some } C = O(1)}\left(\frac{n}{t}\right)^{\alpha}\left(\frac{n}{t}\right)^{1+\alpha}\left(\frac{t}{n}\right)^{2(1+\alpha)}\Bigg)^{t} \leq 2\left(C\left(\frac{t}{n}\right)\right)^{t}.$$

We used that α^α is bounded (has limit 1) for $\alpha \to 0$. The resulting term is $o(1)$ for $t = 1, 2, 3$ (and only 1, 2 or 3 choices for α are possible). For $4 \leq t \leq \sqrt{n}$, it is

$$2\left(C\left(\frac{t}{n}\right)\right)^{t} \leq 2\left(\frac{C}{\sqrt{n}}\right)^{t} \leq 2\left(\frac{C}{\sqrt{n}}\right)^{4} = O(n^{-2}).$$

Summing over all combinations of $O(\sqrt{n} \cdot \sqrt{n})$ choices for α and t, we get a sum of $O(n^{-1})$. For $\sqrt{n} \leq t \leq \delta n$ with $\delta = \frac{1}{2C}$ we get

$$2\left(C\left(\frac{t}{n}\right)\right)^{t} \leq 2\left(\frac{1}{2}\right)^{t} \leq 2 \cdot 2^{-\sqrt{n}},$$

which is clearly $o(1)$ even if we sum over all $O(n^2)$ combinations for choosing t and α. ◀

Proof of Theorem D. It suffices to show that $\gamma_{k,\ell}$ is the threshold for the event $\{M(G_n) = n(\ell - 1 + c)\}$ since by Propositions 9.1 and 9.2 this event coincides with the events that $\hat{W}_n$ and W_n are orientable.

- $c > \gamma_{k,\ell}$. If $c = \gamma_{k,\ell} + \varepsilon$ for $\varepsilon > 0$, then by Proposition 9.4 we have $\lim_{n\to\infty} \frac{M(G_{n,cn})}{n} = \ell - 1 + c - \varepsilon'$ almost surely for some $\varepsilon' > 0$. This clearly implies that $M(G_{n,cn}) < n(\ell - 1 + c)$ whp.
- $c < \gamma_{k,\ell}$. Let $c = \gamma_{k,\ell} - \varepsilon$ for some $\varepsilon > 0$ and define $c' := \gamma_{k,\ell} - \frac{\varepsilon}{2}$. We generate $G_{n,cn}$ from $G_{n,c'n}$ by removing $\frac{\varepsilon}{2}n$ vertices from A_R. The idea is to derive orientability of $G_{n,cn}$ from Lemma 9.5 and "almost-orientability" of $G_{n,c'n}$.
  More precisely, let $G^{(0)} := G_{n,c'n} = (A = A_C \cup A_R, B, \text{"}\ni\text{"})$ and let $G^{(i+1)}$ be obtained from $G^{(i)}$ by removing a vertex from $A_R \cap V(G^{(i)})$ uniformly at random, $0 \leq i < \frac{\varepsilon}{2}n$. By Proposition 9.4 we have $\lim_{n\to\infty} \frac{M(G_{n,c'n})}{n} = \ell - 1 + c'$ almost surely, which implies $M(G_{n,c'n}) = n(\ell - 1 + c') - o(n) = \eta(A) - o(n)$ whp. For any subgraph G of $G^{(0)}$, define

$\mathrm{gap}(G) \coloneqq \eta(A \cap V(G)) - M(G)$ which measures how far G is away from being orientable. This ensures $\mathrm{gap}(G^{(0)}) = o(n)$ whp as well as $\mathrm{gap}(G^{(i+1)}) \in \{\mathrm{gap}(G^{(i)}), \mathrm{gap}(G^{(i)}) - 1\}$. We say a vertex $a \in V(G^{(i)}) \cap A_R$ is *good* for $G^{(i)}$ if $\mathrm{gap}(G^{(i)} - a) = \mathrm{gap}(G^{(i)}) - 1$. Assume $\mathrm{gap}(G^{(i)}) > 0$. We now show that $\Theta(n)$ vertices are good for $G^{(i)}$ whp. Let $a \in A$ be one vertex of $G^{(i)}$ that is not saturated in a maximum allocation σ of $G^{(i)}$. Let $X \subseteq B$ and $Y \subseteq A$ be the vertices from A and B reachable from a via an *alternating path*, i.e. a path of the form $(a = a_1, b_1, a_2, b_2, \dots)$ such that $\sigma(b_j, a_{j+1}) > 0$ for all j.
It is easy to check that all vertices from X are saturated in σ (otherwise σ could be increased), Y exceeds X in total weight and every vertex from $Y \cap A_R$ is good for $G^{(i)}$. Moreover, when viewed as a subset of $V(\hat{W}_{n,c'n})$, X induces at least Y. Discounting the low probability event that Lemma 9.5 does not apply to $\hat{W}_{n,c'n}$, we conclude $|X| > \delta n$, and taking into account $|Y \cap A_C| \leq |X|$ (clear from definition of A_C) together with $(\ell - 1)|Y \cap A_C| + |Y \cap A_R| = \eta(Y) > \eta(X) = \ell|X|$ we obtain $|Y \cap A_R| > \delta n$.
This means that whp on the way from $G^{(0)}$ to $G^{(\frac{\varepsilon}{2}n)}$, we start with a gap of $o(n)$ and have up to $\frac{\varepsilon}{2}n = \Omega(n)$ chances to reduce a non-zero gap by 1, namely by choosing a good vertex for removal, and the probability is at least $\delta = \Omega(1)$ every time. Now simple Chernoff bounds imply that the gap vanishes whp meaning $G^{(\frac{\varepsilon}{2}n)}$ is orientable whp. Since the distributions of $G_{n,cn}$ and $G^{(\frac{\varepsilon}{2}n)}$ coincide, we are done. ◀

9.6. Numerical approximations of the Thresholds.

Rewriting the definition of $\gamma_{k,\ell}$ in Equation (9.8) we get

$$\gamma_{k,\ell} = \inf_{\lambda > 0}\{f_{k,\ell}(\lambda) \mid g_{k,\ell}(\lambda) < 0\}$$

where $f_{k,\ell}(\lambda) = c_\lambda$ and $g_{k,\ell}(\lambda) = F(\vec{\rho}_\lambda, c_\lambda) - \ell + 1 - c_\lambda$. We give the plots of $f_{2,2}$ and $g_{2,2}$ in Figure 9.5.

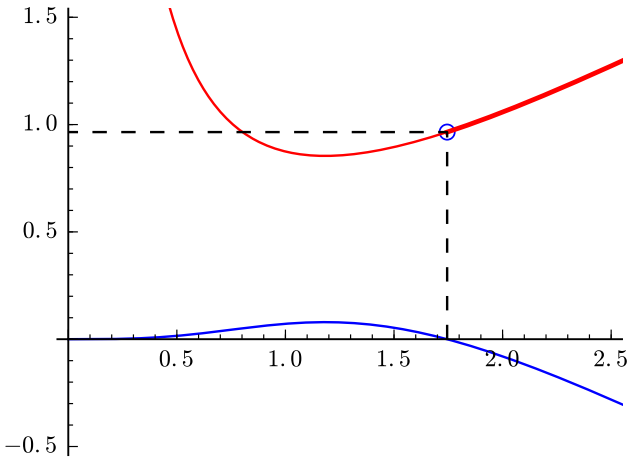


Figure 9.5 The functions $f_{2,2}(\lambda)$ and $g_{2,2}(\lambda)$, with $\gamma_{2,2} = \inf_{\lambda>0}\{f_{2,2}(\lambda) \mid g_{2,2}(\lambda) < 0\}$.

It looks as though $\{\lambda \mid g_{2,2}(\lambda) < 0\}$ is an interval $(\lambda^* \approx 1.74, \infty)$ on which $f_{2,2}$ is monotonically increasing, meaning $\gamma_{2,2} = f_{2,2}(\lambda^*) \approx 0.965$. Here is a semi-rigorous argument

that properly done plots cannot be misleading: Regardless of $k, \ell \geq 2$ it is fairly easy to see that $c_\lambda = \Omega(\lambda^{-(k-1)\ell+1})$ for $\lambda \to 0$ and $c_\lambda = \Omega(\lambda)$ for $\lambda \to \infty$. In particular, for fixed k, ℓ it is easy to obtain bounds $0 < \lambda_0 < \lambda_1 < \infty$ such that for $\lambda \in (0, \lambda_0) \cup (\lambda_1, \infty)$ we can guarantee $f_{k,\ell}(\lambda) > 1$. Because $\gamma_{k,\ell} < 1$, only the interval $[\lambda_0, \lambda_1]$ can be relevant. Being real analytic, the functions $f_{k,\ell}$ and $g_{k,\ell}$ have bounded first and second derivatives on $[\lambda_0, \lambda_1]$ which vindicates plots of sufficient resolution: There cannot be unexpected zeroes in between sampled positions and what looks strictly monotonic in the plot actually is. So starting with a golden ratio search close to the apparent root of $g_{2,2}$ we are guaranteed to find λ^* and $\gamma_{2,2} = f_{2,2}(\lambda^*)$. This can be made formal.

Handling it this way saves us the trouble of having to deal with unwieldy functions. However, our lack of analytical insight means we have to consider each pair (k, ℓ) separately to make sure that $f_{k,\ell}$ and $g_{k,\ell}$ do not exhibit qualitatively different behaviour. We did this for $\ell \leq 6$ and $k \leq 7$, i.e. for the values we provided in table Table 2.6.

Relevant Definition and Theorems for Chapter 10

(Originally on Pages 22 and 35)

► **Definition 2.9** (Double Hashing Hypergraph). *Let $n, m, k \in \mathbb{N}$ with $k \geq 3$.*

$$D^k_{n,m} := (\mathbb{Z}_n, \{e_1, e_2, \ldots, e_m\}), \quad \text{where } e_i = \{a_i + jb_i \bmod n \mid j \in [k]_0\},$$
$$\text{with } a_i \leftarrow\!\!\text{⚄} \ \mathbb{Z}_n, \ b_i \leftarrow\!\!\text{⚄} \ [n-1] \text{ for } i \in [m].$$

► **Theorem E.** *For any $k \geq 3, \ell \geq 1$, the ℓ-orientability threshold of $(D^k_{n,cn})_{c\in\mathbb{R}^+, n\in\mathbb{N}}$ is $c^*_{k,\ell}$.*

► **Theorem E1.** *Let $k \geq 3$, $\ell \geq 1$. Consider k-ary cuckoo hashing with buckets of size ℓ. Instead of using k independent, fully random hash functions, consider using 2 hash functions and double hashing. This neither affects the load threshold, nor the worst-case number of cache misses and key comparisons per query.*

10. Load Thresholds for Cuckoo Hashing with Double Hashing

The purpose of this chapter is to prove Theorem E, restated on the facing page. In other words, the ℓ-orientability thresholds of $(H^k_{n,cn})_{c\in\mathbb{R}^+,n\in\mathbb{N}}$ and $(D^k_{n,cn})_{c\in\mathbb{R}^+,n\in\mathbb{N}}$ coincide. Theorem E1 is then an immediate consequence.

Previous work has come close to this theorem as shown in Figure 10.1. Recall that for $d \in \mathbb{N}$ we say a hypergraph $H = (V, E)$ is *d-almost ℓ-orientable* if there is $E' \subseteq E$ of size $|E'| = |E| - d$ such that $H' = (V, E')$ is ℓ-orientable.

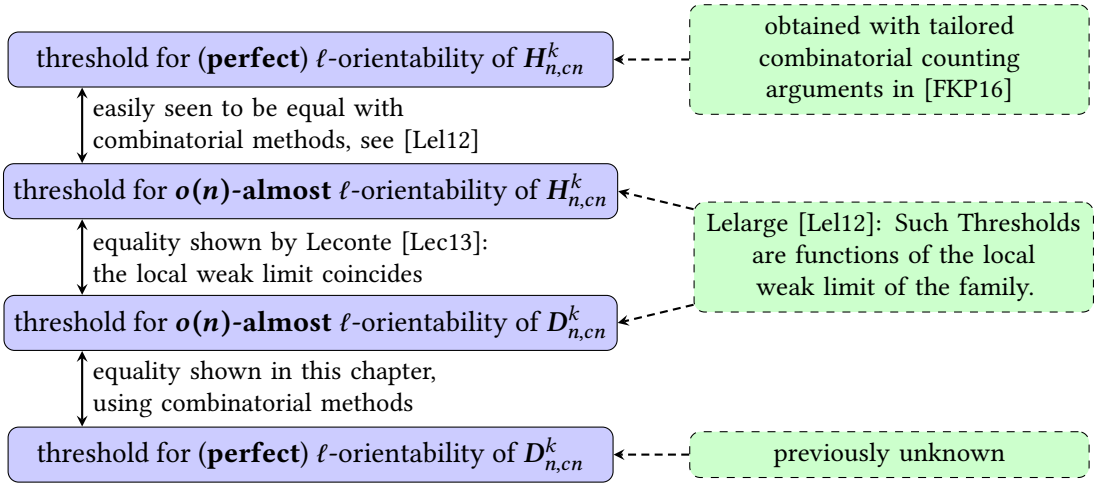


Figure 10.1 Two out of three steps for proving Theorem E are already done.

Leconte [Lec13] showed that, for all $c \in \mathbb{R}^+$, the families $(H^k_{n,cn})_{n\in\mathbb{N}}$ and $(D^k_{n,cn})_{n\in\mathbb{N}}$ have the same Galton-Watson tree as local weak limit. Lelarge [Lel12] showed that whether or not a family of hypergraphs is $o(n)$-almost ℓ-orientability only depends on the local weak limit of the family. It is fairly easy to reconcile ℓ-orientability and $o(n)$-almost ℓ-orientability for $(H^k_{n,cn})_{n\in\mathbb{N}}$ (see [Lel12]), showing that the thresholds are the same for that family. In order to establish Theorem E, all we need to prove is an analogous result for $(D^k_{n,cn})_{n\in\mathbb{N}}$, which is done in the following proposition. Note that ℓ-orientability trivially implies $o(n)$-almost ℓ-orientability, so only the non-trivial direction is given.

▶ **Proposition 10.1.** *Let $k \geq 3$ and $\ell \geq 1$ be fixed constants and $c^*_{k,\ell}$ the $o(n)$-almost ℓ-orientability threshold for $(D^k_{n,cn})_{c\in\mathbb{R}^+,n\in\mathbb{N}}$. Then for any $c < c^*_{k,\ell}$, $D^k_{n,cn}$ is ℓ-orientable whp.*

The proof uses two lemmas that are proved in Sections 10.1 and 10.2, respectively. To understand them, we need another concept. In the context of discussing ℓ-orientability of a hypergraph $H = (V, E)$, we call $V' \subseteq V$ a *Hall-witness* if the set $E(V')$ of hyperedges

induced by V' has size $|E(V')| > \ell \cdot |V'|$. By Hall's Theorem (restated in Section 10.2), H is ℓ-orientable if and only if no Hall-witness exists.

The lemmas we utilize are as follows:

▶ **Lemma 10.2.** *Let $k \geq 3$, $\ell \geq 1$ and $c > 0$ be fixed constants. Then there exists a constant $\delta > 0$ such that, whp, no Hall-witness of size less than δn exists for $D^k_{n,cn}$.*

▶ **Lemma 10.3.** *If $H = (V, E)$ is d-almost ℓ-orientable and $e \in E$ is contained in some minimal Hall-witness, then $H^{(e)} = (V, E - \{e\})$ is $(d-1)$-almost ℓ-orientable.*

Given these lemmas, we prove Proposition 10.1, following [Lel12].

Proof of Proposition 10.1. Let $c = c^*_{k,\ell} - \varepsilon$ for some $\varepsilon > 0$. We may sample $D^k_{n,cn}$ by first sampling $D^k_{n,c'n}$ for $c' = c^*_{k,\ell} - \varepsilon/2$ and then removing $\varepsilon n/2$ hyperedges. More precisely, we set $D^{(0)} \coloneqq D^k_{n,c'n}$ and obtain $D^{(i)}$ from $D^{(i-1)}$ by removing a hyperedge uniformly at random for $i \in [\varepsilon n/2]$. Then $D^{(\varepsilon n/2)}$ is distributed as $D^k_{n,cn}$.

For $0 \leq i \leq \varepsilon n/2$, let d_i be the smallest d such that $D^{(i)}$ is d-almost ℓ-orientable. By definition of $c^*_{k,\ell}$ we have $d_0 = o(n)$ whp. We take δ from Lemma 10.2 and condition on the high probability event that any Hall-witness of $D^{(0)}$ has size at least δn. Since removing hyperedges cannot create new Hall-witnesses, the same bound applies to Hall-witnesses of the subgraphs $D^{(i)}$ with $i > 1$.

