Simple Hashing-Based Algorithms with Strong Theoretical Guarantees

PhD Thesis

Anders Aamand

Supervisors:
Mikkel Thorup and Mikkel Abrahamsen

Submitted August 31st, 2020

This thesis has been submitted to the PhD School of The Faculty of Science, University of Copenhagen

Abstract

In recent years, demand has immensely increased for algorithms for handling and analyzing large-scale data. The data often arrives in a stream, and if it is not processed in time, the information is lost. Indeed, the high volume of the data makes storing a complete copy and performing exact computations an intractable task. These challenges motivate the following general question. *Can we design efficient algorithms and data structures that provide reliable statistical analyses in large-scale data scenarios?*

A powerful tool in designing such algorithms is randomization, and in particular, randomization through the use of *hash functions*. In the analysis of randomized algorithms, such hash functions are commonly assumed to be *truly random*. However, this ignores the practical issue that efficiently implementing a truly random hash function is impossible, and further, that using too weak a hash function may completely undermine the strong theoretical guarantees attained in an idealised analysis. Often, particular structured data sets will cause the strong theoretical guarantees to vanish. This thesis presents new results on theoretical guarantees that can be obtained using *practical tabulation-based* hashing schemes. First, we introduce a new family of hash functions, *tabulation-permutation*, which is simple to implement and demonstrated to be extremely fast in practice. We prove that it provides strong concentration bounds on hash-based sums. We further show how access to such a hashing scheme leads to speedups of various streaming algorithms. Second, we study the number of non-empty bins when hashing n balls to m bins using *simple tabulation hashing*, another fast and practical hash function. In several ways, this distribution is shown to resemble that from the truly random scenario.

Along a slightly different line of research, we present new results on the streaming algorithm Count-Min and Count-Sketch for estimating frequencies of the elements of a data stream. We show how access to an oracle which can decide whether an item is a heavy hitter, can be used to obtain better frequency estimates.

Finally, we present a basic graph algorithm. The algorithm takes as input a graph, the edges of which have been partitioned into trails, and decides in linear time whether the trails can be consistently oriented to make the resulting directed graph strongly connected. Moreover, it finds such a strong trail orientation if it exists.

Dansk resumé

I de seneste år er efterspørgslen på algoritmer til at håndtere og analysere store mængder af data steget dramatisk. Typisk er der tale om strømme af data, hvis information går tabt, hvis de ikke behandles i tide. Det er nemlig umuligt at gemme en komplet kopi af data i så stor skala og udføre præcise analyser og beregninger. Disse udfordringer motiverer følgende generelle spørgsmål. *Kan vi designe algoritmer og datastrukturer, som effektivt og pålideligt kan analysere data i stor skala?*

Et vigtigt redskab i arbejdet med sådanne algoritmer er randomisering, specielt randomisering ved brug af *hashfunktioner*. I analysen af randomiserede algoritmer der bruger hashfunktioner antages det ofte, at disse hashfunktioner er *fuldt tilfældige*. Desværre er det umuligt at implementere fuldt tilfældig hashing i praksis, og ved brug af for svage hashfunktioner risikerer man at underminere de stærke teoretiske garantier, man får fra den idealiserede analyse med fuld tilfældighed. Ofte vil dårligt struktureret data få de stærke teoretiske garantier til at forsvinde.

Denne afhandling præsenterer nye resultater om de teoretiske garantier, som kan opnås ved brug af *praktiske* hashfunktioner baseret på *tabulering*. Vi introducerer først en ny familie af hashfunktioner, *tabulation-permutation*, som er let at implementere og ekstremt hurtig i praksis. Vi viser, at denne type hashing giver stærke koncentrationsresultater for hashbaserede summer. Vi viser også, hvordan adgangen til en hashfunktion med sådanne egenskaber giver hurtigere algoritmer til pålideligt at analysere strømme af data. Derefter studerer vi fordelingen af ikke-tomme spande når n bolde fordeles i m spande ved brug af *simple tabulation* hashing, som er en anden hurtig og praktisk hashfunktion. Vi viser, at denne fordeling på mange måder ligner den tilsvarende fordeling med fuldt tilfældig hashing.

I en lidt anden forskningsretning præsenterer vi nye resultater om algoritmerne Count-Min og Count-Sketch, der kan bruges til at estimere frekvenserne af elementerne fra en strøm af data. Vi viser, hvordan adgang til et orakel, der kan afgøre hvorvidt et element optræder hyppigt eller ej, kan bruges til at opnå bedre estimater.

Til sidst præsenterer vi en basal grafalgoritme. Som input får algoritmen en graf hvis kanter er opdelt i stier, og algoritmen afgør i lineær tid, om disse stier kan ensrettes konsistent således, at den resulterende orienterede graf bliver stærkt sammenhængende. Algoritmen finder desuden en sådan orientering af stierne hvis det er muligt.