Let i be an index with $d_i > 0$. Then $D^{(i)}$ is not ℓ-orientable and a minimal Hall-witness exists. Its size is at least δn, and it induces at least $\delta \ell n + 1$ hyperedges. In particular, the probability that a random hyperedge of $D^{(i)}$ is contained in this minimal Hall-witness is at least $\frac{\delta \ell n + 1}{c'n} \geq \delta \ell / c' = \Theta(1)$. If such a hyperedge is chosen for removal, then by Lemma 10.3 we have $d_{i+1} = d_i - 1$. Until we reach $D^{(\varepsilon n/2)}$, there are $\varepsilon n/2 = \Theta(n)$ opportunities to reduce the d-value by 1, and each opportunity is realized with probability $\Theta(1)$. Since the initial gap is $o(n)$, the probability that we have $d_{\varepsilon n/2} > 0$ is $\Pr[X < o(n)]$ where $X \sim \mathrm{Bin}(\Theta(n), \Theta(1))$. Simple concentration bounds on binomial random variables prove that this is an $o(1)$-probability event, so we have $d_{\varepsilon n/2} = 0$ whp. Thus $D^{(\varepsilon n/2)} \stackrel{\mathrm{d}}{=} D^k_{n,cn}$ is (perfectly) ℓ-orientable whp as desired. ◀

10.1. No small Hall-witness exists

In this section, we prove Lemma 10.2. We argue first that it is enough to prove the statement in the case $k = 3$ and $\ell = 1$. Indeed, if $D^k_{n,cn}$ contains no $V' \subseteq V$ inducing more than $|V'|$ hyperedges, then certainly no such V' induces more than $\ell|V'|$ hyperedges. Moreover, let us write $e = \{a_e + ib_e : 0 \leq i < k\}$ for a hyperedge e of $D^k_{n,cn}$. We project each hyperedge e in $D^k_{n,cn}$ to $e' = \{a_e + ib_e : 0 \leq i < 3\}$; then the resulting 3-uniform hypergraph is distributed like $D^3_{n,cn}$, and each $V' \subseteq V$ induces at least as many hyperedges as in $D^k_{n,cn}$. It therefore suffices to show the unlikeliness of small Hall witnesses in the case of $k = 3$, $\ell = 1$, and fixed $c \in \mathbb{R}^+$.

To assess the probability that a set $S \subseteq \mathbb{Z}_n$ is a Hall-witness, the size $s = |S|$ of S and the number t of arithmetic triples modulo n contained in S are both important. We call a set with corresponding parameters an (s, t)-*set*, for $3 \leq s \leq n$, $1 \leq t \leq \binom{s}{2}$. The upper bound $\binom{s}{2}$ comes from the fact that the first two terms of the arithmetic progression determine the third and each arithmetic progression can be enumerated in two directions.

Our plan is to use first moment methods and bound the sum:

$$\sum_{(s,t)} Q_{s,t} p_{s,t}$$

where $Q_{s,t}$ is the number of (s,t)-sets that could be minimal Hall-witnesses, and $p_{s,t}$ is an upper bound on the probability that an (s,t)-set actually is a minimal Hall-witness in $D^3_{n,cn}$.

We separately deal with the following ranges of the parameters s and t.

Case 1: Small s. If $s = o(n^{1/2})$ we show that $\sum_t Q_{s,t}$ is sufficiently small by direct counting.

Case 2: Medium s, small-ish t. For $s = \omega(n^{2/5})$ and $t \le \frac{s^2}{4ce^2}$, the probability $t/\binom{n}{2} \approx \frac{2t}{n^2}$ that random hyperedges are contained in such an (s,t)-set is small enough to find a good bound on $p_{s,t}$.

Case 3: Medium s, large t. For $\delta n \ge s = \omega(n^{2/5})$ (for a small δ chosen later) and $t > \frac{s^2}{4ce^2}$ it turns out that t far exceeds the number of arithmetic triples that would be expected from a random set of size s. A concentration bound by Warnke [War17] then gives a useful bound on $Q_{s,t}$.

We deal with these three cases below. We use the following simple bounds on $p_{s,t}$. For $\ell = 1$, a set of size s is a Hall-witness, if it induces at least $s+1$ hyperedges. We therefore find:

$$p_{s,t} \le \left(\frac{t}{\binom{n}{2}}\right)^{s+1} \binom{cn}{s+1} \le \left(\frac{2t}{n^2}\right)^{s+1} \left(\frac{cne}{s}\right)^{s+1} = \left(\frac{2cet}{sn}\right)^{s+1} \tag{10.1}$$

$$\le \left(\frac{ces}{n}\right)^{s+1}. \tag{10.2}$$

The bound is derived by taking the probability that for a set of $s+1$ hyperedges, each hyperedge turns out to be one of the t arithmetic triples contained in S. This is multiplied with the number of ways to choose $s+1$ out of the cn hyperedges of $D^3_{n,cn}$. For the second line we used the trivial bound $t \le \binom{s}{2} \le \frac{s^2}{2}$.

Case 1: $s = o(\sqrt{n})$. Assume $S \subseteq \mathbb{Z}_n$ is a minimal Hall-witness for $D^3_{n,cn}$ inducing a set P of hyperedges. As a hypergraph, (S,P) is spanning, i.e. each vertex is contained in a hyperedge, otherwise the isolated vertex can be removed for a smaller Hall-witness. Also, (S,P) is connected, i.e. for any $x, y \in S$ there is a sequence $e_1, \dots, e_j \in P$ with $x \in e_1, y \in e_j$ and $e_i \cap e_{i+1} \neq \emptyset$ for $1 \le i < j$). Otherwise, at least one connected component forms a smaller Hall-witness.

So for fixed s, we can count all (s,t)-sets (with arbitrary t) that might be minimal Hall-witnesses by counting vertex sets that can support connected spanning hypergraphs. We do this by counting *annotated depth-first-search-runs* (dfs-runs), associated with such (s,t)-sets, in the following way. A dfs-run through S starts at a root vertex $r \in S$ and puts it on the *stack*, whose topmost element is referred to as top. Then a sequence of steps follow, each of which either removes top from the stack (backtrack) or finds new vertices in S that are then put on the stack. More precisely, new vertices are found by specifying an arithmetic triple that is contained in S and involves top. The two vertices other than top may either both be new (find_2), or only one vertex is new, and a third vertex v was already found in a previous step (find_1). The following data about the dfs-run is needed to reconstruct S from it:

- The root vertex r. There are n possibilities.
- The type of each step, which can be backtrack, find_1 or find_2. Since there are at most $2s$ steps, there are at most 3^{2s} possibilities in total.
- For each step of type find_1, the vertex v that was previously found and that together with top and the new vertex forms an arithmetic triple. There are less than s possibilities. In addition we need the position of top and v in the arithmetic triple (essentially four possibilities). The newly discovered vertex can then be computed from top and v.
- For each step of type find_2, the difference between adjacent elements of the arithmetic triple – there are $n/2$ possibilities (considering that a "forward" difference of i corresponds to a "backward" difference of $n-i$). Also, the position of top in that triple – there are 3 possibilities.

If f_1 and f_2 count the number of times the steps find_1 and find_2 are used in the dfs-run through S of size s, then we have $f_1 + 2f_2 = s - 1$. For $s = o(\sqrt{n})$, the find_2-steps yield a significantly higher number of possibilities per found vertex compared to find_1-steps, so we compute:

$$\sum_t Q_{s,t} \le n \cdot 3^{2s} \cdot (4s)^{f_1} (3n/2)^{f_2} \le n \cdot 3^{2s} \cdot (3n/2)^{(s-1)/2} \le c_1^s n^{(s+1)/2}$$

where c_1 is a constant. Using Equation (10.2) we get:

$$\sum_{s=3}^{o(\sqrt{n})} \sum_t Q_{s,t} p_{s,t} \le \sum_{s=3}^{o(\sqrt{n})} \left(\sum_t Q_{s,t}\right) \left(\max_t p_{s,t}\right) \le \sum_{s=3}^{o(\sqrt{n})} c_1^s n^{(s+1)/2} \left(\frac{ces}{n}\right)^{s+1} \le \sum_{s=3}^{o(\sqrt{n})} \left(\frac{c_2 s}{\sqrt{n}}\right)^{s+1}$$

for a new constant c_2. Since each term in the sum is $O(n^{-2})$ and since there are $o(n^{1/2})$ terms, the sum is clearly $o(n^{-3/2}) = o(1)$, closing this case.

Case 2: $s = \omega(n^{2/5})$ and $t \le \frac{s^2}{4ce^2}$. Combining the trivial bound of $Q_{s,t} \le \binom{n}{s}$, Equation (10.1), and our assumption on t we obtain:

$$Q_{s,t} \cdot p_{s,t} \le \binom{n}{s} \cdot \left(\frac{2cet}{sn}\right)^{s+1} \le \left(\frac{ne}{s}\right)^s \cdot \left(\frac{s}{2ne}\right)^{s+1} \le \left(\frac{1}{2}\right)^s.$$

This is clearly $o(1)$, even after summing over all $O(n)$ admissible choices for s and all $O(n^2)$ choices for t.

Case 3: $\omega(n^{2/5}) \le s \le \delta n$ and $t > \frac{s^2}{4ce^2}$. A random set $S \subseteq \mathbb{Z}_n$ of size s in this range behaves very much like a random set T that is obtained by picking each element of $\mathbb{Z}_n$ independently with probability $p = \frac{s}{n}$. Let X be the number of arithmetic triples in T. We have $\mu := \mathbb{E}[X] = \binom{n}{2} p^3 \le \frac{s^3}{2n}$. In particular, the case $X > \frac{s^2}{4ce^2}$ is very rare if $s < \delta n$ for sufficiently small δ. We can therefore expect the number $Q_{s,t}$ to be significantly less than $\binom{n}{s}$. Formally we write:

$$Q_{s,t} \le \binom{n}{s} \Pr[S \text{ contains } t \text{ a.p.}] = \binom{n}{s} \Pr[X = t \mid |T| = s] \le \binom{n}{s} \Pr[X = t] O(\sqrt{n})$$

where $O(\sqrt{n})$ is the inverse of the probability of the event $|T| = s$. Using Theorem 1 from [War17] with $k = 3$, $p = s/n$, we get positive constants $b, B > 0$ such that for sufficiently large n

$$\Pr[X = t] \leq \Pr[X \geq (1 + \tfrac{t-\mu}{\mu})\mu] \overset{[\text{War17}]}{\leq} \exp(-b\sqrt{\tfrac{t-\mu}{\mu}}\sqrt{\mu}\log(1/p))$$

Using our bound on t and assuming $\delta \leq \frac{1}{4ce^2}$ we can bound the negated exponent by:

$$b\sqrt{t-\mu}\log(\tfrac{1}{p}) \geq b\sqrt{\tfrac{s^2}{4ce^2} - \tfrac{s^3}{2n}}\log(\tfrac{1}{\delta}) \geq bs\sqrt{\tfrac{1}{4ce^2} - \tfrac{\delta}{2}}\log(\tfrac{1}{\delta}) \geq bs\sqrt{\tfrac{1}{8ce^2}}\log(\tfrac{1}{\delta}) = sc_3\log(\tfrac{1}{\delta})$$

for some constant $c_3 > 0$ which yields $\Pr[X = t] \leq (\delta^{c_3})^s$. Combining this with Equation (10.2), this time assuming $\delta^{c_3} \leq \frac{1}{2ce^2}$, we can write

$$\begin{aligned} Q_{s,t} \cdot p_{s,t} &\leq \binom{n}{s}\Pr[X = t]O(\sqrt{n}) \cdot p_{s,t} \leq \left(\frac{ne}{s}\right)^s (\delta^{c_3})^s O(\sqrt{n})\left(\frac{ces}{n}\right)^{s+1} \\ &\leq O(\sqrt{n})\left(ce^2\delta^{c_3}\right)^s \leq O(\sqrt{n}) \cdot 2^{-s} \end{aligned}$$

which is $o(1)$, even when summing over all $\Theta(n)$ admissible values s and all $\Theta(n^2)$ admissible values for t. This concludes the proof of Lemma 10.2.

10.2. The significance of Hall-witnesses

To understand how Hall's Theorem relates to our situation, consider the incidence graph $G = (E, V, \text{“}\ni\text{”})$ of a hypergraph $H = (V, E)$. A $(1, \ell)$-matchings in G is a set $M \subseteq E \times V$ of edges in G such that any $e \in E$ has degree at most 1 in M and any $v \in V$ has degree at most ℓ in M. An ℓ-orientation σ of H, viewed as a set of pairs $\sigma \subseteq E \times V$, is then precisely a *hyperedge-perfect* $(1, \ell)$-matching (each $e \in E$ has degree precisely 1). We call the corresponding notion of a vertex-perfect $(1, \ell)$-matching (each $v \in V$ has degree ℓ in M) an *ℓ-saturation*.

Hall's Theorem is easily generalized as follows, where $N(X)$ denotes the direct neighbours of X in the incidence graph (note that $X \subseteq V$ and $X \subseteq E$ are both allowed) and $E(V')$ to denote the set of hyperedges contained in $V' \subseteq V$.

► **Theorem 10.4** (Hall's Theorem).

(i) *H has an ℓ-orientation $\Leftrightarrow \nexists E' \subseteq E$ with $\ell|N(E')| < |E'|$*
$\Leftrightarrow \nexists V' \subseteq V$ with $\ell|V'| < |E(V')| \Leftrightarrow$: No Hall-witness exists.

(ii) *H has an ℓ-saturation $\Leftrightarrow \nexists V' \subseteq V$ with $|N(V')| < \ell|V'|$.*

We are now ready to prove Lemma 10.3.

Proof of Lemma 10.3. Let $H = (V, E)$ be a non-ℓ-orientable hypergraph and let $S \subseteq V$ be a minimal Hall-witness to this fact, which exists by Theorem 10.4 **(i)**. Consider $H_S = (S, E(S))$, the sub-hypergraph of H induced by S. Within H_S we have $|N_{H_S}(S')| > \ell|S'|$ for any $\varnothing \neq S' \subseteq S$, as otherwise, i.e. assuming $|N_{H_S}(S')| \leq \ell|S'|$, we have

$$|E(S - S')| = |E(S) - N_{H_S}(S')| = |E(S)| - |N_{H_S}(S')| > \ell|S| - \ell|S'| = \ell|S - S'|$$

which would make $S - S'$ a smaller Hall-witness than S, contradicting minimality.

Now let $e \in E(S)$. For $H_S^{(e)} := (S, E(S) - \{e\})$ we have (replacing ">" with "≥") $|N_{H_S^{(e)}}(S')| \geq \ell|S'|$ for any $S' \subseteq S$ (the claim is trivial for $S' = \emptyset$). By Theorem 10.4 **(ii)**, there is an ℓ-saturation $M_S^{(e)}$ of $H_S^{(e)}$.

Now assuming H is d-almost ℓ-orientable and $M \subseteq E \times V$ is a corresponding $(1, \ell)$-matching of size $|E| - d$, our task is to obtain a $(1, \ell)$-matching M' with $|M| = |M'|$ in $H^{(e)} = (V, E - \{e\})$ where a hyperedge $e \in E(S)$ was removed. This will imply that $H^{(e)}$ is $(d-1)$-almost ℓ-orientable as desired.

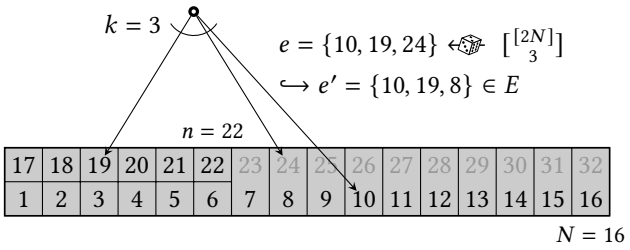
Constructing M' is easy: Simply remove all hyperedges from $E(S)$ from M (this certainly gets rid of e if it was used) and re-saturate the vertices from S by adding an appropriate subset $Y \subseteq M_S^{(e)}$. Then $M' := (M \setminus E(S)) \cup Y$ has the same size as M. ◀

Relevant Definition and Theorems for Chapter 11
(Originally on Pages 23, 24 and 37)

▶ **Definition 2.10** (DySECT Hypergraph). *Let $k, n, N, m \in \mathbb{N}$ with $N \leq n \leq 2N$.*

$H^k_{n,N,m} := ([n], E)$, *with $|E| = m$ and each $e' \in E$ chosen i.i.d. as follows:*

$$e' = \{f(j) \mid j \in e\} \text{ where } e \leftarrow \binom{[2N]}{k} \text{ and } f(j) = \begin{cases} j - N & j > n \\ j & \text{otherwise.} \end{cases}$$

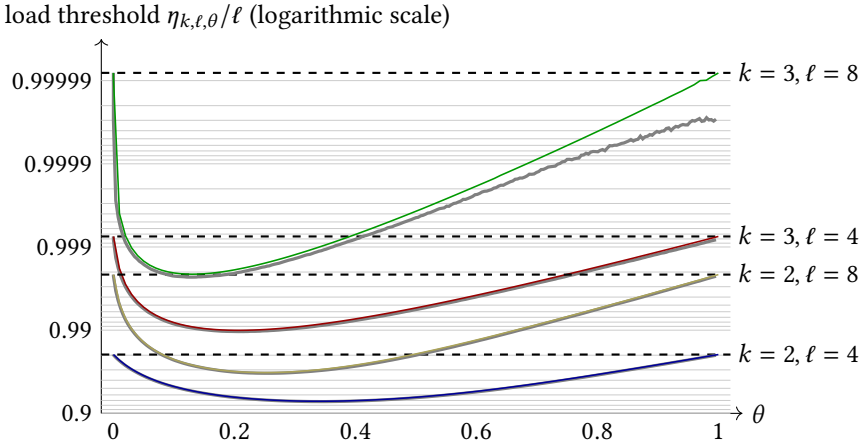


phase: $\theta = \frac{n-N}{N} = \frac{6}{16}$

▶ **Theorem F.** *Let $k, \ell \in \mathbb{N}$ with $k + \ell \geq 4$ and $\theta \in [0, 1]$. The ℓ-orientability threshold $\eta_{k,\ell,\theta}$ of $(H^k_{n,n/(1+\theta),cn})_{c\in\mathbb{R}^+,n\in\mathbb{N}}$ is characterised by Equation* (11.3) *in Chapter 11.*

▶ **Theorem F1.** *Let $k, \ell \in \mathbb{N}$ with $k + \ell \geq 4$ and $\theta \in [0, 1]$. Consider the DySECT hashing scheme with k hash functions, buckets of size ℓ at growth phase θ. The load threshold $\eta_{k,\ell,\theta}/\ell$ is characterised by Equation* (11.3) *in Chapter 11.*

Phase-Dependent Load Thresholds (originally in Figure 3.4)



11. Dynamic Space Efficient Hashing

The goal of this chapter is to prove Theorem F, restated on the facing page. Recall that it is motivated by dynamically growing cuckoo tables as explained abstractly in Section 2.4.3 and concretely in Section 3.3.1. Throughout this chapter, we have $k, n, N \in \mathbb{N}$ with $N \leq n \leq 2N$ and a *phase* $\theta = \frac{n-N}{N} \in [0, 1]$. It is helpful to imagine that $H^k_{n,N,m}$ arises from the fully random hypergraph $H^k_{N,m}$ by splitting $\theta N = n - N$ vertices of $H^k_{N,m}$. For each vertex v that is split into v_1 and v_2 this way, each incidence of v is moved to either v_1 or v_2 by flipping a fair coin. We shall call v_1 and v_2 *light vertices* in contrast to *heavy vertices* that have not been split.