Preface

The *General rules and guidelines for the PhD programme* at the Faculty of Science, University of Copenhagen allows for a PhD dissertation to be written "*as a synopsis with manuscripts of papers or already published papers attached*". The present dissertation has this form.

To get a coherent thesis I have chosen to only present a selected subset of my work. For completeness, I will now provide a very brief introduction to all my papers, including those not presented in this thesis. Throughout my PhD, I have been the co-author of nine papers, five of which are published at peer-reviewed conferences or journals and the remaining four being in submission. The full list of papers is as follows.

[AAKR19] Anders Aamand, Mikkel Abrahamsen, Jakob B. T. Knudsen, and Peter M. R. Rasmussen. Classifying convex bodies by their contact and intersection graphs. *CoRR*, 2019. `https://arxiv.org/abs/1902.01732`. In submission.

[AAT20] Anders Aamand, Mikkel Abrahamsen, and Mikkel Thorup. Disks in curves of bounded convex curvature. *The American Mathematical Monthly*, 127(7):579–593, 2020.

[ADK+20] Anders Aamand, Debarati Das, Evangelos Kipouridis, Jakob B. T. Knudsen, Peter M. R. Rasmussen, and Mikkel Thorup. No repetition: Fast streaming with highly concentrated hashing. *CoRR*, 2020. `arxiv.org/abs/2004.01156`. In submission.

[AHHR18] Anders Aamand, Niklas Hjuler, Jacob Holm, and Eva Rotenberg. One-way trail orientations. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, pages 6:1–6:13, 2018.

[AIV19] Anders Aamand, Piotr Indyk, and Ali Vakilian. (learned) frequency estimation algorithms under zipfian distribution. *CoRR*, 2019. `https://arxiv.org/abs/1908.05198`. In submission.

[AKK$^+$20] Anders Aamand, Jakob B. T. Knudsen, Mathias B. T. Knudsen, Peter M. R. Rasmussen, and Mikkel Thorup. Fast hashing with strong concentration bounds. *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, 2020.

[AKŁ$^+$20] Anders Aamand, Adam Karczmarz, Jakub Łacki, Nikos Parotsidis, Peter M. R. Rasmussen, and Mikkel Thorup. Randomized decremental connectivity in linear time for less dense graphs, 2020. In submission.

[AKT18] Anders Aamand, Mathias B. T. Knudsen, and Mikkel Thorup. Power of d choices with simple tabulation. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, pages 5:1–5:14, 2018. (**Based on my Master's thesis**).

[AT19] Anders Aamand and Mikkel Thorup. Non-empty bins with simple tabulation hashing. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 2498–2512, 2019.

Roughly speaking, these papers can be partitioned into three topics. The first is hash functions and hashing based algorithms, which is the main topic of this thesis [AKK$^+$20, ADK$^+$20, AKT18, AT19, AIV19]. The two remaining topics are graph algorithms [AHHR18, AKŁ$^+$20] and finally (computational) geometry [AAKR19, AAT20].

Hash functions and hashing based algorithms The primary focus of of my work within the field of hashing has been on the design of fast and practical hash functions that provably have some of the same good theoretical guarantees as fully random hashing [AKK$^+$20, ADK$^+$20, AT19, AKT18]. These papers are joint work with, among others, my advisor Mikkel Thorup. First, [AKK$^+$20] presents a new tabulation-based hashing scheme which satisfies Chernoff-style concentration bounds. The hashing is easy to implement and is demonstrated to be very fast in practice. Second, [ADK$^+$20] demonstrates how the access to a family of hash functions with such concentration guarantees yields speed-ups in different streaming algorithms. Third, [AT19] analyses the distribution of the number of non-empty bins when m balls are hashed to n bins using simple tabulation hashing. This paper has practical applications for the implementation of Bloom filters, filter hashing, and multi-choice cuckoo hashing. Finally, [AKT18] analyses the expected maximum load of a bin when $m = O(n)$ balls are distributed into n bins using the d-choice balanced allocation scheme with the hashing implemented as simple tabulation hashing.

Along a slightly different line of research is the paper [AIV19] on the frequency estimation algorithms Count-Min and Count-Sketch. It is a follow-up to a previous paper by, among others, my two co-authors which explored the classic frequency estimation algorithms Count-Min and Count-Sketch when augmented by an oracle that can predict if an item is a heavy hitter, in that case allocating it a unique bucket. Their paper showed empirical improvements using a machine learning oracle but it lacked a tight theoretical analysis. This is what is provided in [AIV19], which is a purely theoretical paper.

Graph algorithms I have further been involved in two projects on graph algorithms. The first one, [AHHR18], asks the following question. Suppose that the edges of the graph is partitioned into trails. Can we consistently orient the trails such that the resulting directed graph becomes strongly connected? The paper presents a simple necessary and sufficient criterion and a linear time algorithm for finding such a strongly connected trail orientation if it exists. The second paper, [AKŁ+20], considers decremental connectivity, that is, connectivity in a graph as edges are deleted. Starting with $m = \Omega(n^{3/2} \operatorname{polylog} n)$ edges, it present a Monte Carlo algorithm that supports all deletions in $O(m)$ total time, and can answer connectivity queries in constant time.

Geometry Finally, I have been involved in two projects within geometry. First, [AAKR19] classifies convex bodies by their contact, union and intersection graphs. Secondly, [AAT20] introduces the new notion of bounded convex curvature. The paper proves that any simple closed curve of bounded convex curvature contains a unit disk in its interior. Both of these papers are joint work with, among others, my advisor Mikkel Abrahamsen.

Papers included in this thesis This thesis first contains a general introduction which motivates the study of practical hashing and hashing based algorithms. It then proceeds to introduce the papers included in the thesis which are [AKK+20, AT19, ADK+20, AIV19, AHHR18]. The former four are all concerned with hashing or hashing based algorithms. The latter is an outlier, a basic graph algorithm, included to illustrate the breath of my PhD project.

Acknowledgments I want go give a big thank to my two advisors Mikkel Thorup and Mikkel Abrahamsen. Mikkel Thorup was the person to introduce me to theoretical computer science. His energy and passion for the field was what got me excited in the first place and he has been an inspiration to work with throughout my PhD. I also thank him for creating the collaborative, friendly and, on the whole, fantastic environment of BARC which has been a joy to be a part of. Mikkel Abrahamsen has been a great inspiration too. He introduced me to the field of computational geometry and I have collaborated with him on several exciting projects within this area. I am grateful for all the humorous and absurd conversations we have had (they include the design of a scale for grading oranges using words from the English language ranging from *traumatizing* to *memorable*). These conversations have always been a healthy breathing room for me. Thanks to both of you. I consider myself very lucky to have had you as advisors.

Next I wish to thank all the people of BARC. Together they have made it a friendly environment which has been perfect for academic and personal growth. I will never forget the hourlong discussions of interesting and difficult problems, the cozy evenings at restaurants and bars, the runs to the seaside for a swim in the warm days of summer, and our mushroom hunting trips to Tisvilde and Sweden. Thanks to you all.

I would like to thank Piotr Indyk who hosted me at MIT and introduced me to new and exciting problems within machine learning. I also wish to thank his former student Ali Vakilian for our productive discussions and for helping to make my visit to MIT a very pleasurable stay. Finally, I would like to thank Alexandr Andoni who hosted me for an academic visit at Columbia University and got me excited about edit distance and nearest neighbour search problems. You have all helped broaden my academic horizon, triggering my curiosity with new and exciting fields.

I also wish to thank all my co-authors Mikkel Abrahamsen, Debarati Das, Niklas Hjuler, Jacob Holm, Piotr Indyk, Adam Karczmarz, Evangelos Kipouridis, Jakob Bæk Tejs Knudsen, Mathias Bæk Tejs Knudsen, Jakub Łacki, Nikos Parotsidis, Peter Michael Reichstein Rasmussen, Eva Rotenberg, Mikkel Thorup, and Ali Vakilian. It has been inspiring to work with you and I am proud of the results we have obtained together.

Finally, I wish to thank my friends and family who have always been supportive. You have always been good at showing interest in what I do and at asking the right questions, even if what I have been working on has sometimes been very abstract. For that I am extremely grateful.

Contents

Part I

Synopsis

Chapter 1

Introduction

Recent years have brought with them an immense demand for methods for handling and analyzing large-scale data. The data often arrives in a streaming-like fashion and due to the high volume of the data, storing the full data stream and performing exact computation is an intractable task. One example of such a streaming scenario comes from the Internet. When processing packages passing through a high-end Internet router, each application only gets very limited time to look at each packet before it is forwarded. If it is not done in time, the information is lost and slowing down the Internet is typically not an option. These challenges motivate the following general question. *Can we design efficient algorithms and data structures that provide reliable statistical analyses in large-scale data scenarios*

Many interesting problems and influential algorithmic solutions have been studied within this line of research. A powerful common tool in designing algorithms for solving such problems is to use randomization, and in particular, randomization through the use of *hash functions*. A simplifying assumption when analysing the performance of these algorithms is that the hash functions are *fully random*, i.e., that the hash values of keys are independent and uniformly distributed. This assumption typically leads to a much simpler analysis. However, implementing fully random hashing is impossible in practice. If on the other hand a too weak hash function is used, the strong theoretical guarantees vanish, and the algorithms may break down completely on certain structured sets of data. It is therefore desirable to find *implementable* and *practical* hash functions which provably have some of the same good theoretical guarantees as fully random hashing.