The theorem characterises the ℓ-orientability threshold of $(H^k_{n,n/(1+\theta),cn})_{n\in\mathbb{N},c\in\mathbb{R}^+}$ in terms of k, ℓ and θ. In practice, N is a power of 2 implying that θ is a function of $\log_2 n - \lfloor \log_2 n \rfloor$ and DySECT tables exhibit corresponding performance fluctuations that are periodic in $\log_2 n$.

► Remark 11.1. As pointed out in [MSW19], there is an intuitive partial explanation why $0 < \theta < 1$ yields smaller thresholds than $\theta = 0$ (or $\theta = 1$). While in $H^k_{n,cn}$ every vertex has expected degree ck, in $H^k_{n,n/(1+\theta),cn}$ the light and heavy vertices have expected degree $kc(1+\theta)/2 < kc$ and $kc(1+\theta) > kc$, respectively, meaning degrees are less evenly distributed. This leads to a higher *number of bucket defects*, defined as $\sum_{v\in[n]} \max\{0, \deg(v) - \ell\}$. The number is an obvious lower bound on "unused vertex capacity" when ℓ-orientability is concerned. We leave this aside here, directly aiming for exact thresholds.

We heavily use notions from Chapter 5 and assume the reader is familiar with them. More so than usual, we abuse the notations $\mathrm{Po}(\lambda)$ and $\mathrm{Bin}(n, p)$ to not only denote Poisson and Binomial distributions but also corresponding random variables.

11.1. The Limiting Bipartite Galton-Watson Tree

Let $k \geq 2, c \in \mathbb{R}^+$ and $\theta \in [0, 1]$ be constants. Let $G^c_n = G(n, k, \theta, c)$ be the incidence graph of $H^k_{n,n/(1+\theta),cn}$. Recall the notions of *local weak convergence* and how two distributions Φ^A and Φ^B on stars give rise to a *bipartite Galton-Watson tree* $\mathrm{GWT}_{\Phi^A,\Phi^B}$. Ignoring weights for now, Φ^A and Φ^B are simply distributions on natural numbers (degrees of the stars).

► **Lemma 11.2** (Local Weak Limit). $(G^c_n)_{n\in\mathbb{N}} \xrightarrow{LWL} GWT_{\delta_k,\Phi^B}$, *where*

$$\Phi^B(i) = \tfrac{1-\theta}{1+\theta} \Pr[\mathrm{Po}(ck(1+\theta)) = i] + \tfrac{2\theta}{1+\theta} \Pr[\mathrm{Po}(ck(1+\theta)/2) = i] \text{ for } i \in \mathbb{N}_0.$$

The proof is similar to that of Theorem 5.2 and we only point out two key points relating to Φ^B. Let us write $G^c_n = (A_n, B_n, E_n)$ and consider a random vertex $b \leftarrow B_n$, corresponding to a random vertex of $H^k_{n,n/(1+\theta),cn}$. The vertex b is heavy with probability $(1-\theta)/(1+\theta)$ and light with probability $2\theta/(1+\theta)$. Assume b is heavy. Each of the cn vertices in A_n has k random incidences to B-vertices and each heavy vertex is chosen with

probability $\frac{1}{N} = \frac{1+\theta}{n}$, meaning $\deg(b) \sim \mathrm{Bin}(kcn, \frac{1+\theta}{n})$. If b is light on the other hand, we get $\deg(b) \sim \mathrm{Bin}(kcn, \frac{1+\theta}{2n})$. Standard arguments on the convergence of Binomial random variables to Poisson random variables show that Φ^B is the limiting distribution of $\deg(b)$.

At some point in the proof we consider vertices $b \in B_n$ reached in a breadth first search from a random vertex of G_n^c after a constant number of steps and need to identify the distribution of the number $\deg(b) - 1$ of successors of b in the BFS tree, conditioned on what has already been revealed about G_n. The first thing to note is that, disregarding error terms of order $O(1/n)$, b is a heavy or light vertex with probabilities $1-\theta$ and θ, respectively — not $\frac{2\theta}{1+\theta}$ and $\frac{1-\theta}{1+\theta}$ — because b was reached by a random *edge*. Depending on whether b is heavy or light we get the distributions $\deg(b) - 1 \sim \mathrm{Bin}(kcn - O(1), \frac{1+\theta}{n} - O(n^{-2}))$ or $\deg(b) - 1 \sim \mathrm{Bin}(kcn - O(1), \frac{1+\theta}{2n} - O(n^{-2}))$, respectively, where the error terms are needed since a small number of edges in G_n^c are already revealed. The distribution is easily seen to converge to $\tilde{\Phi}^B$ with

$$\tilde{\Phi}^B(i) = (1-\theta) \cdot \Pr[\mathrm{Po}(ck(1+\theta)) = i] + \theta \cdot \Pr[\mathrm{Po}(ck(1+\theta)/2) = i] \quad \text{for } i \in \mathbb{N}_0.$$

Simple calculations suffice to show that this is identical to the offspring distribution prescribed for non-root vertices of type B in the Galton-Watson tree, see Equation (5.1).

11.2. Applying Lelarge's Theorem

Now assume the A and B vertices of G_n and $\mathrm{GWT}_{\delta_k,\Phi^B}$ have weights of 1 and ℓ, respectively, as is appropriate to analyse ℓ-orientability of $H_{n,n/(1+\theta),cn}$.

With Lemma 11.2 in place, we apply Lelarge's Theorem in the special case from Equation (5.3). For the size $M(G_n^c)$ of the largest allocation of G_n^c we get:

$$\lim_{n\to\infty} \tfrac{M(G_n^c)}{cn} = \mathcal{M}(\delta_k, \Phi^B) = \inf_{p,q\in[0,1]} F(p,q,c) \quad \text{almost surely, for}$$
$$\mathbb{F}(p,q,c) = 1 - (1-p)^k + \tfrac{1}{c}\ell \Pr[\mathrm{Bin}(D^B, q) > \ell]$$

where $D^B \sim \Phi^B$, $\tilde{D}^B \sim \tilde{\Phi}^B$ and the infimum is taken over the solutions (p,q) of the equations

$$q = (1-p)^{k-1} \qquad p = \Pr[\mathrm{Bin}(\tilde{D}^B, q) < \ell]. \tag{11.1}$$

To simplify this, we use the distributional equation $\mathrm{Bin}(\mathrm{Po}(\lambda_1), \lambda_2) \stackrel{\mathrm{d}}{=} \mathrm{Po}(\lambda_1\lambda_2)$. The underlying identity

$$\Pr[\mathrm{Bin}(\mathrm{Po}(\lambda_1), \lambda_2) = i] = \sum_{j\geq i} e^{-\lambda_1} \frac{\lambda_1^j}{j!} \binom{j}{i} \lambda_2^i (1-\lambda_2)^{j-i} = e^{-\lambda_1\lambda_2} \frac{(\lambda_1\lambda_2)^i}{i!}$$

is easily verified. Applying this to the equation for p after substituting the definition of $\tilde{\Phi}^B$ we obtain

$$\begin{aligned} p &= (1-\theta) \cdot \Pr[\mathrm{Bin}(\mathrm{Po}(ck(1+\theta)), q) < \ell] + \theta \cdot \Pr[\mathrm{Bin}(\mathrm{Po}(ck(1+\theta)/2), q) < \ell] \\ &= (1-\theta) \cdot \Pr[\mathrm{Po}(cqk(1+\theta)) < \ell] + \theta \cdot \Pr[\mathrm{Po}(cqk(1+\theta)/2) < \ell]. \end{aligned} \tag{11.2}$$

11.3. The (implicit) Threshold Function

To derive a threshold η, we treat θ, k, ℓ as constants and p, q, c as variables. The set of solution triples (p, q, c) to Equation (11.1) can be parametrised by $\lambda := cq$. Using Equation (11.2) we obtain the corresponding value $p(\lambda)$ and Equation (11.1) yields $q(\lambda) = (1 - p(\lambda))^{k-1}$. The definition of λ gives $c(\lambda) = \lambda/q(\lambda)$. Lastly we write $F(\lambda) = F(p(\lambda), q(\lambda), c(\lambda))$. The only solutions not of this form are the trivial solutions with $(p, q) = (1, 0)$ that do not affect the following discussion.

We now propose a candidate for the threshold $\eta = \eta_{k,\ell,\theta}$ and immediately rewrite it using the above considerations:

$$\eta = \inf\{c \in [0, \ell] \mid \lim_{n\to\infty} \tfrac{M(G_n^c)}{cn} < 1\} = \inf\{c(\lambda) \mid \lambda > 0, F(\lambda) < 1\}. \tag{11.3}$$

Proof of Theorem F. We show that the left-hand side of Equation (11.3) really is the threshold, which implies that the right hand side is the desired characterisation.

The proof is standard. Recall that ℓ-orientability of $H^k_{\eta,n/(1+\theta),cn}$ is equivalent to $M(G_n^c) = cn$. For $c > \eta$ there is $c' \in (\eta, c)$ such that $\lim_{n\to\infty} \frac{M(G_n^c)}{cn} < 1$ almost surely, which implies $M(G_n^{c'}) < cn$ whp. By monotonicity this carries over to the higher load $c > c'$.

For $c < \eta$ we have $\lim_{n\to\infty} \frac{M(G_n^c)}{cn} = 1$ almost surely which, at first, *does not* guarantee $M(G_n^c) = cn$ whp. We only obtain $M(G_n^c) = cn - o(n)$ whp. Luckily this discrepancy can be reconciled with standard arguments, very similar to the one provided in Section 9.5 and [Lel12]. Essentially, one has to show that if $H^k_{n,n/(1+\theta),cn}$ is not ℓ-orientable, then a large set X of vertices is overloaded in the sense that every maximum partial ℓ-orientation saturates each vertex in X. Adding a single random hyperedge then increases the gap $cn - M(G_n^c)$ with positive probability, namely if all incidences of the new hyperedge are within X. This excludes the possibility that there is a non-trivial interval of values for c in which the "gap" $n - M(G_n^c)$ is positive but $o(n)$.

Put differently, we have to show that there exists a small $\delta \in (0, 1)$ such that no set of $s \le \delta n$ vertices is overloaded whp. This can be achieved with a simple union bound argument. ◀

11.4. Obtaining Numerical Approximations

Equation 11.3 is not an explicit characterization of $\eta = \eta_{k,\ell,\theta}$, but easily solved numerically. The functions $c(\lambda)$ and $F(\lambda)$ are smooth and after noting that $\lim_{\lambda\to 0} c(\lambda) = \lim_{\lambda\to\infty} c(\lambda) = \infty$, we can restrict our attention to values λ from a compact interval $[\varepsilon, 1/\varepsilon]$ where both functions have bounded derivatives. Evaluating the functions with sufficient resolution permits to approximate η with arbitrary precision (see Section 9.6 for the discussion of a similar case).

Actually, we claim without proof that $F(\lambda) - 1$ has a unique positive root λ_0 and $F(\lambda) - 1$ is negative precisely for $\lambda \in (\lambda_0, \infty)$. Moreover, $c(\lambda)$ is increasing on this interval. This implies λ_0 can be found efficiently, for instance with golden ratio search and η is simply $c(\lambda_0)$.

A plot of $\eta_{k,\ell,\theta}$ for some combinations of k, ℓ and all $\theta \in [0, 1]$ is given in Figure 3.4, together with matching experimental approximations of the thresholds for finite n.

Part IV.

Evaluation

12. Experiments

In this chapter, we consider the **practical performance** of the data structures discussed in our theorems. This is relevant for the following reasons.

- The theorems hide **constants** and offer no guidance when comparing, say, two approaches that both have "$O(1)$" times for executing **eval** operations.
- The theorems make asymptotic claims with limited implications for relevant **finite input sizes**. Especially in the cases of Theorems B2 and C2, where the input is partitioned into chunks of size n^α such that $\alpha \in (0, 1)$ is a "small constant" but n^α is still "large", it is unclear which α balances the conflicting requirements for, say, $n = 10^7$.
- Some immediate **follow-up questions** remain untouched by all theoretical considerations, for instance, construction and insertion times of cuckoo hash tables and the benefits of lazy Gaussian elimination (see Section 4.4.2) when solving linear systems arising from Theorem B2.

For the convenience of the reader, we distil the main experimental results of this chapter into compact empirical claims.

12.1. Empirical Claims

Claims α, β and γ relate to retrieval data structures from Theorems A2, B2 and C2 and their *overhead*, *eval time* (i.e. time to execute an **eval** operation) and *construction time*. They are supported by detailed experiments in Section 12.2. We use the approach GOV from [GOV16] as a basis for comparison.

▶ **Empirical Claim α.** *The retrieval data structure from Theorem A2 significantly beats GOV in terms of construction time with comparable overhead and eval time.*

A good choice for z is $z = \Theta(n^{1/3})$.

▶ **Empirical Claim β.** *In the case of 1-bit retrieval[1] the approach from Theorem B2 can be tuned to have, compared to GOV, significantly smaller overhead, similar eval times and slightly larger construction times.*

▶ **Empirical Claim γ.** *In the case of 1-bit retrieval[2] the approach from Theorem C2 with larger chunk size achieves, compared to the strategy from Claim α, even faster construction times at similar trade-offs between eval times and overhead.*

---

[1] Our experiment are for 1-bit retrieval only. Concerning r-bit retrieval for $r > 1$, Table 3.2 shows that the eval times of the approaches from Theorems B2 and C2 are sensitive to r, while GOV and the approach from Theorem A2 are not. We suspect that Claims β and γ still hold for, say, 2-bit and 3-bit retrieval, but the underlying approaches may ultimately become less attractive as r grows much larger.

[2] See previous footnote.

Alternatively, the approach can be tuned for significantly smaller overhead when significantly larger eval and construction times are tolerable.

Claims δ and ε are addressed in Sections 12.3 and 12.4 and concern the cuckoo hash tables from Theorems D1 and E1, answering some simple follow-up questions. Claim φ is related to Theorem F1 and states the main result from [MSW19]. This thesis does *not* provide independent justification in this case.

▶ **Empirical Claim δ.** *Concerning cuckoo hashing with unaligned blocks and its threshold $\gamma_{k,\ell}$ from Theorems D and D1, we claim*

1. *For $c = \gamma_{k,\ell} - \omega(1/\sqrt{n})$ the scheme works whp (not just for $c = \gamma_{k,\ell} - \Theta(1)$).*
2. *For $c = \gamma_{k,\ell} - \varepsilon$ a placement of all key/value pairs can be constructed with the LSA algorithm [Kho13] in expected time $O(n \cdot f(\varepsilon))$ for some function f not depending on n.*
3. *If elements are inserted sequentially using random walk insertion, then an insertion at load $c = \gamma_{k,\ell} - \varepsilon$ has expected running time $O(g(\varepsilon))$ where g does not depend on n.*

▶ **Empirical Claim ε.** *Using double hashing instead of fully random hashing in k-ary cuckoo tables with buckets of size ℓ does not affect the expected number of evictions during random walk insertion.*

▶ **Empirical Claim φ.** *If a hash table must dynamically grow (and shrink) to maintain high memory efficiency ($> 90\%$) at every point of its lifetime, DySECT beats all competing strategies in relevant benchmarks. [MSW19, Section 7]*

12.2. Benchmarks for Retrieval Data Structures

Let the names COUPLED, 2-BLOCK and 1-BLOCK denote our new approaches for building retrieval data structures from Theorems A2, B2 and C2, respectively. We have implemented those, and for comparison also the existing approaches BPZ [BPZ13], GOV [GOV16] and LMSS [Lub+01] discussed in Section 3.4.3. Some approaches achieve several relevant performance trade-offs for different parameters. Let us call an approach with fixed parameters a *configuration*. A list of all configurations we have attempted and their performance is given in Figure 12.1, which we shall now discuss in detail. The space overheads are plotted against each of the three measured running times, namely construction time and eval times in an "in-cache" and "out-of-cache" setting (see below).