A large part of this thesis is focused on new results on the theoretical guarantees that can be obtained with *tabulation-based* hashing schemes.

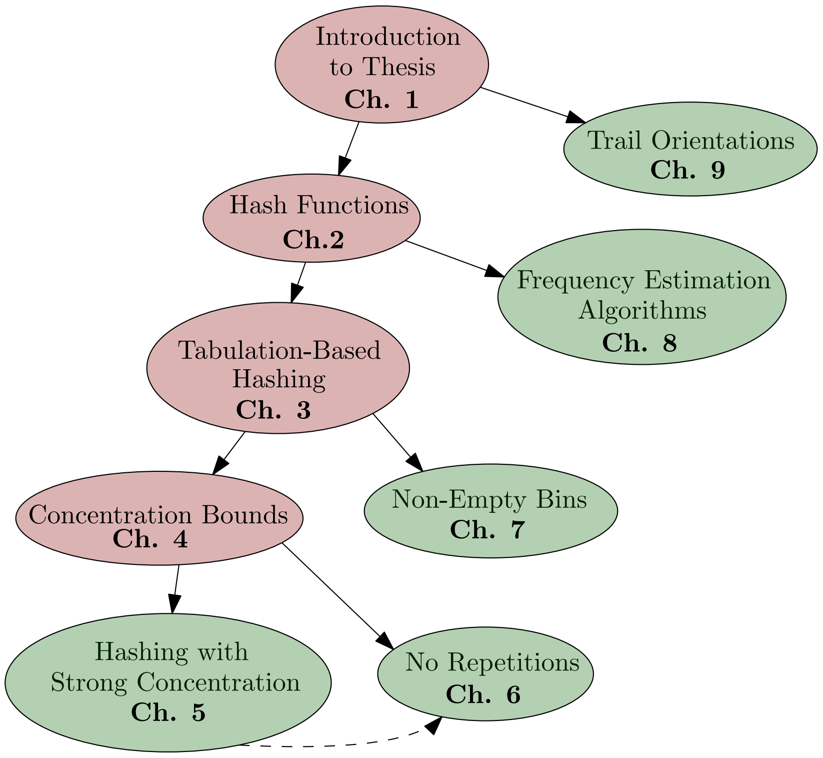
Introduction to Thesis **Ch. 1**
Trail Orientations **Ch. 9**
Hash Functions **Ch.2**
Frequency Estimation Algorithms **Ch. 8**
Tabulation-Based Hashing **Ch. 3**
Concentration Bounds **Ch. 4**
Non-Empty Bins **Ch. 7**
Hashing with Strong Concentration **Ch. 5**
No Repetitions **Ch. 6**

Figure 1.1: The nine chapters of this thesis in a tree. We present these chapters in depth-first search order. Each leaf represents a paper. In order to obtain a complete introduction to a given paper, the reader is encouraged to follow the path from the root to the corresponding leaf, reading each chapters met along the way.

First, it presents a new family of hash functions, *tabulation-permutation* hashing which is shown to provide strong concentration guarantees of hash-based sums. It is simple to implement and demonstrated to be extremely fast in practice. It is shown how such concentration guarantees give speedups of different streaming algorithms, namely when estimating set similarity or estimating the number of distinct elements of the stream. Second, the thesis studies the number of non-empty bins when hashing n balls to m bins using *simple tabulation hashing*, another simple and practical hashing scheme. In several ways, this distribution is shown to resemble that from the fully random scenario. For example, this yields an asymptotically space-optimal implementation of Bloom filters.

Along a slightly different line of research, the thesis presents new results on the streaming algorithm Count-Min and Count-Sketch for estimating frequencies of the elements of a data stream. In [HIKV19], it was shown empirically that these algorithms can be improved using a machine learning oracle that predicts whether an item of the stream is a heavy hitter. Under the common assumption that the input follows a Zipfian distribution, the paper also presented a preliminary analysis of Count-Min assuming that the heavy hitter oracle is perfect. Unfortunately, their analysis was not tight and they provided no analysis of Count-Sketch. In this thesis we present the first tight bounds for Count-Min and Count-Sketch, both their standard variants and when augmented by an oracle for detecting heavy hitters.

Finally, to demonstrate the breath of this PhD project, the thesis presents a basic graph algorithm. The input to the algorithm is a graph, the edges of which have been partitioned into trails, and the algorithm decides in linear time whether the trails can be consistently oriented making the resulting directed graph strongly connected. It moreover finds such a strong trail orientation if it exists.