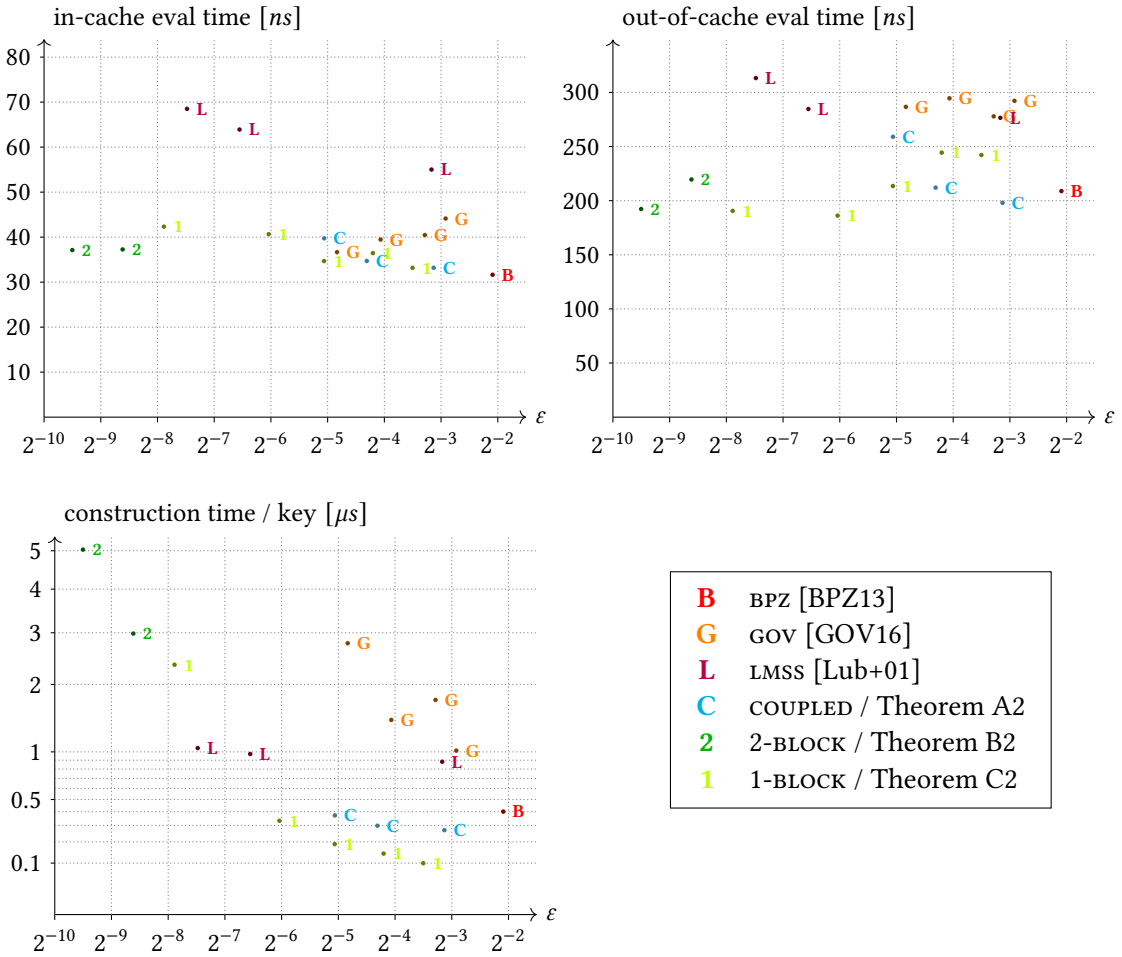
We shall have to provide some context. Before going into the individual approaches, we say a few words about the testing framework.

12.2.1. Testing Framework

A C++ implementation of the experiments is available online [Wal20a]. The general setup is as follows.

Input size. The input size is $m = 10^7$. Some configurations are tuned for this size and parameters would have to be adapted for smaller values of m.

Universe of Keys. We use a set S of random distinct 64-bit integers as keys. Using "real data", namely URLs from the `eu-2015-host` dataset gathered by [Bol+14], does not produce qualitatively different results, only diluting them somewhat due to the universally increased cost for hashing large keys.



Approach Parameters	Overhead	construct [μs/key]	cached eval[ns]	uncached eval[ns]
BPZ($c = 0.81$)	23.46%	0.40	32	209
GOV($k = 3, c = 0.91, C = 10000$)	10.24%	1.74	40	278
GOV($k = 3, c = 0.90, C = 1000$)	12.91%	1.02	44	292
GOV($k = 4, c = 0.97, C = 10000$)	3.44%	2.78	37	287
GOV($k = 4, c = 0.96, C = 1000$)	5.97%	1.43	39	295
LMSS($D = 12, c = 0.90$)	11.12%	0.88	55	277
LMSS($D = 150, c = 0.99$)	1.06%	0.97	64	285
LMSS($D = 800, c = 0.997$)	0.56%	1.05	69	313
COUPLED($k = 3, z = 120, c = 0.91$)	11.42%	0.27	33	198
COUPLED($k = 4, z = 120, c = 0.96$)	5.04%	0.30	35	212
COUPLED($k = 7, z = 120, c = 0.98$)	3.00%	0.37	40	259
2-BLOCK($\ell = 16, c = 0.9995, C = 10000$)	0.25%	2.98	37	220
2-BLOCK($\ell = 16, c = 0.9995, C = 20000$)	0.14%	5.03	37	192
1-BLOCK($L = 64, c = 0.93, C = 10000$)	8.79%	0.10	33	242
1-BLOCK($L = 64, c = 0.96, C = 10000$)	5.43%	0.14	36	244
1-BLOCK($L = 128, c = 0.98, C = 20000$)	2.99%	0.19	35	214
1-BLOCK($L = 192, c = 0.99, C = 50000$)	1.52%	0.33	41	186
1-BLOCK($L = 1024, c = 0.9960, C = n$)	0.41%	2.36	42	191

Figure 12.1 Experimental overheads and running times of the retrieval approaches we implemented, in various configurations. Plots and table display the same data.

The Function to be Stored. The function $f\colon S \to \{0,1\}$ to be stored is simply the parity $f(s) \coloneqq s \bmod 2$. Since the number and type of operations performed by **construct** and **eval** is (mostly) independent of f, this trivial choice does not affect the measurements.

Hash function. We use `xxhash` [Col20] to produce a 128-bit hash value, the usage of which depends on the approach. Unless stated otherwise, configurations that need additional hash bits use techniques resembling double-hashing to avoid further evaluations of the hash function.

Note that using `xxhash` departs from the full randomness assumption made by our theorems.

Storing Chunk Offsets and Seeds. The approaches GOV, 1-BLOCK and 2-BLOCK use input partitioning as explained in Section 3.4.2 and for each chunk, an offset and a seed value has to be stored. We implement two strategies for doing so. The default *packing strategy* uses 26 bits for the offset and 6 bits for the seed, packing both into a 32-bit word for each chunk. The alternative *compression strategy* is more compact. Instead of actual offsets, it stores for the i-th chunk the *deviation* of its offset from its *expected offset*, which is the offset the i-th chunk *would have* if all chunks had the expected size. The maximum seed and the maximum deviation determine the number of bits used for each chunk and each seed, and all values are concatenated into one large array of bits. The resulting data structure is smaller, but recovering offset and seed of a chunk is slightly slower due to an unaligned load and additional computation.

Chosen Seeds. When choosing parameters aggressively, we occasionally see a failure in a construction of approaches not using input partitioning. In this case, construction would be repeated with a new seed and construction time almost doubles. We hide this effect in the data[3].

Data Dependency Between Evals. Reported eval times are obtained by performing eval operations for many keys in a for-loop and dividing the time for the execution of the loop by the number of operations. To prevent the processor from parallelising calls internally, we introduce an artificial dependence of each operation on the result of the previous one. This affects the measurements significantly.

In-Cache vs Out-Of-Cache. With $m = 10^7$, the retrieval data structures occupy $n = (1+\varepsilon)\cdot 10^7$ bits and comfortably fit into the processor's (L3-)cache even for the largest overhead $\varepsilon = 0.23\%$ we encounter. We therefore assume to see "in-cache" performance by default. For the "out-of-cache" setting, we make 1024 copies of the retrieval data structure and perform evals in a round-robin fashion on these copies.

Overhead measurement. By the space overhead of a configuration we mean $\frac{N}{m} - 1$ where N is the total number of bits owned by the data structure, including all auxiliary data. A complete account of what contributes to N is given below for the case of 2-BLOCK.

Parallel Construction. All constructions using input partition could be parallelised in a straightforward way. We have not pursued this.

Averaging. All reported numbers are averages of 10 executions.

Testing Machine. Experiments are performed on a Microsoft Surface Pro 6 with an Intel Core i5-8250U Processor with a maximum single-core frequency of 3.40GHz. This is

[3] We hard-code seeds that we know to work. Occasionally prescribing a seed of 1 instead of 0 is sufficient.

clearly not the fastest conceivable setup but should suffice for assessing the *relative* performance of the implemented approaches.

12.2.2. On the Experiments with BPZ

The approach is based on peeling the hypergraph $H^3_{m/c,m}$ for $c < c^{\triangle}_{3,1} \approx 0.81$. We chose a straightforward implementation. Had we used input partitioning, we might have seen improvements in construction times for cache efficiency reasons, cf. [Bel+14].

12.2.3. On the Experiments with GOV

The approach is based on solving the linear system underlying the hypergraph $H^k_{m/c,m}$ for $c < c^*_{k,1}$ and $k \in \{3, 4\}$ (though other values of k are theoretically possible).

We implemented the LazyGauss algorithm suggested by the authors but solve the residual dense linear system using the Method of Four Russians discussed in Section 4.4.3 instead of ordinary Gaussian elimination. Input partitioning is required for feasible construction times. To demonstrate the effect of the chunk size C we provide configurations with $C = 1000$ and $C = 10000$.

When using the smaller chunk size, we use a slightly smaller load c to keep the fraction of chunks for which the first construction attempt fails at a low level, since keys in such chunks require an additional (costly) hash function evaluation during evals. Since, for $C = 1000$, one 32-bit word of meta-data per chunk would mean an additional overhead of 3.2%, we switch to the compression strategy for storing offsets and seeds.

12.2.4. On the Experiments with LMSS

The approach is based on peeling a hypergraph with maximum hyperedge size $D + 4$, average hyperedge size $\approx \ln(D)$ and hyperedge density at most $1 - 1/D$. To our knowledge, these hypergraphs have not been considered in the context of retrieval. The configuration with $D = 800$ and $c = 0.997$ is an attempt to test how far our implementation of the approach can be stretched in terms of overhead for $m = 10^7$. Construction attempts with $c = 0.998$ have failed, even for much larger values of D.

12.2.5. On the Experiments with COUPLED

Recall that the approach is based on peeling $F_n = F(n, k, c, z)$. The choice of k affects the upper bound $c^*_{k,1}$ on the achievable hyperedge density we denote by $\hat{c} = m/n$. Recall that the logical density parameter $c = \hat{c}\frac{z+1}{z}$ is slightly higher. We provide experiments for $k \in \{3, 4, 7\}$.

On Choosing z. Our theoretical considerations treat z as a constant and offer no guidance in how $z = z(n)$ should be chosen in practice. But consider the following heuristic argument, suggesting that $z = \Theta(n^{1/3})$ maximises $\hat{c}$.

For finite z and n, two things keep us from achieving $\hat{c} = c^*_{k,1}$. *Firstly*, vertices with positions $x \in [0, 1] \cup [z, z + 1]$ at the borders have on average half the expected degree compared to other vertices. This leads to an overhead of $\varepsilon_1 = c - \hat{c} \approx 1/z$. *Secondly*, we need to choose $c = c^*_{k,1} - \varepsilon_2$ slightly smaller than $c^*_{k,1}$. To see why, imagine the peeling process

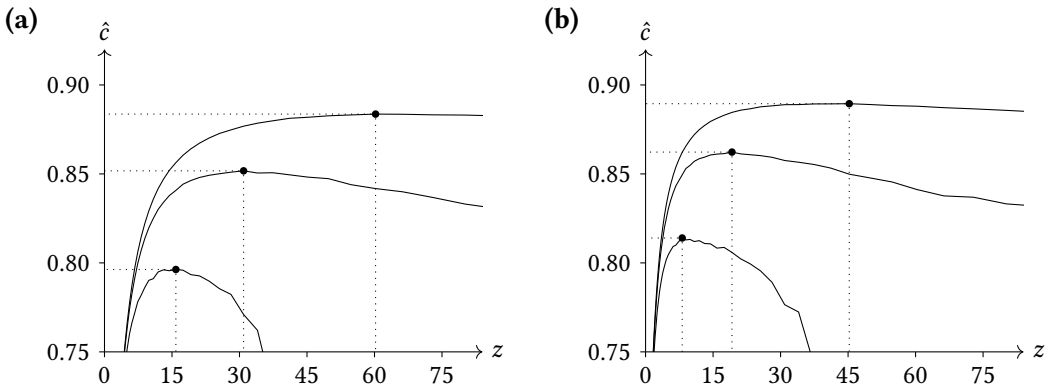


Figure 12.2 (a) Hyperedge densities $\hat{c}$ such that $F(n, 3, \hat{c}\frac{z+1}{z}, z)$ is 1-peelable with probability $\frac{1}{2}$, for different choices of z. The three plot lines correspond, from bottom to top, to $n = 10^4$, $n = 10^5$, $n = 10^6$.
(b) Using the idea of *folding* the hypergraph, slightly larger maximum hyperedge densities can be achieved.

working linearly through the coupling dimension X. What happens around $x \in X$ should mostly depend on the expected $O(n/z)$ hyperedges with positions close to x. The standard deviation of their number is $O(\sqrt{n/z})$. If the local density is correspondingly increased by $O(\sqrt{n/z}/(n/z)) = O(\sqrt{z/n})$, this should not lead to the local density exceeding $c^*_{k,1}$ (otherwise we might get stuck), which calls for $\varepsilon_2 = c^*_{k,1} - c \geq O(\sqrt{z/n})$. Balancing ε_1 and ε_2 yields our recommendation of $z = \Theta(n^{1/3})$.

Supporting experiments are found in Figure 12.2 **(a)**. There we estimate, for $n \in \{10^4, 10^5, 10^6\}$ and various z the value $\hat{c}(n, z)$ for which $F(n, 3, \hat{c}(n, z)\frac{z+1}{z}, z)$ is 1-peelable with probability exactly $1/2$. To do so, we construct 500 hypergraphs with distribution $H \stackrel{\text{d}}{=} F(n, 3, 1, z)$. Let the hyperedge set be $E = \{e_1, \ldots, e_m\}$. We then determine the values

$$m^* = \max_{m' \in [m]} \{H' = ([n], \{e_1, \ldots, e_{m'}\}) \text{ is 1-peelable}\}.$$

This can be done by a customised peeling process on H that, whenever no vertex of degree 1 exists, deletes the hyperedge with highest index. Then $m^* + 1$ is the highest index of a hyperedge that was deleted using the special rule. The median of the values $\frac{m^*}{n}$ over the 500 runs is our approximation of $\hat{c}(n, z)$. The z values maximising $\hat{c}(10^4, z)$, $\hat{c}(10^5, z)$ and $\hat{c}(10^6, z)$ differ by a factor of roughly 2, close to the factor of $10^{1/3} \approx 2.15$ predicited by the argument above.

The choice of $z = 120$ for $m = 10^7$ is extrapolated from these observations.

Additional Space Savings by Folding. The coupling dimension $X = [0, z + 1]$ has two border regions $[0, 1]$ and $[z, z + 1]$ and F_n has two regions with correspondingly reduced density. Intuitively, just one border is required for the peeling process to get going.

The following trick was found by Thomas Mueller Graf (personal communication): We start with $F(2n, k, \cdot, 2z)$ and identify for $i \in \{0, \ldots, n - 1\}$ the vertices i and $2n - i - 1$. This folds the hypergraph in the middle, with the borders landing on top of each other. As shown in Figure 12.2 **(b)**, slightly higher densities can be achieved this way. In the experiments of Figure 12.1, this trick is not used.

12.2.6. On the Experiments with 2-BLOCK

Recall that the approach is based on solving several linear systems of the form $A^L_{m,m/c} \cdot \vec{z} = \vec{b}$. The choices $L = 16$ and $c = 0.9995$ turned out to work well in practice. We try out chunk sizes $C = 10000$ and $C = 20000$. The latter achieves a smaller overhead but has a larger construction time.