Structure of this thesis. The thesis consists of nine chapters presented in Figure 1.1 as nodes of a rooted tree. Each leaf represents a paper and the corresponding chapter is an introduction to that paper. The chapter presents the main results of the paper and their relation to international state-of-the-art. In order to obtain a complete introduction to a given paper, the reader is encouraged to follow the path from the root to the corresponding leaf, reading each chapters met along the way. The thesis presents the chapters of the tree in depth-first search order. The dashed line in the tree indicates that Chapter 5 is very relevant for Chapter 6 even though the two chapters can be read independently. We do not provide any proofs of our main results in the introductory chapters, but refer the reader to the full papers in the appendix. Chapters 5 to 9 are slightly modified subsets of the introductions from the full papers, so the reader might just as well read these papers.

Chapter 2

Hash Functions

In this chapter, we provide a very brief introduction to the field of hashing and hashing based algorithms.

The concept of a hash function dates back to the 1950s [Dum56] and today it is one of the fundamental tools in the design of randomized algorithms and data structures. A *hash function* is a random map $h : U \to R$ from a universe of keys, U, to a range, R, chosen with respect to some probability distribution, $\mathcal{D}$, on the set of all such functions. Here both U and R are typically bounded integer ranges, $U = [u] = \{0, 1, \ldots, u-1\}$ and $R = [m] = \{0, 1, \ldots, m-1\}$. In this thesis, $\mathcal{D}$ will always be the uniform distribution restricted to some subset $\mathcal{H} \subseteq R^U$ of the set of all such functions. In this case, we say that $\mathcal{H}$ is a *family* of hash functions, implicitly assuming that $\mathcal{D}$ is the uniform distribution over $\mathcal{H}$. It is common to refer to the keys of U as *balls* and to R as a set of *bins*. With this terminology, the hash function can be thought of as throwing the balls of U into the bins of R randomly according to the distribution $\mathcal{D}$. If $\mathcal{D}$ is the uniform distribution on R^U, h is said to be a *truly random* or a *fully random* hash function. A truly random hash function $h : U \to R$ thus assigns a uniformly random hash value $h(x) \in R$ to each key $x \in U$ and the hash values $(h(x))_{x \in U}$ are mutually independent.

Truly random hash functions play a special role in the study of hashing-based algorithms and data structures. Indeed, assuming access to truly random hash functions, the mutual independence of the hash values often leads to a simple probabilistic analysis that demonstrates quite strong theoretical guarantees on the performance of the algorithm. Unfortunately, truly random hash functions cannot be implemented in practice. To represent a truly random hash function, we need to store $|U| \log_2 |R|$ random bits and in most applications the universe, U, is far too large for such a representation. In fact, the whole idea behind hashing is that for large U, we hope to get away with using a much smaller random seed and still preserve the good properties of the hash function. More precisely, we seek *practical* families of hash functions that can be sampled using a small random seed and quickly evaluated, that are still 'random enough' to provide the same good theoretical guarantees as truly random hashing.

An example of a hash function that uses a small random seed is the map $h : [p] \to [p]$ defined by $h(x) = (ax + b) \mod p$, where p is prime and a and b are independent and uniformly random members of $[p]$. To represent h, we only require $2\lceil \log_2 p \rceil$ bits and the hash function can be evaluated in constant time. However, in many applications, h would be too weak to provide the desired theoretical guarantees.

Let us provide an example of a hashing-based algorithm, namely the classic *hashing with chaining*. In hashing with chaining (see, e.g., [Knu98]), we distribute the keys of a set $X \subseteq U$ into a table using a hash function $h : U \to R$, storing $x \in X$ at entry $h(x)$. Collisions are handled by making a linked list of all keys hashing to the same entry. Consider the special case where $|X| = |R| = n$, i.e., where we distribute n balls into n bins. If h is truly random, the probability that there exists a bin receiving more than k balls is at most

$$n\binom{n}{k}n^{-k} \le n\left(\frac{e}{k}\right)^k ,$$

using a simple union bound. Choosing $k = O(\log n / \log\log n)$ sufficiently large, it follows from standard calculations that the maximum load of a bin is at most k with high probability, and this is also a high probability upper bound on the length of any of the linked lists. In this analysis we used that any $k = O(\log n / \log\log n)$ distinct keys have mutual independent hash values — which in particular is the case when h is truly random.

When studying a family of hash functions, $\mathcal{H}$, we are fundamentally interested in its performance on three parameters

1. **Time**. The time needed to evaluate the hash function.

2. **Space.** The space needed to represent the hash function.

3. **Theoretical guarantees.** In principle, for any property of a truly random hash function or of randomized algorithm that assumes access

to a truly random hash function, we can ask whether there exists a practical hash family that provides similar theoretical guarantees. The theoretical guarantees of the idealized truly random hash functions are nearly always our baseline.