The Benefits of Lazy Gaussian Elimination. We use the Lazy Gauss algorithm as described in Section 4.4.2 to act as a presolver for the linear systems. While we can give no theoretical guarantees, in the experiments for $L = 16$ and independently of n, about 85% of variables and equations are removed as shown in Figure 12.3. This considerably reduces the running time of the subsequent step, which uses the Method of Four Russians Section 4.4.3. In Figure 12.4 we show the relative contribution of the Lazy-Gauss and the Four-Russian phases.[4]

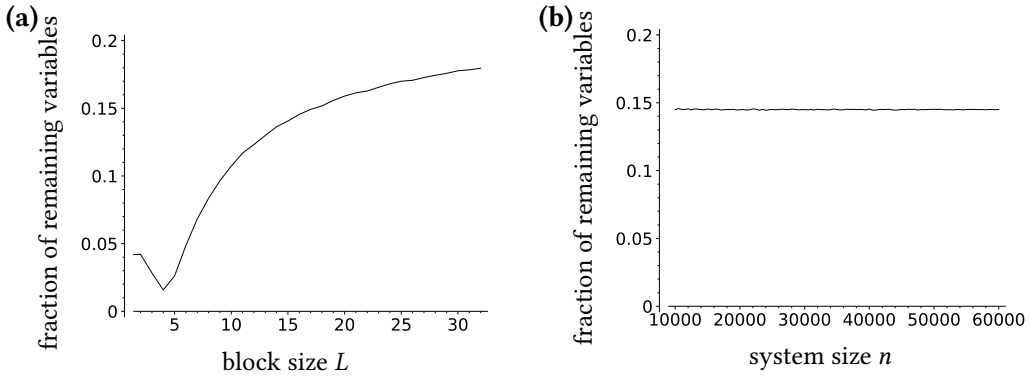


Figure 12.3 Fraction of variables that remain in the dense system after applying the Lazy Gauss presolver to a linear system $A^L_{n,n}\vec{x} = \vec{b}$. **(a)** is for $n = 10000$ and variable L while **(b)** is for $L = 16$ and variable n. Each data point is the average of 25 independent executions.

Breakdown of the Overhead. Let us see where the overhead of $\approx 0.25\%$ in the experiments with $C = 10^4$ comes from.

For $i \in [m/C]$, let m_i be the actual chunk sizes that randomly fluctuate around the desired chunk size C. The choice of $c = 0.9995$ dictates that $(1+\varepsilon)m_i$ bits should be used for the i-th chunk where $\varepsilon = 1/c \approx 0.0005$. Let n_i be the least multiple of $L = 16$ that is at least $(1+\varepsilon)m_i$. Note that $n_i - (1+\varepsilon)m_i$ has an expectation of roughly $\frac{L-1}{2} = 7.5$. Generating and solving a system $A^L_{m_i,n_i}\vec{z}_i = \vec{b}_i$ yields $\vec{z}_i \in \{0,1\}^{n_i}$. Construction is repeated with a new seed if necessary. Let s_i be the seed of the first successful construction for chunk i.

The vectors $\vec{z}_i$ are concatenated into one bit string $\vec{z}$. Let $o_i = \sum_{j<i} n_i/L$ be the offset (counted in blocks) of z_i within z. Using the compression strategy for storing offsets and

[4] The data may have been obtained with an older (but not substantially different) version of the code, possibly on a different machine.

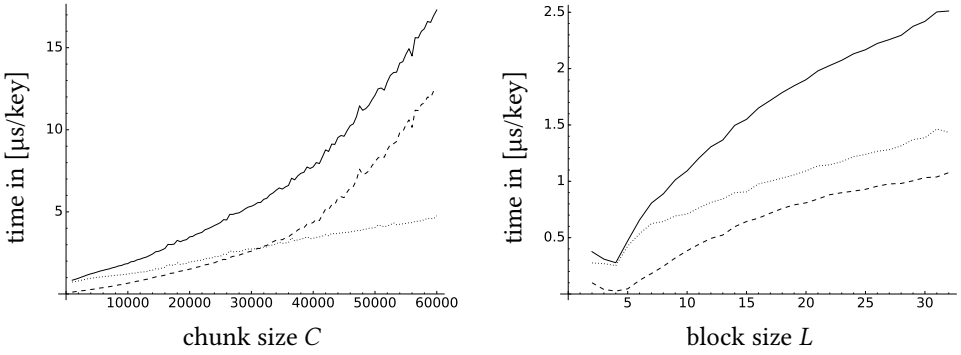


Figure 12.4 Contributions to construction time per key in the 2-BLOCK approach. ⋰ is the time for the Lazy-Gauss phase, ⋰ the time for the Four-Russian phase and ⟋ the sum. On the left the block size is $L = 16$ and the chunk size C varies. On the right $C = 10^4$ is fixed and L varies.

seeds, we actually store $d_i = o_i - \lfloor \frac{i-1}{m/C} |\vec{z}|/L \rfloor \approx o_i - \mathbb{E}[o_i]$ instead of o_i as the binary representation of d_i is typically only half as long.

Finally, let $\hat{d} := \max_i d_i$ and $\hat{s} := \max_i s_i$. In addition to $\vec{z}$, m, C, $\hat{d}$ and $\hat{s}$ we need to store the meta data $((s_i, d_i))_{i \in [m/C]}$ for the chunks using $(\lceil \log(\hat{d}+1) \rceil + \lceil \log(\hat{s}+1) \rceil) m/C$ bits. A full account of everything that needs to be stored with concrete numbers is given in Table 12.1.

Number of bits	bits used for	per element
m	entropy lower bound	1.00000
εm	planned overhead	0.00050
$\sum_i n_i - (1+\varepsilon)m$	padding ensuring $L \mid n_i$	0.00072
$\lceil \log(1+\max_i s_i) \rceil \cdot m/C$	seed for each chunk	0.00030
$\lceil \log(1+\max_i d_i) \rceil \cdot m/C$	offset info for each chunk	0.00100
	everything	1.00253

Table 12.1 Overview of the space usage of the 2-BLOCK approach. The concrete values on the right correspond to a run on a data set with $m = 10^7$ keys, chunk size $C = 10^4$, $L = 16$ and $c = 0.9995$. In that run $\lceil \log(1+\max_i s_i) \rceil = 3$ and $\lceil \log(1+\max_i d_i) \rceil = 10$.

12.2.7. On the Experiments with 1-BLOCK

Recall that the approach is based on solving several linear systems of the form $M^L_{m,m/c} \cdot \vec{z} = \vec{b}$. On a 64-bit machine, there is little reason not to choose L as a multiple of 64. The closer c is to 1, the higher the construction time becomes due to the higher number of row additions and—if we are forced to increase L—the higher cost of each row addition.

The configuration with the extremely high value of $L = 1024$ demonstrates that very small overheads are possible with this approach, albeit with high construction times. Input partitioning is not needed in this case.

The first *two* 64-bit words of each L-bit pattern are generated with `xxhash` (prompting one call to `xxhash` in addition to the call used for all approaches) and the remaining words

are generated from these two using double hashing, i.e. the $L/64$ words form an arithmetic progression modulo 2^{64}. Using arithmetic progressions modulo 2^{32} would have saved one call to `xxhash` but failed to produce linearly independent rows reliably.

12.2.8. Discussion

From Figure 12.1 we see that, if extremely small overhead is a priority, 2-BLOCK and, possibly, 1-BLOCK seem to be the best choices. If eval times are the main concern, then LMSS is worse than the other approaches and has the additional disadvantage of high variance in eval times since, in the underlying hypergraph, the largest hyperedge size is exponential in the average hyperedge size. Among the other approaches, results are too close to confidently call a clear winner. When construction time is also taken into account, 1-BLOCK and COUPLED have a clear advantage over the other approaches, with 1-BLOCK being faster than COUPLED.

On the downside, COUPLED can achieve small overheads only for relatively large values of m as seen in Figure 12.2, and 1-BLOCK may lose some of its relative advantage when r-bit retrieval for $r > 1$ is needed.

Concerning ease of implementation, the peeling approaches BPZ, COUPLED and – with some reservations – LMSS are the simplest, GOV and especially 2-BLOCK are the most challenging and 1-BLOCK is somewhere in between.

Overall we believe COUPLED is our most promising contribution. However, more research is required to explore the complex space of possible input sizes, configurations and achievable trade-offs between overhead and running times. A full exploration is beyond the scope of this more theoretically oriented thesis.

12.3. Experiments on Cuckoo Hashing with Unaligned Blocks

12.3.1. Speed of Convergence and Practical Table Sizes

It is natural to wonder to what degree the asymptotic results on $W^{k,\ell}_{n,cn}$ predict the behaviour of corresponding hash tables for realistic values of n, say a hash table of size $n = 10^5$. Formally, for $k, \ell \geq 2$ and $t \in [0, 1]$ we define the functions

$$p^{k,\ell}_n(c) \coloneqq \Pr[W^{k,\ell}_{n,cn} \text{ is } \textit{not} \text{ 1-orientable}] \quad \text{and} \quad \operatorname{step}_t(c) = \begin{cases} 0 & \text{if } c < t \\ \frac{1}{2} & \text{if } c = t \\ 1 & \text{if } c > t \end{cases}$$

Theorem D shows that $p^{k,\ell}_n : [0, 1] \to [0, 1]$ converges point-wise to $\operatorname{step}_{\gamma_{k,\ell}}$, except, possibly, at the threshold $c = \gamma_{k,\ell}$ itself. To give an idea of the speed of this convergence, we plotted approximations of $p^{k,\ell}_n$ for $n \in \{10^4, 10^5, 10^6\}$ and $(k, \ell) \in \{(2, 2), (2, 3), (3, 2)\}$ in Figure 12.5.

To obtain the data, we carried out 1000 trials. In each trial, a copy of $W^{k,\ell}_{n,n}$ was generated by adding random hyperedges one by one as long as the hypergraph remained 1-orientable. The number of hyperedges m where 1-orientability was lost corresponds to a load $c = \frac{m}{n}$. As an estimate for $p^{k,\ell}_n(c)$ we take the fraction of the 1000 trials where 1-orientability was lost at a value less than c.

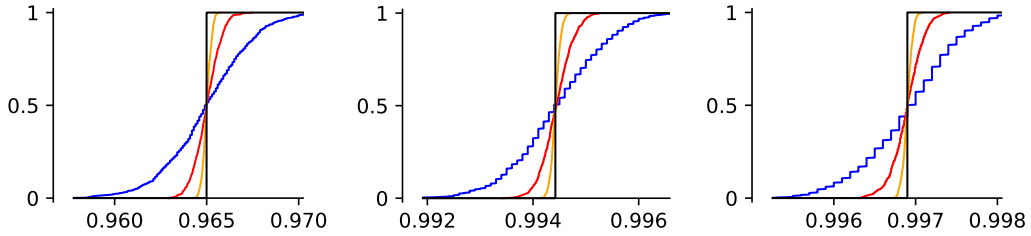


Figure 12.5 Approximate probabilities for $W^{k,\ell}_{n,cn}$ to not be orientable depending on c. From left to right, the plots correspond to $(k,\ell) \in \{(2,2),(2,3),(3,2)\}$. In each plot, the three curves correspond to $n \in \{10^4, 10^5, 10^6\}$, with curves for larger n visibly getting closer to the step function that we know is the limit for $n \to \infty$. Details are given in the text.

The plots suggest that, at the threshold, $p^{k,\ell}_n$ assumes a value of $\frac{1}{2}$ and rises from almost zero to almost one within an interval of size $O(1/\sqrt{n})$.

12.3.2. Linear Time Construction of an Orientation

The following paragraphs concern the construction of a 1-orientation of $W^{k,\ell}_{n,cn}$ and therefore the construction of a cuckoo hash table with unaligned blocks. However, they are stated in the equivalent form involving $\hat{W}^{k,\ell}_{n,cn}$, introduced in Section 9.2.

Algorithms to orient random hypergraphs include the *selfless algorithm* analysed by Cain, Sanders and Wormald [CSW07] and an algorithm by Fernholz and Ramachandran [FR07] involving so-called *excess degree reduction*. While Dietzfelbinger et al. [Die+10] have suggested a generalisation of the selfless algorithm, the algorithm that seemed easiest to adapt to our particular hypergraph setting is the Local Search Allocation (LSA) algorithm by Khosla [Kho13] (with improved analysis in [KA19]).

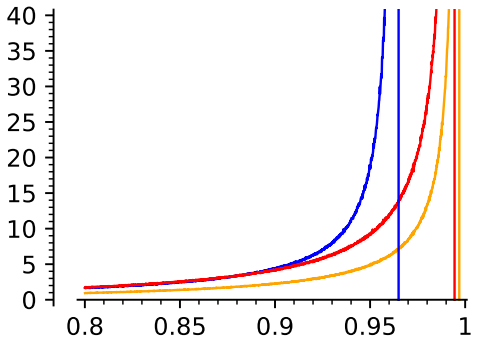


Figure 12.6 Average number of balls touched (one plus the number of evictions) to insert a new ball at a certain load using Khosla's LSA algorithm (each point in the plot being the average of 10000 insertions). The blue, red and orange curves correspond to k-ary cuckoo hashing with unaligned blocks of size ℓ for $(k,\ell) \in \{(2,2),(2,3),(3,2)\}$, respectively, and a table of size $n = 10^7$. The corresponding thresholds are shown as vertical lines. Analogous plots for $n = 10^6$ are visually indistinguishable, suggesting the plots are stable as n varies.

The Algorithm. Following Khosla's terminology, we describe the task of orienting $\hat{W}^{k,\ell}_{n,cn}$ as a problem of placing balls into bins. There are n bins of capacity ℓ arranged in a circle, and for each pair of adjacent bins, there are $(\ell - 1)$ helper balls that may be placed into one of those two bins. Moreover, there are ordinary balls, each of which has k random bins it may be placed into.

We start with all helper balls placed in their "left" option. In particular, all bins have room for just one more ball. Now the ordinary balls are placed one by one. We maintain a label for each bin. All labels are natural numbers, initially zero. We place a ball simply by putting it in the admissible bin with least label. If placing a ball results in an overloaded bin b, one ball must be evicted from b and inserted again. The ball to be evicted is chosen to have, among all balls in b, an alternative bin of least label.

Whenever the content of a bin b changes (after insertion or insertion + eviction), its label is updated. The new label is one more than the least label of a bin that is the alternative bin of a ball currently placed in b.

Analysis. Labels can be thought of as lower bounds on the distance of a bin to the closest non-full bin in the directed hypergraph where bins are vertices and a hyperedge from b_1 to b_2 indicates that a ball in b_1 has b_2 as an alternative bin. It is fairly easy to see that this algorithm finds a placement in quadratic time whenever a placement exists. To show that running time is linear whp, it suffices to show that the running time is linear in the sum of all labels in the end and that the sum of the distances mentioned above is linear whp. We do not attempt a proof here, although we expect it to be possible with Khosla's techniques.

Experiments. The results from Figure 12.6 suggest that the expected number of evictions per insertion is bounded by a constant as long as the load c is bounded away from the threshold. Eviction counts sharply increase close to the threshold.

12.3.3. Random Walk Insertion

Even though no complete analysis of random walk insertion (see Figure 3.2) for cuckoo hash tables is known, the algorithm is widely believed to achieve expected constant insertion time as long as the hash table's load is bounded away from the underlying hashing scheme's threshold. For cuckoo hashing with unaligned blocks, this seems to be no different as shown in Figure 12.7 **(a)**.

To obtain the data, we have inserted random keys using random walk insertion into an initially empty hash table. For each insertion, we count the number of evictions it caused. For $i \in [100]_0$ a plot point $(\frac{i}{100}, e)$ shows the average number e of evictions caused by the insertions at loads between $\frac{i}{100}$ and $\frac{i+1}{100}$, averaged over 10 runs of the experiment.

The curves for $n \in \{10^4, 10^5, 10^6, 10^7\}$ are almost indistinguishable, suggesting the expected number of evictions is independent of n.

12.4. Experiments on Cuckoo Hashing with Double Hashing

We have proved that the families of fully random hypergraphs $(H^k_{n,cn})_{c\in\mathbb{R}^+,n\in\mathbb{N}}$ and double hashing hypergraphs $(D^k_{n,cn})_{c\in\mathbb{R}^+,n\in\mathbb{N}}$ share an ℓ-orientability threshold, but the similarities go much further. This is hardly surprising since the families share a random weak limit

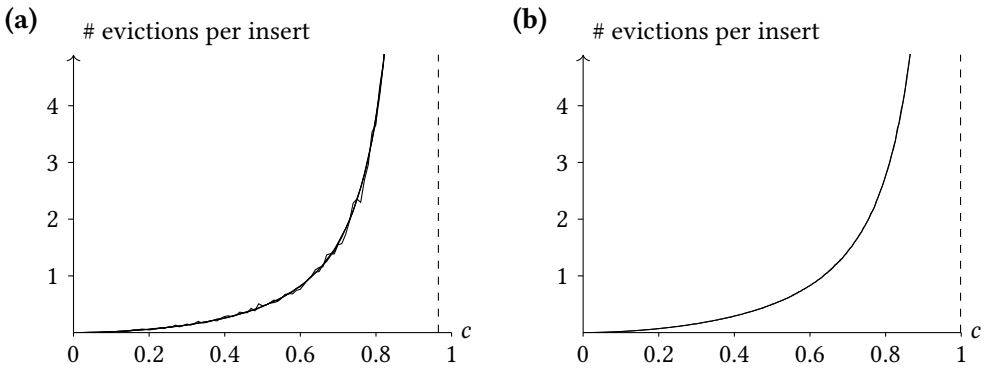


Figure 12.7 (a) For $k = \ell = 2$, $n \in \{10^4, 10^5, 10^6, 10^7\}$ and varying load c, we show the average number of evictions caused by random walk insertion into a cuckoo hash table using unaligned blocks. The curves for different n are almost identical.
(b) For $k = 4, \ell = 2$ and $n = 10^6$ we perform an analogous experiment for k-ary cuckoo hashing with buckets of size 2. A curve for fully random hashing and another for double hashing are visually indistinguishable.

as Leconte [Lec13] has shown. Mitzenmacher and Thaler [MT12] conjectured and partly proved that the families have equally sized ℓ-cores (for $\ell > 2$) and behave almost identical in experiments on cuckoo hashing with a stash.

We performed a simple experiment of our own, comparing the performance of random walk insertion using 4-ary cuckoo hashing with buckets of size 2 with and without double hashing. The setup corresponds to the experiment in Section 12.3.3. The resulting curves in Figure 12.7 **(b)** are visually indistinguishable, suggesting essentially identical performance.

13. Conclusion

This thesis studied implementations of dictionaries and related data types that offer constant worst-case access times, combined with small memory overhead and good cache efficiency. Most of the ideas and corresponding algorithms are simple. The main challenge was the mathematical analysis, including the derivation of load thresholds for peelability, solvability and orientability of underlying random hypergraphs. The results fall into three groups.

Results on Cuckoo Hashing. Theorems D, E and F provide theoretical justification for previously observed experimental findings concerning cuckoo hashing schemes.

- We proved that **unaligned blocks** yield higher load thresholds than aligned blocks of the same size, by determining the thresholds exactly. This confirms and strengthens existing results [DW07; LP09]. Whether or not unaligned blocks can lead to superior performance overall remains doubtful, however, see for instance [DW07, Figs. 7+8]. While aligned blocks call for a larger block size (say, 8 instead of 4), the performance impact of scanning the longer blocks is small. Moreover, the alignment can be chosen in agreement with the alignment of cache lines or the alignment required for SIMD instructions for benefits on modern CPU architectures. One could consider alignment to a non-trivial divisor of the block size, e.g. using blocks of size 8 aligned to multiples of 4. Such a compromise may benefit from the advantages of both approaches.
  When considering balls-into-bins settings detached from the application of cuckoo hashing, it could be of theoretical interest to study the effects of a geometric structure on the bins more generally. Imagine each ball chooses two random positions in a metric space and must be put into a bin that is *close* to one of the two positions. Theorem D considers this in the special case of offline load balancing for a *one-dimensional* arrangement of bins. Higher dimensions could be interesting, but would likely require completely different methods.
- Using **double hashing** in cuckoo tables improves running time and simplifies implementation in practice. We are happy to have completed the proof from [Lec13] that the load thresholds remain unaffected. We thereby extended the family of results saying that double hashing exhibits, in some setting, the same behaviour as fully random hashing. It is unfortunate that each proof so far required a specialised argument. Unifying the existing results and exposing a central reason why double hashing works in a wide range of cases would be a worthwhile project. This could also settle the question for ℓ-peelability thresholds for $2 \leq \ell < k$, where equality is still not proven, see [MT12].
- Lastly, we derived the load thresholds for **Dynamic Space Efficient Cuckoo Tables** [MSW19], settling the main theoretical question of the approach.

New Static Retrieval Data Structures. Theorems B and C examine a new kind of random matrix to be used in static retrieval data structures. In contrast to previous approaches

where the 1-entries are scattered, in the new approach, they are concentrated in **one block** or **two blocks** of $O(\log n)$ random bits per row. The resulting constructions profit from bit-parallel processing and good cache-efficiency.

The approach from Theorem B achieves constant query times and negligible overhead, though construction times are on the higher end. It seems that the approach from Theorem C balances the three performance characteristics most convincingly. This latter point is also supported by a very recent[1] implementation by Peter Dillinger [Dil20]. His Ribbon-Filter is an AMQ-Filter data structure building on the retrieval data structure from Theorem C1. However, the Ribbon-filter uses a different linear system solver, improving construction time by relevant constant factors, and a novel partitioning strategy that has, to the author's knowledge, not yet been analysed. Ribbon filters will soon be incorporated into the codebase of the widely used data storage engine RocksDB. We are happy to see our theoretical insights to be so quickly translated into practical progress.

Peeling at High Densities. The author's favourite result is Theorem A, which establishes that k-uniform spatially coupled hypergraphs have *peelability* thresholds that coincide with the *orientability* thresholds of fully random hypergraphs.

The construction can serve as a drop-in replacement in data structures that work by peeling random hypergraphs. Thereby the memory efficiency of these data structures improves. For the most part, the improvement comes "for free". Small caveats are a higher number of rounds taken by the peeling process, issues with downscaling to small input sizes (see Figure 12.2) and, by extension, issues with parallelisation.

We conjecture that the achieved thresholds are best possible, i.e. no family of k-uniform hypergraphs with i.i.d. random hyperedges has an ℓ-peelability threshold exceeding $c^*_{k,\ell}$. Indeed, even achieving ℓ-orientability beyond $c^*_{k,\ell}$ seems unlikely.

The construction exploits the phenomenon of "threshold saturation via spatial coupling" that was discovered in coding theory, and our proof borrows the powerful methods that were developed in the area.

What's next? Many results of this thesis rest on ideas and theorems imported from other areas, such as belief propagation (via [Lel12]), spatial coupling (via [KRU15]) and queuing theory (via [Coo90; EZB06]). In the early days of cuckoo hashing some of these fruitful connections have been dimly visible at best. It therefore makes sense, or so the author believes, to revisit some questions to which early works could not provide satisfying answers. One of these questions concerns the analysis of dynamic cuckoo hash tables, in particular the performance of random walk insertion. Despite some progress [FMM11; FPS13; FJ19], the question of whether the insertion time is constant in expectation is open (except for very small loads). Equipped with the versatile toolkit that has become available since, it may be worthwhile to take another stab at this. Moreover, when taking into account Theorem A, an analysis can exploit peelability without sacrificing memory efficiency, which may offer new avenues of attack as well.

[1] The relevant talk by Peter Dillinger was given a week prior to this thesis's defence. This new developement is therefore not reflected in the main part of this thesis.

Bibliography

[AB12a] Jean-Philippe Aumasson and Daniel J. Bernstein. *Hash-flooding DoS reloaded: attacks and defenses*. 2012. URL: https://131002.net/siphash/siphashdos_29c3_slides.pdf (visited on 27/02/2020) (cited on p. 55).

[AB12b] Jean-Philippe Aumasson and Daniel J. Bernstein. SipHash: A Fast Short-Input PRF. In: *Proc. 13th INDOCRYPT*. 2012, pp. 489–508. DOI: 10.1007/978-3-642-34931-7_28 (cited on p. 55).

[ACW16] Jyrki Alakuijala, Bill Cox and Jan Wassenberg. Fast keyed hash/pseudo-random function using SIMD multiply and permute. In: *CoRR* (2016). arXiv: 1612.06257 (cited on p. 55).

[ADR09] Martin Aumüller, Martin Dietzfelbinger and Michael Rink. Experimental Variations of a Theoretically Good Retrieval Data Structure. In: *Proc. 17th ESA*. 2009, pp. 742–751. DOI: 10.1007/978-3-642-04128-0_66 (cited on pp. 2, 38, 42).

[AH74] Alfred V. Aho and John E. Hopcroft. *The Design and Analysis of Computer Algorithms*. 1st. Addison-Wesley Longman Publishing Co., Inc., 1974. ISBN: 0201000296 (cited on p. 59).

[AL07] David Aldous and Russell Lyons. Processes on Unimodular Random Networks. In: *Electron. J. Probab.* 12 (2007), pp. 1454–1508. DOI: 10.1214/EJP.v12-463 (cited on p. 65).

[AMD08] AMD. *AMD Developer Guides, Manuals & ISA Documents*. 2008. URL: https://developer.amd.com/resources/developer-guides-manuals/ (visited on 27/02/2020) (cited on p. 52).

[ANS10] Yuriy Arbitman, Moni Naor and Gil Segev. Backyard Cuckoo Hashing: Constant Worst-Case Operations with a Succinct Representation. In: *Proc. 51th FOCS*. 2010, pp. 787–796. DOI: 10.1109/FOCS.2010.80 (cited on p. 63).

[App12] Austin Appleby. *SMHasher test suite designed to test the distribution, collision, and performance properties of non-cryptographic hash functions*. 2012. URL: https://github.com/aappleby/smhasher/wiki (visited on 27/02/2020) (cited on p. 55).

[Arl+70] V. Arlazarov, E. Dinic, M. Kronrod and I. Faradzev. On economical construction of the transitive closure of a directed Graph. In: *Dokl. Akad. Nauk SSSR, 194* (1970) (cited on p. 59).

[AS04] David Aldous and J. Michael Steele. The Objective Method: Probabilistic Combinatorial Optimization and Local Weak Convergence. In: *Probability on Discrete Structures*. Springer, 2004, pp. 1–72. ISBN: 978-3-662-09444-0. DOI: 10.1007/978-3-662-09444-0_1 (cited on pp. 6, 7, 68, 76).

[Bar09] Gregory V. Bard. In: *Algebraic Cryptanalysis*. Springer, 2009. Chap. The Method of Four Russians, pp. 133–158. ISBN: 978-0-387-88757-9. DOI: 10.1007/978-0-387-88757-9_9 (cited on p. 59).

[BBD09] Djamal Belazzougui, Fabiano C. Botelho and Martin Dietzfelbinger. Hash, Displace, and Compress. In: *Proc. 17th ESA*. 2009, pp. 682–693. DOI: 10.1007/978-3-642-04128-0_61 (cited on p. 43).

[Bel+10] Djamal Belazzougui, Paolo Boldi, Rasmus Pagh and Sebastiano Vigna. Fast Prefix Search in Little Space, with Applications. In: *Proc. 18th ESA*. 2010, pp. 427–438. DOI: 10.1007/978-3-642-15775-2_37 (cited on p. 29).

[Bel+14] Djamal Belazzougui, Paolo Boldi, Giuseppe Ottaviano, Rossano Venturini and Sebastiano Vigna. Cache-Oblivious Peeling of Random Hypergraphs. In: *Proc. DCC*. 2014, pp. 352–361. DOI: 10.1109/DCC.2014.48 (cited on pp. 43, 45, 143).

[Bey12] Stephan Beyer. Analysis of the Linear Probing Variant of Cuckoo Hashing. MA thesis. Technische Universität Ilmenau, 2012. URL: http://gso.gbv.de/DB=2.1/PPNSET?PPN=685166759 (cited on p. 25).

[BKZ05] Fabiano C. Botelho, Yoshiharu Kohayakawa and Nivio Ziviani. A Practical Minimal Perfect Hashing Method. In: *Proc. 4th WEA*. 2005, pp. 488–500. DOI: 10.1007/11427186_42 (cited on pp. 2, 40).

[Blo70] Burton H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. In: *Commun. ACM* 13.7 (1970), pp. 422–426. DOI: 10.1145/362686.362692 (cited on pp. 3, 29, 45).

[BM03] Andrei Z. Broder and Michael Mitzenmacher. Network Applications of Bloom Filters: A Survey. In: *Internet Mathematics* 1.4 (2003), pp. 485–509. DOI: 10.1080/15427951.2004.10129096 (cited on pp. 3, 29, 45).

[Bol+14] Paolo Boldi, Andrea Marino, Massimo Santini and Sebastiano Vigna. BUbiNG: Massive Crawling for the Masses. In: *Proc. 23rd WWW'14*. 2014, pp. 227–228. DOI: 10.1145/2567948.2577304 (cited on p. 140).

[Bor16] Charles Bordenave. *Lecture notes on random graphs and probabilistic combinatorial optimization*. 2016. URL: https://www.math.univ-toulouse.fr/~bordenave/coursRG.pdf (visited on 09/03/2020) (cited on pp. 6, 65–67, 71).

[Bot08] Fabiano C. Botelho. Near-Optimal Space Perfect Hashing Algorithms. PhD thesis. Federal University of Minas Gerais, 2008. URL: http://cmph.sourceforge.net/papers/thesis.pdf (cited on p. 43).

[BPZ07] Fabiano C. Botelho, Rasmus Pagh and Nivio Ziviani. Simple and Space-Efficient Minimal Perfect Hash Functions. In: *Proc. 10th WADS*. 2007, pp. 139–150. DOI: 10.1007/978-3-540-73951-7_13 (cited on p. 43).

[BPZ13] Fabiano C. Botelho, Rasmus Pagh and Nivio Ziviani. Practical Perfect Hashing in Nearly Optimal Space. In: *Inf. Syst.* 38.1 (2013), pp. 108–131. DOI: 10.1016/j.is.2012.06.002 (cited on pp. 2, 20, 38, 42–44, 140, 141).

[BS11] Itai Benjamini and Oded Schramm. Recurrence of Distributional Limits of Finite Planar Graphs. In: *Selected Works of Oded Schramm*. Springer New York, 2011, pp. 533–545. DOI: 10.1007/978-1-4419-9675-6_15 (cited on pp. 6, 25, 65).

[CC08] Denis Xavier Charles and Kumar Chellapilla. Bloomier Filters: A Second Look. In: *Proc. 16th ESA*. 2008, pp. 259–270. DOI: 10.1007/978-3-540-87744-8_22 (cited on pp. 3, 45).

[Cha+04] 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. 2004, pp. 30–39. URL: http://dl.acm.org/citation.cfm?id=982792.982797 (cited on p. 38).

[CHM92] Zbigniew J. Czech, George Havas and Bohdan S. Majewski. An optimal algorithm for generating minimal perfect hash functions. In: *Inf. Process. Lett.* 43.5 (1992), pp. 257–264. DOI: 10.1016/0020-0190(92)90220-P (cited on p. 43).

[CHM97] Zbigniew J. Czech, George Havas and Bohdan S. Majewski. Perfect Hashing. In: *Theor. Comput. Sci.* 182.1-2 (1997), pp. 1–143. DOI: 10.1016/S0304-3975(96)00146-6 (cited on p. 43).

[CLM85] Pedro Celis, Per-Åke Larson and J. Ian Munro. Robin Hood Hashing. In: *Proc. 26th FOCS*. 1985, pp. 281–288. DOI: 10.1109/SFCS.1985.48 (cited on pp. 62, 100).

[CMV13] Kai-Min Chung, Michael Mitzenmacher and Salil P. Vadhan. Why Simple Hash Functions Work: Exploiting the Entropy in a Data Stream. In: *Theory of Computing* 9 (2013), pp. 897–945. DOI: 10.4086/toc.2013.v009a030 (cited on p. 54).

[Col20] Yann Collet. *xxHash - Extremely fast hash algorithm*. 2020. URL: https://github.com/Cyan4973/xxHash (visited on 24/03/2020) (cited on p. 142).

[Coo00] Colin Cooper. On the rank of random matrices. In: *Random Structures & Algorithms* 16.2 (2000), pp. 209–232. DOI: 10.1002/(SICI)1098-2418(200003)16:2<209::AID-RSA6>3.0.CO;2-1 (cited on pp. 19, 20).

[Coo04] Colin Cooper. The Cores of Random Hypergraphs with a Given Degree Sequence. In: *Random Struct. Algorithms* 25.4 (2004), pp. 353–375. DOI: 10.1002/rsa.20040 (cited on pp. 14, 26, 76).

[Coo90] Robert B. Cooper. *Introduction to queueing theory*. 3rd. George Washington University, 1990. ISBN: 978-0941893039 (cited on pp. 7, 104, 152).

[Cor+09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. ISBN: 978-0-262-03384-8. URL: http://mitpress.mit.edu/books/introduction-algorithms (cited on pp. 27, 53, 62, 98).

[CPS19] David Clayton, Christopher Patton and Thomas Shrimpton. Probabilistic Data Structures in Adversarial Environments. In: *Proc. 26th CCS*. 2019, pp. 1317–1334. DOI: 10.1145/3319535.3354235 (cited on p. 55).

[CSW07] Julie A. 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*. 2007, pp. 469–476. URL: http://dl.acm.org/citation.cfm?id=1283383.1283433 (cited on pp. 2, 14, 24, 26, 32, 33, 148).

[CW03] Scott A. Crosby and Dan S. Wallach. Denial of Service via Algorithmic Complexity Attacks. In: *Proc. 12th USENIX*. 2003. URL: https://www.usenix.org/conference/12th-usenix-security-symposium/denial-service-algorithmic-complexity-attacks (cited on p. 55).

[CW79] Larry Carter and Mark N. Wegman. Universal Classes of Hash Functions. In: *J. Comput. Syst. Sci.* 18.2 (1979), pp. 143–154. DOI: 10.1016/0022-0000(79)90044-8 (cited on p. 54).

[Die+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)*. 2010, pp. 213–225. DOI: 10.1007/978-3-642-14165-2_19 (cited on pp. 14, 18, 20, 24, 26, 76, 148).

[Die+97] Martin Dietzfelbinger, Torben Hagerup, Jyrki Katajainen and Martti Penttonen. A Reliable Randomized Algorithm for the Closest-Pair Problem. In: *J. Algorithms* 25.1 (1997), pp. 19–51. DOI: 10.1006/jagm.1997.0873 (cited on p. 54).

[Die07] Martin Dietzfelbinger. Design Strategies for Minimal Perfect Hash Functions. In: *Proc. 4th SAGA*. 2007, pp. 2–17. DOI: 10.1007/978-3-540-74871-7_2 (cited on p. 43).

[Dil20] Peter C. Dillinger. *RIBBON: A practical and near-optimal static Bloom alternative for RocksDB*. Oct. 2020. URL: https://www.youtube.com/watch?v=XfwxUBL8xT8&t=1h16m10s (visited on 01/11/2020) (cited on p. 152).

[DM02] Olivier Dubois and Jacques Mandler. The 3-XORSAT Threshold. In: *Proc. 43rd FOCS*. 2002, pp. 769–778. DOI: 10.1109/SFCS.2002.1182002 (cited on pp. 14, 18).

[DMR11] Martin Dietzfelbinger, Michael Mitzenmacher and Michael Rink. Cuckoo Hashing with Pages. In: *Proc. 19th ESA*. 2011, pp. 615–627. DOI: 10.1007/978-3-642-23719-5_52 (cited on pp. 2, 32, 33).