Examples of the theoretical guarantees in 3. come in abundances. Relevant to this thesis are concentration bounds on hash-based sums, distribution on the number of non-empty bins, and the level of independence of the hashing scheme. Another example comes from [AKT18], which considers the maximum load of a bin when $n = O(m)$ balls are distributed into n bins in the d-choice balanced allocation scheme from [ABKU99].

k-independence In 1979 Wegman and Carter [WC81] introduced the important notion of k-independence of a hash function. A hash function, h, drawn from a family of hash functions, $\mathcal{H}$, is said to be *k-independent* or *k-universal* if

1. For any k distinct keys, $x_1, \ldots, x_k \in U$, the hash values $h(x_1), \ldots, h(x_k)$ are mutually independent.
2. For each $x \in U$, $h(x)$ is uniformly in R.

When analysing the performance of a hash function in a given application, it is common to use its independence. For example, we are typically interested in how well a hash based sum, X, is concentrated around its mean, μ. If the hash function is k-independent (k even), we can use the classic *k'th moment bound*

$$\Pr[|X - \mu| \geq t] \leq \mathbb{E}\left[|X - \mu|^k\right]/t^k.$$

The k-independence ensures that $\mathbb{E}\left[|X - \mu|^k\right]$ is the same as if h were a truly random hash function. An important special case of the definition of k-independence is the choice $k = 2$. A hash function that is 2-independent is said to be *strongly universal* and for a strongly universal hash function, the moment bound above is just an application of Chebyshev's inequality. In Chapter 6, we will see an example of such a moment bound in the analysis of two streaming algorithms.

The classic construction of a k-independent hash function is the polynomial hashing scheme introduced by Wegman and Carter [WC81]. Here, the hash function $h : [p] \to [p]$, p prime, is obtained by choosing $a_0, \ldots, a_{k-1} \in$

$[p]$ independently and uniformly at random and defining

$$h(x) = \left(\sum_{i=0}^{k-1} a_i x_i \right) \mod p. \tag{2.1}$$

If one wish to hash to $R = [m]$, one can choose $p \gg m$ and define $h_1(x) = h(x) \mod m$. Then part 2. of the definition of k-independence is only approximately satisfied but usually this is not a problem. In hashing based algorithms, the truly random hash function can often be replaced with a k-independent hash function for a sufficiently large k and still give the same theoretical guarantees. For example, in the analysis of hashing with chaining above, it is enough to assume that the hash function is k-independent for $k = O(\log n / \log \log n)$ sufficiently large.

The drawback of resorting to the k-independent hash function as defined in (2.1) is that in applications involving n objects, we often have to choose $k = \Omega(\log n)$, or $k = \Omega(\log n / \log \log n)$ as in our running example. With an evaluation time of $\Theta(k)$, this is typically too large. In the next chapter, we introduce tabulation-based hashing which uses more space, but can be evaluated in constant time and which provides surprisingly strong theoretical guarantees even though it is often only 3-independent.

Chapter 3

Tabulation-Based Hashing

We next proceed to introduce tabulation-based hashing. Parts of this chapter are taken from the introduction of [AKK$^+$20].

If we aim for strong theoretical guarantees using k-independent hashing, e.g., the polynomial hashing scheme in (2.1), we are often required to put $k = \Omega(\log n)$ in problems that involve the hashing of n keys from U. With an evaluation time of $\Omega(k)$, this is typically too slow in practice. It turns out that this problem is inherent to k-independent hashing that use little space in the following sense: Siegel [Sie04] has shown that any k-independent hash function, which requires less that k memory probes to be evaluated, must use space $\Omega(|U|^{1/k})$ to be represented. If we want highly independent hashing that can be evaluated in constant time, we thus have to use a lot of space. Siegel also presents a family of hash functions which uses space $O(|U|^{1/c})$, can be evaluated in time $c^{O(c)}$ and which is $|U|^{\Omega(1/c^2)}$-independent. However, in the same paper he writes that it is "*far too slow for any practical applications*".

The hashing scheme presented by Siegel in [Sie04] is an example of a *tabulation-based* hashing scheme. What these hashing schemes have in common is that they need access to large tables of random entries. We think of these tables as functions $T : A \to R$, where A is some set, typically of size $|A| = |U|^{\varepsilon}$ for some $\varepsilon = \Omega(1)$, or $|A| = O(n)$ in applications involving the hashing of n items. Let us highlight the two fundamental differences between the polynomial hashing scheme in (2.1) and tabulation-based hashing. First, to sample a polynomial hash function, we require a random seed of size proportional to its independence and we need the same amount of space to store the hash function. In contrast, storing the tables $T : A \to R$

requires significantly more space[1], typically $|U|^{\varepsilon}$ or $O(n)$. Second, when evaluating polynomial hashing on a key, we need to access the entire random seed, whereas for tabulation-based hashing methods, we only access a small subset of the table.