[DMV04] Luc Devroye, Pat Morin and Alfredo Viola. On Worst-Case Robin Hood Hashing. In: *SIAM J. Comput.* 33.4 (2004), pp. 923–936. DOI: 10.1137/S0097539702403372 (cited on pp. 63, 100).

[DP08] Martin Dietzfelbinger and Rasmus Pagh. Succinct Data Structures for Retrieval and Approximate Membership (Extended Abstract). In: *Proc. 35th ICALP (1)*. 2008, pp. 385–396. DOI: 10.1007/978-3-540-70575-8_32 (cited on pp. 2, 3, 20, 29, 38, 42, 45).

[DR09] Martin Dietzfelbinger and Michael Rink. Applications of a Splitting Trick. In: *Proc. 36th ICALP (1)*. 2009, pp. 354–365. DOI: 10.1007/978-3-642-02927-1_30 (cited on pp. 2, 40, 54, 55).

[DW07] Martin Dietzfelbinger and Christoph Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. In: *Theor. Comput. Sci.* 380.1-2 (2007), pp. 47–68. DOI: 10.1016/j.tcs.2007.02.054 (cited on pp. 2, 6, 21, 32, 151).

[DW19a] Martin Dietzfelbinger and Stefan Walzer. Constant-Time Retrieval with $O(\log m)$ Extra Bits. In: *Proc. 36th STACS*. 2019, 24:1–24:16. ISBN: 978-3-95977-100-9. DOI: 10.4230/LIPIcs.STACS.2019.24 (cited on p. 8).

[DW19b] Martin Dietzfelbinger and Stefan Walzer. Dense Peelable Random Uniform Hypergraphs. In: *Proc. 27th ESA*. 2019, 38:1–38:16. DOI: 10.4230/LIPIcs.ESA.2019.38 (cited on pp. iv, 8, 75).

[DW19c] Martin Dietzfelbinger and Stefan Walzer. Efficient Gauss Elimination for Near-Quadratic Matrices with One Short Random Block per Row, with Applications. In: *Proc. 27th ESA*. 2019, 39:1–39:18. DOI: 10.4230/LIPIcs.ESA.2019.39 (cited on pp. iv, 8).

[EG11] David Eppstein and Michael T. Goodrich. Straggler Identification in Round-Trip Data Streams via Newton's Identities and Invertible Bloom Filters. In: *IEEE Trans. Knowl. Data Eng.* 23.2 (2011), pp. 297–306. DOI: 10.1109/TKDE.2010.132 (cited on pp. 30, 47, 48).

[EGV20] Emmanuel Esposito, Thomas Mueller Graf and Sebastiano Vigna. RecSplit: Minimal Perfect Hashing via Recursive Splitting. In: *Proc. ALENEX20*. 2020, pp. 175–185. DOI: 10.1137/1.9781611976007.14 (cited on pp. 2, 40, 41, 43, 44).

[Epp+11] David Eppstein, Michael T. Goodrich, Frank Uyeda and George Varghese. What's the difference?: efficient set reconciliation without prior context. In: *Proc. SIGCOMM '11*. 2011, pp. 218–229. DOI: 10.1145/2018436.2018462 (cited on pp. 47, 48).

[Epp16] David Eppstein. Cuckoo Filter: Simplification and Analysis. In: *Proc. 15th SWAT*. 2016, 8:1–8:12. DOI: 10.4230/LIPIcs.SWAT.2016.8 (cited on p. 46).

[ER60] Paul Erdős and Alfréd Rényi. On the Evolution of Random Graphs. In: *Publ. Math. Inst. Hung. Acad. Sci.* (1960). URL: http://www.renyi.hu/~p_erdos/1961-15.pdf (cited on p. 24).

[EZB06] Regina Egorova, Bert Zwart and Onno Boxma. Sojourn Time Tails in the M/D/1 Processor Sharing Queue. In: *Probability in the Engineering and Informational Sciences* 20 (2006), pp. 429–446. DOI: 10.1017/S0269964806060268 (cited on pp. 7, 104, 152).

[FAK13] Bin Fan, David G. Andersen and Michael Kaminsky. Cuckoo Filter: Better Than Bloom. In: *;login:* 38.4 (2013). URL: https://www.usenix.org/publications/login/august-2013-volume-38-number-4/cuckoo-filter-better-bloom (cited on pp. 3, 31, 46).

[Fan+14] Bin Fan, David G. Andersen, Michael Kaminsky and Michael Mitzenmacher. Cuckoo Filter: Practically Better Than Bloom. In: *Proc. 10th CoNEXT*. 2014, pp. 75–88. DOI: 10.1145/2674005.2674994 (cited on p. 46).

[FB99] Ehud Friedgut and Jean Bourgain. Sharp Thresholds of Graph Properties, and the k-SAT Problem. In: *Journal of the American Mathematical Society* 12.4 (1999), pp. 1017–1054. ISSN: 08940347, 10886834. URL: http://www.jstor.org/stable/2646096 (cited on p. 15).

[FJ19] Alan M. Frieze and Tony Johansson. On the insertion time of random walk cuckoo hashing. In: *Random Struct. Algorithms* 54.4 (2019), pp. 721–729. DOI: 10.1002/rsa.20808 (cited on pp. 34, 152).

[FKP11] Nikolaos Fountoulakis, Megha Khosla and Konstantinos Panagiotou. The Multiple-Orientability Thresholds for Random Hypergraphs. In: *Proc. 22nd SODA*. 2011, pp. 1222–1236. URL: https://dl.acm.org/doi/10.5555/2133036.2133129 (cited on p. 15).

[FKP16] Nikolaos Fountoulakis, Megha Khosla and Konstantinos Panagiotou. The Multiple-Orientability Thresholds for Random Hypergraphs. In: *Combinatorics, Probability & Computing* 25.6 (2016), pp. 870–908. DOI: 10.1017/S0963548315000334 (cited on pp. 14, 15, 24, 26, 32, 82, 83, 86, 125).

[FKS84] Michael L. Fredman, János Komlós and Endre Szemerédi. Storing a Sparse Table with $O(1)$ Worst Case Access Time. In: *J. ACM* 31.3 (1984), pp. 538–544. DOI: 10.1145/828.1884 (cited on pp. 43, 44).

[FM12] Alan M. Frieze and Páll Melsted. Maximum Matchings in Random Bipartite Graphs and the Space Utilization of Cuckoo Hash Tables. In: *Random Struct. Algorithms* 41.3 (2012), pp. 334–364. DOI: 10.1002/rsa.20427 (cited on pp. 24, 26).

[FMM11] Alan M. Frieze, Páll Melsted and Michael Mitzenmacher. An Analysis of Random-Walk Cuckoo Hashing. In: *SIAM J. Comput.* 40.2 (2011), pp. 291–308. DOI: 10.1137/090770928 (cited on pp. 34, 152).

[Fot+05] Dimitris Fotakis, Rasmus Pagh, Peter Sanders and Paul G. Spirakis. Space Efficient Hash Tables with Worst Case Constant Access Time. In: *Theory Comput. Syst.* 38.2 (2005), pp. 229–248. DOI: 10.1007/s00224-004-1195-x (cited on pp. 2, 32, 34).

[FP10] Nikolaos Fountoulakis and Konstantinos Panagiotou. Orientability of Random Hypergraphs and the Power of Multiple Choices. In: *Proc. 37th ICALP (1)*. 2010, pp. 348–359. DOI: 10.1007/978-3-642-14165-2_30 (cited on pp. 14, 24).

[FP12] Nikolaos Fountoulakis and Konstantinos Panagiotou. Sharp Load Thresholds for Cuckoo Hashing. In: *Random Struct. Algorithms* 41.3 (2012), pp. 306–333. DOI: 10.1002/rsa.20426 (cited on pp. 24, 26, 83).

[FPS13] Nikolaos Fountoulakis, Konstantinos Panagiotou and Angelika Steger. On the Insertion Time of Cuckoo Hashing. In: *SIAM J. Comput.* 42.6 (2013), pp. 2156–2181. DOI: 10.1137/100797503 (cited on p. 152).

[FPS16] Guy Feigenblat, Ely Porat and Ariel Shiftan. Linear Time Succinct Indexable Dictionary Construction with Applications. In: *Proc. DCC16*. 2016, pp. 13–22. DOI: 10.1109/DCC.2016.70 (cited on p. 44).

[FR07] Daniel Fernholz and Vijaya Ramachandran. The k-orientability Thresholds for $G_{n,p}$. In: *Proc. 18th SODA*. 2007, pp. 459–468. URL: http://dl.acm.org/citation.cfm?id=1283383.1283432 (cited on pp. 2, 14, 24, 26, 32, 33, 148).

[FZ99] Alberto J. Feltström and Kamil Sh. Zigangirov. Time-varying periodic convolutional codes with low-density parity-check matrix. In: *IEEE Trans. Information Theory* 45.6 (1999), pp. 2181–2191. DOI: 10.1109/18.782171 (cited on p. 75).

[GKL15] Thomas Gerbet, Amrit Kumar and Cédric Lauradoux. The Power of Evil Choices in Bloom Filters. In: *Proc. 45th DSN*. 2015, pp. 101–112. DOI: 10.1109/DSN.2015.21 (cited on p. 55).

[GL19] Thomas Mueller Graf and Daniel Lemire. Xor Filters: Faster and Smaller Than Bloom and Cuckoo Filters. In: *CoRR* (2019). arXiv: 1912.08258 (cited on pp. 3, 46).

[GM11] Michael T. Goodrich and Michael Mitzenmacher. Invertible bloom lookup tables. In: *Proc. 49st Allerton*. 2011, pp. 792–799. DOI: 10.1109/Allerton.2011.6120248 (cited on pp. 47, 48).

[GMU12] Andrei Giurgiu, Nicolas Macris and Rüdiger L. Urbanke. How to prove the Maxwell conjecture via spatial coupling - A proof of concept. In: *Proc. ISIT*. 2012, pp. 458–462. DOI: 10.1109/ISIT.2012.6284230 (cited on p. 17).

[Goo18] Google. *Swiss Tables and absl::Hash*. 2018. URL: https://abseil.io/blog/20180927-swisstables (visited on 03/03/2020) (cited on p. 63).

[GOV16] Marco Genuzio, Giuseppe Ottaviano and Sebastiano Vigna. Fast Scalable Construction of (Minimal Perfect Hash) Functions. In: *Proc. 15th SEA*. 2016, pp. 339–

352. DOI: 10.1007/978-3-319-38851-9_23 (cited on pp. 2, 38, 40, 42, 43, 56, 57, 105, 139–141).

[GOV20] Marco Genuzio, Giuseppe Ottaviano and Sebastiano Vigna. Fast scalable construction of ([compressed] static | minimal perfect hash) functions. In: *Information and Computation* (2020). ISSN: 0890-5401. DOI: 10.1016/j.ic.2020.104517 (cited on pp. 2, 40, 43, 45).

[Gri80] Geoffrey R. Grimmett. Random labelled trees and their branching networks. In: *Journal of the Aust MS* 30.2 (1980), pp. 229–237. DOI: 10.1017/S1446788700016517 (cited on p. 67).

[GS78] Leonidas J. Guibas and Endre Szemerédi. The Analysis of Double Hashing. In: *J. Comput. Syst. Sci.* 16.2 (1978), pp. 226–274. DOI: 10.1016/0022-0000(78)90046-6 (cited on p. 53).

[GW10] Pu Gao and Nicholas C. Wormald. Load Balancing and Orientability Thresholds for Random Hypergraphs. In: *Proc. 42nd STOC*. 2010, pp. 97–104. DOI: 10.1145/1806689.1806705 (cited on pp. 14, 24).

[GW15] Pu Gao and Nicholas C. Wormald. Orientability Thresholds for Random Hypergraphs. In: *Combinatorics, Probability & Computing* 24.5 (2015), pp. 774–824. DOI: 10.1017/S096354831400073X (cited on p. 24).

[Hal35] Philip Hall. On Representatives of Subsets. In: *Journal of the London Mathematical Society* (1935). DOI: 10.1112/jlms/s1-10.37.26 (cited on p. 14).

[HK73] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. In: *SIAM J. Comput.* 2.4 (1973), pp. 225–231. DOI: 10.1137/0202019 (cited on p. 33).

[HKP09] Jóhannes B. Hreinsson, Morten Krøyer and Rasmus Pagh. Storing a Compressed Function with Constant Time Access. In: *Proc. 17th ESA*. 2009, pp. 730–741. DOI: 10.1007/978-3-642-04128-0_65 (cited on p. 43).

[HMU13] S. Hamed Hassani, Nicolas Macris and Rüdiger L. Urbanke. The space of solutions of coupled XORSAT formulae. In: *Proc. ISIT*. 2013, pp. 2453–2457. DOI: 10.1109/ISIT.2013.6620667 (cited on p. 17).

[HP12] John L. Hennessy and David A. Patterson. *Computer Architecture - A Quantitative Approach, 5th Edition*. Morgan Kaufmann, 2012. ISBN: 978-0-12-383872-8 (cited on p. 31).

[HT01] Torben Hagerup and Torsten Tholey. Efficient minimal perfect hashing in nearly minimal space. In: *Proc. 18st STACS*. 2001, pp. 317–326. DOI: 10.1007/3-540-44693-1_28 (cited on pp. 43, 44).

[Int07] Intel. *Intel 64 and IA-32 Architectures Software Developer Manuals*. 2007. URL: https://software.intel.com/en-us/articles/intel-sdm (visited on 27/02/2020) (cited on p. 52).

[Jan05] Svante Janson. Individual displacements for linear probing hashing with different insertion policies. In: *ACM Trans. Algorithms* 1.2 (2005), pp. 177–213. DOI: 10.1145/1103963.1103964 (cited on p. 100).

[Jan08] Svante Janson. Individual Displacements in Hashing with Coalesced Chains. In: *Comb. Probab. Comput.* 17.6 (2008), pp. 799–814. DOI: 10.1017/S0963548308009395 (cited on p. 102).

[JL07] Svante Janson and Malwina J. Luczak. A simple solution to the k-core problem. In: *Random Struct. Algorithms* 30.1-2 (2007), pp. 50–62. DOI: 10.1002/rsa.20147 (cited on pp. 14, 26, 83).

[JV16] Svante Janson and Alfredo Viola. A Unified Approach to Linear Probing Hashing with Buckets. In: *Algorithmica* 75.4 (2016), pp. 724–781. DOI: 10.1007/s00453-015-0111-x (cited on p. 100).

[KA19] Megha Khosla and Avishek Anand. A Faster Algorithm for Cuckoo Insertion and Bipartite Matching in Large Graphs. In: *Algorithmica* 81.9 (2019), pp. 3707–3724. DOI: 10.1007/s00453-019-00595-4 (cited on pp. 33, 148).

[Ken53] David G. Kendall. Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain. In: *Ann. Math. Statist.* 24.3 (Sept. 1953), pp. 338–354. DOI: 10.1214/aoms/1177728975 (cited on p. 104).

[Kho13] Megha Khosla. Balls into Bins Made Faster. In: *Proc. 21st ESA*. 2013, pp. 601–612. DOI: 10.1007/978-3-642-40450-4_51 (cited on pp. 33, 140, 148).

[Kim06] Jeong Han Kim. Poisson cloning model for random graphs. In: *Proc. ICM, Vol. III*. 2006, pp. 873–898. URL: https://www.mathunion.org/fileadmin/ICM/Proceedings/ICM2006.3/ICM2006.3.ocr.pdf (cited on p. 71).

[KM08] Adam Kirsch and Michael Mitzenmacher. Less hashing, same performance: Building a better Bloom filter. In: *Random Struct. Algorithms* 33.2 (2008), pp. 187–218. URL: https://dl.acm.org/doi/10.5555/1400123.1400125 (cited on p. 53).

[Knu98] Donald E. Knuth. *Sorting and Searching*. 2nd. Addison-Wesley, 1998. URL: http://dl.acm.org/citation.cfm?id=280635 (cited on p. 53).

[KR87] Richard M. Karp and Michael O. Rabin. Efficient Randomized Pattern-Matching Algorithms. In: *IBM Journal of Research and Development* 31.2 (1987), pp. 249–260. DOI: 10.1147/rd.312.0249 (cited on p. 31).

[KRU10] Shrinivas Kudekar, Tom Richardson and Rüdiger L. Urbanke. Threshold saturation via spatial coupling: Why convolutional LDPC ensembles perform so well over the BEC. In: *Proc. ISIT*. 2010, pp. 684–688. DOI: 10.1109/ISIT.2010.5513587 (cited on pp. 7, 75).

[KRU11] Shrinivas Kudekar, Tom Richardson and Rüdiger L. Urbanke. Threshold Saturation via Spatial Coupling: Why Convolutional LDPC Ensembles Perform So Well over the BEC. In: *IEEE Trans. Inf. Theory* 57.2 (2011), pp. 803–834. DOI: 10.1109/TIT.2010.2095072 (cited on p. 7).

[KRU13] Shrinivas Kudekar, Tom Richardson and Rüdiger L. Urbanke. Spatially Coupled Ensembles Universally Achieve Capacity Under Belief Propagation. In: *IEEE Trans. Information Theory* 59.12 (2013), pp. 7761–7813. DOI: 10.1109/TIT.2013.2280915 (cited on pp. 17, 75).

[KRU15] Shrinivas Kudekar, Tom Richardson and Rüdiger L. Urbanke. Wave-Like Solutions of General 1-D Spatially Coupled Systems. In: *IEEE Trans. Inf. Theory* 61.8 (2015), pp. 4117–4157. DOI: 10.1109/TIT.2015.2438870 (cited on pp. 7, 17, 75, 79, 80, 82, 83, 152).

[Krz+12] Florent Krzakala, Marc Mézard, François Sausset, Yifan Sun and Lenka Zdeborová. Statistical-Physics-Based Reconstruction in Compressed Sensing. In: *Phys. Rev. X* 2 (2 2012) (cited on p. 75).

[Lec13] Mathieu Leconte. Double hashing thresholds via local weak convergence. In: *Proc. 51st Allerton*. 2013, pp. 131–137. DOI: 10.1109/Allerton.2013.6736515 (cited on pp. 6, 23–25, 35, 53, 65, 71, 125, 150, 151).

[Lel12] Marc Lelarge. A New Approach to the Orientation of Random Hypergraphs. In: *Proc. 23rd SODA*. 2012, pp. 251–264. DOI: 10.1137/1.9781611973099.23 (cited on pp. 6, 7, 14, 15, 24, 25, 32, 65, 70, 83, 85, 86, 112, 117, 125, 126, 135, 152).

[Lim+17] Antoine Limasset, Guillaume Rizk, Rayan Chikhi and Pierre Peterlongo. Fast and Scalable Minimal Perfect Hashing for Massive Key Sets. In: *Proc. 16th SEA*. 2017, 25:1–25:16. ISBN: 978-3-95977-036-1. DOI: 10.4230/LIPIcs.SEA.2017.25 (cited on p. 43).

[LLM13] Mathieu Leconte, Marc Lelarge and Laurent Massoulié. Convergence of Multivariate Belief Propagation, with Applications to Cuckoo Hashing and Load Balancing. In: *Proc. 24th SODA*. 2013, pp. 35–46. URL: http://dl.acm.org/citation.cfm?id=2627817.2627820 (cited on pp. 6, 24, 25, 65, 68, 70, 110–112, 117).

[LM93] George S. Lueker and Mariko Molodowitch. More analysis of double hashing. In: *Combinatorica* 13.1 (1993), pp. 83–96. DOI: 10.1007/BF01202791 (cited on p. 53).

[LN93] Richard J. Lipton and Jeffrey F. Naughton. Clocked Adversaries for Hashing. In: *Algorithmica* 9.3 (1993), pp. 239–252. DOI: 10.1007/BF01190898 (cited on p. 55).

[LO90] Brian A. LaMacchia and Andrew M. Odlyzko. Solving large sparse linear systems over finite fields. In: *Proc. 10th CRYPTO*. 1990, pp. 109–133. DOI: 10.1007/3-540-38424-3_8 (cited on pp. 56, 57).

[LP09] Eric Lehman and Rina Panigrahy. 3.5-Way Cuckoo Hashing for the Price of 2-and-a-Bit. In: *Proc. 17th ESA*. 2009, pp. 671–681. DOI: 10.1007/978-3-642-04128-0_60 (cited on pp. 2, 6, 21, 24, 25, 32, 151).

[LP14] Po-Shen Loh and Rasmus Pagh. Thresholds for Extreme Orientability. In: *Algorithmica* 69.3 (2014), pp. 522–539. DOI: 10.1007/s00453-013-9749-4 (cited on p. 24).

[Lub+01] Michael Luby, Michael Mitzenmacher, M. Amin Shokrollahi and Daniel A. Spielman. Efficient Erasure Correcting Codes. In: *IEEE Trans. Inf. Theory* 47.2 (2001), pp. 569–584. DOI: 10.1109/18.910575 (cited on pp. 17, 140, 141).

[Lub+97] Michael Luby, Michael Mitzenmacher, M. Amin Shokrollahi, Daniel A. Spielman and Volker Stemann. Practical Loss-Resilient Codes. In: *Proc. 29th STOC*. 1997, pp. 150–159. DOI: 10.1145/258533.258573 (cited on p. 42).

[Luc91] Tomasz Luczak. Size and connectivity of the k-core of a random graph. In: *Discrete Mathematics* 91.1 (1991), pp. 61–68. DOI: 10.1016/0012-365X(91)90162-U (cited on p. 83).

[Luo+19] Lailong Luo, Deke Guo, Richard T. B. Ma, Ori Rottenstreich and Xueshan Luo. Optimizing Bloom Filter: Challenges, Solutions, and Comparisons. In: *IEEE Communications Surveys and Tutorials* 21.2 (2019), pp. 1912–1949. DOI: 10.1109/COMST.2018.2889329 (cited on p. 45).

[Maj+96] Bohdan S. Majewski, Nicholas C. Wormald, George Havas and Zbigniew J. Czech. A Family of Perfect Hashing Methods. In: *Comput. J.* 39.6 (1996), pp. 547–554. DOI: 10.1093/comjnl/39.6.547 (cited on pp. 20, 40, 42, 43).

[Mau19] Jens Maurer. *P0553R4: Bit operations*. 2019. URL: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2019/p0553r4.html (visited on 27/02/2020) (cited on p. 52).

[Mit09] Michael Mitzenmacher. Some Open Questions Related to Cuckoo Hashing. In: *Proc. 17th ESA*. 2009, pp. 1–10. DOI: 10.1007/978-3-642-04128-0_1 (cited on p. 34).

[Mit14] Michael Mitzenmacher. Balanced allocations and double hashing. In: *Proc. 26th ACM SPAA*. ACM. 2014, pp. 331–342. DOI: 10.1145/2612669.2612684 (cited on p. 53).

[Mit16] Michael Mitzenmacher. More Analysis of Double Hashing for Balanced Allocations. In: *Proc. ANALCO16*. 2016, pp. 1–9. DOI: 10.1137/1.9781611974324.1 (cited on p. 53).

[Mit91] Michael Mitzenmacher. The Power of Two Choices in Randomized Load Balancing. PhD thesis. Harvard University, 1991. URL: http://www.eecs.harvard.edu/~michaelm/postscripts/mythesis.pdf (cited on p. 30).

[MM09] Marc Mezard and Andrea Montanari. *Information, Physics, and Computation*. USA: Oxford University Press, Inc., 2009. ISBN: 019857083X (cited on pp. 26, 76).

[MNS11] Ilya Mironov, Moni Naor and Gil Segev. Sketching in Adversarial Environments. In: *SIAM J. Comput.* 40.6 (2011), pp. 1845–1870. DOI: 10.1137/080733772 (cited on p. 55).

[Mol04] Michael Molloy. The pure literal rule threshold and cores in random hypergraphs. In: *Proc. 15th SODA*. 2004, pp. 672–681. URL: http://dl.acm.org/citation.cfm?id=982792.982896 (cited on p. 76).

[Mol05] Michael Molloy. Cores in random hypergraphs and Boolean formulas. In: *Random Struct. Algorithms* 27.1 (2005), pp. 124–135. DOI: 10.1002/rsa.20061 (cited on pp. 14, 15, 17, 26, 76, 83, 85).

[MP80] William J. Masek and Michael S. Paterson. A faster algorithm computing string edit distances. In: *J. Comput. Syst. Sci.* 20.1 (1980), pp. 18–31. DOI: 10.1016/0022-0000(80)90002-1 (cited on p. 59).

[MPW18] Michael Mitzenmacher, Konstantinos Panagiotou and Stefan Walzer. Load Thresholds for Cuckoo Hashing with Double Hashing. In: *Proc. 16th SWAT*. 2018, 29:1–29:9. DOI: 10.4230/LIPIcs.SWAT.2018.29 (cited on pp. iv, 8, 83).

[MS08] Kurt Mehlhorn and Peter Sanders. *Algorithms and Data Structures: The Basic Toolbox*. Springer, 2008. ISBN: 978-3-540-77977-3. DOI: 10.1007/978-3-540-77978-0 (cited on p. 51).

[MS17] Tobias Maier and Peter Sanders. Dynamic Space Efficient Hashing. In: *Proc. 25th ESA*. 2017, 58:1–58:14. DOI: 10.4230/LIPIcs.ESA.2017.58 (cited on pp. 6, 23, 34, 36).

[MSW19] Tobias Maier, Peter Sanders and Stefan Walzer. Dynamic Space Efficient Hashing. In: *Algorithmica* 81.8 (2019), pp. 3162–3185. DOI: 10.1007/s00453-019-00572-x (cited on pp. iv, 8, 34, 36, 37, 133, 140, 151).

[MT12] Michael Mitzenmacher and Justin Thaler. Peeling Arguments and Double Hashing. In: *Proc. 50th Allerton*. 2012, pp. 1118–1125. DOI: 10.1109/Allerton.2012.6483344 (cited on pp. 6, 35, 53, 150, 151).

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005. URL: http://dl.acm.org/citation.cfm?id=1076315 (cited on pp. 102, 103).

[MU17] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. 2nd. New York, NY, USA: Cambridge University Press, 2017. ISBN: 978-1107154889 (cited on p. 85).

[Mül+14] Ingo Müller, Peter Sanders, Robert Schulze and Wei Zhou. Retrieval and Perfect Hashing Using Fingerprinting. In: *Experimental Algorithms*. Ed. by Joachim Gudmundsson and Jyrki Katajainen. 2014, pp. 138–149. DOI: 10.1007/978-3-319-07959-2_12 (cited on pp. 31, 42).

[MV12a] Michael Mitzenmacher and George Varghese. Biff (Bloom Filter) Codes: Fast Error Correction for Large Data Sets. In: *Proc. ISIT*. 2012, pp. 483–487. DOI: 10.1109/ISIT.2012.6284236 (cited on p. 61).

[MV12b] Michael Mitzenmacher and George Varghese. The complexity of object reconciliation, and open problems related to set difference and coding. In: *Proc. 50th Allerton*. 2012, pp. 1126–1132. DOI: 10.1109/Allerton.2012.6483345 (cited on p. 61).

[NY19] Moni Naor and Eylon Yogev. Bloom Filters in Adversarial Environments. In: *ACM Trans. Algorithms* 15.3 (2019), 35:1–35:30. DOI: 10.1145/3306193 (cited on p. 55).

[OS07] Daisuke Okanohara and Kunihiko Sadakane. Practical Entropy-Compressed Rank/Select Dictionary. In: *Proc. ALENEX07*. 2007. DOI: 10.1137/1.9781611972870.6 (cited on p. 44).

[Pag01] Rasmus Pagh. Low Redundancy in Static Dictionaries with Constant Query Time. In: *SIAM J. Comput.* 31.2 (2001), pp. 353–363. DOI: 10.1137/S0097539700369909 (cited on p. 44).

[Por09] Ely Porat. An Optimal Bloom Filter Replacement Based on Matrix Solving. In: *Proc. 4th CSR*. 2009, pp. 263–273. DOI: 10.1007/978-3-642-03351-3_25 (cited on pp. 2, 38, 40, 42).

[PR04] Rasmus Pagh and Flemming F. Rodler. Cuckoo Hashing. In: *J. Algorithms* 51.2 (2004), pp. 122–144. DOI: 10.1016/j.jalgor.2003.12.002 (cited on pp. 2, 24, 26, 32).

[PS12] Ely Porat and Bar Shalem. A Cuckoo Hashing Variant with Improved Memory Utilization and Insertion Time. In: *Proc. 22nd DCC*. 2012, pp. 347–356. DOI: 10.1109/DCC.2012.41 (cited on pp. 2, 32).

[PS16] Boris Pittel and Gregory B. Sorkin. The Satisfiability Threshold for k-XORSAT. In: *Combinatorics, Probability & Computing* 25.2 (2016), pp. 236–268. DOI: 10.1017/S0963548315000097 (cited on pp. 14, 18, 20).

[PSW96] Boris Pittel, Joel Spencer and Nicholas C. Wormald. Sudden Emergence of a Giant k-Core in a Random Graph. In: *J. Comb. Theory, Ser. B* 67.1 (1996), pp. 111–151. DOI: 10.1006/jctb.1996.0036 (cited on pp. 26, 75).

[Rin13] Michael Rink. Mixed Hypergraphs for Linear-Time Construction of Denser Hashing-Based Data Structures. In: *Proc. 39th SOFSEM*. 2013, pp. 356–368. DOI: 10.1007/978-3-642-35843-2_31 (cited on pp. 17, 42).

[Rin15] Michael Rink. Thresholds for Matchings in Random Bipartite Graphs with Applications to Hashing-Based Data Structures. PhD thesis. Technische Universität Ilmenau, Germany, 2015. URL: http://www.db-thueringen.de/servlets/DocumentServlet?id=25985 (cited on pp. iv, 33, 43, 54).

[RRS07] Rajeev Raman, Venkatesh Raman and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. In: *ACM Trans. Algorithms* 3.4 (2007), p. 43. DOI: 10.1145/1290672.1290680 (cited on p. 44).

[RU08] Thomas J. Richardson and Rüdiger L. Urbanke. *Modern Coding Theory*. Cambridge University Press, 2008. ISBN: 978-0-521-85229-6. DOI: 10.1017/cbo9780511791338 (cited on p. 61).

[San+19] Peter Sanders, Kurt Mehlhorn, Martin Dietzfelbinger and Roman Dementiev. *Sequential and Parallel Algorithms and Data Structures - The Basic Toolbox*. Springer, 2019. ISBN: 978-3-030-25208-3. DOI: 10.1007/978-3-030-25209-0 (cited on p. 27).

[Sci14] SciShow. *How to Supercool Water: A SciShow Experiment*. 2014. URL: https://www.youtube.com/watch?v=NMSxuORKynI (visited on 25/04/2020) (cited on p. 7).

[Sed03] Robert Sedgewick. *Algorithms in Java Parts 1–4: Fundamentals, Data Structures, Sorting, Searching*. 3rd. Addison Wesley, 2003 (cited on p. 62).

[SS90] Jeanette P. Schmidt and Alan Siegel. The Analysis of Closed Hashing under Limited Randomness (Extended Abstract). In: *Proc. 22nd Annual ACM Symposium on Theory of Computing*. 1990, pp. 224–234. DOI: 10.1145/100216.100245 (cited on p. 53).

[Tan81] R. Michael Tanner. A recursive approach to low complexity codes. In: *IEEE Trans. Information Theory* 27.5 (1981), pp. 533–547. DOI: 10.1109/TIT.1981.1056404 (cited on p. 61).

[Tho15] Mikkel Thorup. High Speed Hashing for Integers and Strings. In: *CoRR* (2015). arXiv: 1504.06804 (cited on p. 54).

[TTK12] Keigo Takeuchi, Toshiyuki Tanaka and Tsutomu Kawabata. A Phenomenological Study on Threshold Improvement via Spatial Coupling. In: *IEICE Trans.* 95-A.5 (2012), pp. 974–977. DOI: 10.1587/transfun.E95.A.974 (cited on p. 7).

[Vig08] Sebastiano Vigna. Broadword Implementation of Rank/Select Queries. In: *Proc. 7th WEA*. 2008, pp. 154–168. DOI: 10.1007/978-3-540-68552-4_12 (cited on p. 44).

[Vio05] Alfredo Viola. Exact distribution of individual displacements in linear probing hashing. In: *ACM Trans. Algorithms* 1.2 (2005), pp. 214–242. DOI: 10.1145/1103963.1103965 (cited on pp. 100, 102).

[Wal18] Stefan Walzer. Load Thresholds for Cuckoo Hashing with Overlapping Blocks. In: *Proc. 45th ICALP*. 2018, 102:1–102:10. DOI: 10.4230/LIPIcs.ICALP.2018.102 (cited on p. 8).

[Wal20a] Stefan Walzer. *Experimental Comparison of Retrieval Data Structures*. 2020. URL: https://github.com/sekti/retrieval-test (visited on 01/04/2020) (cited on p. 140).

[Wal20b] Stefan Walzer. Peeling Close to the Orientability Threshold: Spatial Coupling in Hashing-Based Data Structures. In: *CoRR* (2020). arXiv: 2001.10500 (cited on pp. iv, 8, 75).

[War17] Lutz Warnke. Upper tails for arithmetic progressions in random subsets. In: *Israel Journal of Mathematics* (July 2017), pp. 1–49. ISSN: 1565-8511. DOI: 10.1007/s11856-017-1546-3 (cited on pp. 127, 129).

[WC81] Mark N. Wegman and Larry Carter. New Hash Functions and Their Use in Authentication and Set Equality. In: *J. Comput. Syst. Sci.* 22.3 (1981), pp. 265–279. DOI: 10.1016/0022-0000(81)90033-7 (cited on p. 54).

[Wie86] Douglas H. Wiedemann. Solving Sparse Linear Equations Over Finite Fields. In: *IEEE Trans. Inf. Theory* 32.1 (1986), pp. 54–62. DOI: 10.1109/TIT.1986.1057137 (cited on p. 56).

[WRS18] Sean A. Weaver, Hannah J. Roberts and Michael J. Smith. XOR-Satisfiability Set Membership Filters. In: *Proc. 21st SAT*. 2018, pp. 401–418. DOI: 10.1007/978-3-319-94144-8_24 (cited on p. 46).

[ZTR20] Jens Zentgraf, Henning Timm and Sven Rahmann. Cost-optimal assignment of elements in genome-scale multi-way bucketed Cuckoo hash tables. In: *Proc. ALENEX20*. 2020, pp. 186–198. DOI: 10.1137/1.9781611976007.15 (cited on pp. 29, 33).