Since Siegel's paper [Sie04], the theoretical and practical properties of tabulation-based hashing have studied extensively. Several new tabulation-based hashing schemes have been proposed for k-independence with evaluation times better than $O(k)$ (see, e.g., [Tho13b, CPT15]). While these schemes are significantly faster than Siegel's initial hash family, they are still not very useful in practice when k becomes too large, e.g., $k = \log |U|$. These issues motivate the study of practical families of hash functions which provide strong theoretical guarantees via different properties than their independence. Surprisingly, it turns out that tabulation-based hashing often gives such strong guarantees. One of the earliest results in this vein is a simple and elegant construction by Dietzfelbinger and Meyer auf der Heide [DM92]. They present a family of hash functions that to some degree provides Chernoff-style bounds on the number of balls in a bin when n balls are hashed to m bins. In a different direction, Dietzfelbinger and Rink [DR09] create a tabulation-based hash function that is highly independent (building on previous works [DW07, DW03, FPSS05, HT01]) but, contrasting the double tabulation from [Tho13b], only within a fixed set S, not the entire universe. The construction requires an upper bound n on the size of S, and a polynomial error probability of $n^{-\gamma}$ is tolerated. Assuming no such error has occurred, the hash function is fully independent on S. In the same setting, Pagh and Pagh [PP08] have presented a hash function that uses $(1 + o(1))n$ space and which is fully independent on any given set S of size at most n with high probability. This result is very useful, e.g., as part of solving a static problem of size n using linear space, since, with high probability, we may assume fully-random hashing as a subroutine. However, the hashing scheme in [PP08] uses the highly independent hashing of Siegel [Sie04] as a subroutine and, as such, is not very practical. Dietzfelbinger and Woelfel [DW03] found a simpler subroutine that worked in the setting of [PP08] even though it was not highly independent. The most efficient hashing scheme known to provide full randomness on a fixed set, as in [PP08], is the double tabulation scheme by Thorup [Tho13b] (see the analysis in [DKRT15]).

[1]Here it should be mentioned that it is often enough to fill the tables using an $O(\log |U|)$-independent random number generator. We can therefore get away with using a small random seed, so the crucial difference from polynomial hashing is that we require a lot of space to store the hash function if we want to evaluate it quickly.

In this thesis, we present several new results on the theoretical guarantees provided by different tabulation-based hashing schemes. To get started, we need to define and introduce the most basic tabulation-based hashing scheme, namely *simple-tabulation* hashing.

Simple Tabulation Hashing *Simple tabulation* hashing dates back to Zobrist [Zob70]. In simple tabulation hashing, we consider the key domain U to be of the form $U = \Sigma^c$ for some character alphabet Σ and $c = O(1)$, such that each key consists of c characters of Σ. Let $m = 2^\ell$ be given and identify $[m] = \{0, 1, \dots, m-1\}$ with $[2]^\ell$. A simple tabulation hash function, $h\colon \Sigma^c \to [m]$, is then defined as follows. For each $j \in \{1, \dots, c\}$, store a fully random character table $h_j\colon \Sigma \to [m]$ mapping characters of the alphabet Σ to ℓ-bit hash values. To evaluate h on a key $x = (x_1, \dots, x_c) \in \Sigma^c$, we compute

$$h(x) = h_1(x_1) \oplus \cdots \oplus h_c(x_c),$$

where $\oplus$ denotes bitwise XOR – an extremely fast operation. With character tables in cache, this scheme is the fastest known 3-independent hashing scheme [PT12]. We will denote by $u = |U|$ the size of the key domain, identify $U = \Sigma^c$ with $[u]$, and always assume the size of the alphabet, $|\Sigma|$, to be a power of two. For instance, we could consider 32-bit keys consisting of four 8-bit characters. For a given computer, the best choice of c in terms of speed is easily determined experimentally once and for all, and is independent of the problems considered.

As stated, simple tabulation is only 3-independent. Indeed, in the simplest case $c = 2$ we can let $a_1, a_2, b_1, b_2 \in \Sigma$ with $a_1 \neq a_2$ and $b_1 \neq b_2$ and define $x = (a_1, b_1)$, $y = (a_1, b_2)$, $z = (a_2, b_1)$ and $w = (a_2, b_2)$. Then $h(x) \oplus h(y) \oplus h(z) \oplus h(w) = 0$, regardless of the randomness of h, so the keys x, y, z, w do not hash independently. Despite its low independence, simple tabulation turns out to have surprisingly strong theoretical properties. The first systematic study of these properties appeared in [PT12], which provides analyses of its concentration guarantees and its performance in linear probing, cuckoo hashing, and min-wise hashing. In [DKRT15], it is shown that it satisfies k-moment bounds for any constant k. Finally [DKRT16, AKT18] studies its performance in the 2-and d-choice balanced allocation schemes.

Several hashing schemes building on simple tabulation have since been proposed, e.g., *twisted tabulation* [PT13], *double tabulation* [Tho13b] and *mixed tabulation* [DKRT15] and they have been proven to have even stronger theoretical guarantees in several ways. As they are not of direct importance to the results presented in this thesis, we will not dive into further detail with these guarantees. See, however, Chapter 5 for a discussion of the concentration guarantees provided by these hashing schemes.

In this thesis, we will present two new results on tabulation-based hashing schemes.

Hashing with Strong Concentration. In Chapter 5, we introduce a new tabulation-based hashing scheme, *tabulation-permutation* hashing [AKK$^+$20] which is proven to satisfy Chernoff-style concentration bounds in a very strong sense. Leading up to the presentation of this result, we start out by providing an introduction to such Chernoff-style concentration bounds in Chapter 4. In chapter 6, we present the paper [ADK$^+$20] which shows how hashing with strong concentration bounds can be used to obtain speedups in different streaming algorithms. As such, the results of [ADK$^+$20] are independent of those of [AKK$^+$20], but the hashing scheme from [AKK$^+$20] provides the strong concentration guarantees needed for a reliable implementation of the algorithms of [ADK$^+$20].

Non-Empty Bins with Simple Tabulation In Chapter 7, we present the paper [AT19] which studies the distribution of non-empty bins when a set of keys $X \subset U$ is distributed into $[m]$ bins using a simple tabulation hash function $h : U \to [m]$.

Chapter 4

Chernoff-Style Concentration Bounds

This chapter introduces Chernoff-style concentration bounds. What follows is a subset of the introduction of the paper [AKK$^+$20].

Chernoff's concentration bounds [Che52] date back to the 1950s but bounds of this types go even further back to Bernstein in the 1920s [Ber24]. Originating from the area of statistics they are now one of the most basic tools of randomized algorithms [MR95]. A canonical form considers the sum $X = \sum_{i=1}^{n} X_i$ of independent random variables $X_1, \ldots, X_n \in [0,1]$. Writing $\mu = \mathbb{E}[X]$ it holds for every $\varepsilon \geq 0$ that

$$\Pr[X \geq (1+\varepsilon)\mu] \leq \exp(-\mu\, \mathcal{C}(\varepsilon)) \qquad \left[\leq\ \exp(-\varepsilon^2\mu/3) \text{ for } \varepsilon \leq 1\right], \quad (4.1)$$

$$\Pr[X \leq (1-\varepsilon)\mu] \leq \exp(-\mu\, \mathcal{C}(-\varepsilon)) \quad \left[\leq\ \exp(-\varepsilon^2\mu/2) \text{ for } \varepsilon \leq 1\right]. \quad (4.2)$$

Here $\mathcal{C} : (-1, \infty) \to [0, \infty)$ is given by $\mathcal{C}(x) = (x+1)\ln(x+1) - x$, so $\exp(-\mathcal{C}(x)) = \frac{e^x}{(1+x)^{(1+x)}}$. Textbook proofs of 4.1 and 4.2 can be found in [MR95, §4][1]. Writing $\sigma^2 = \mathrm{Var}[X]$, a more general bound is

$$\Pr[|X-\mu| \geq t] \leq 2\exp(-\sigma^2\mathcal{C}(t/\sigma^2)) \quad \left[\leq\ 2\exp(-(t/\sigma)^2/3) \text{ for } t \leq \sigma^2\right]. \quad (4.3)$$

Since $\sigma^2 \leq \mu$ and $\mathcal{C}(-\varepsilon) \leq 1.5\,\mathcal{C}(\varepsilon)$ for $\varepsilon \leq 1$, (4.3) is at least as good as (4.1) and (4.2), up to constant factors, and often better. The bound of (4.3) is known as Bennett's inequality [Ben62].

[1] The bounds in [MR95, §4] are stated as working only for $X_i \in \{0,1\}$, but the proofs can easily handle any $X_i \in [0,1]$.

4.1 Concentration of Hash-Based Sums

In Chapter 5 of this thesis, we will be concerned with Chernoff-style concentration bounds on hash-based sums. The setting is as follows. We choose a random hash function, $h : U \to R$, which assigns a hash value, $h(x) \in R$, to every key $x \in U$. Here, $R = [m]$ for some $m \in \mathbb{N}$. We aim to obtain concentration bounds on a random variable, X, which is of one of the following forms of increasing generality.

1. Let $S \subseteq U$ be a set of balls and assign to each ball, $x \in S$, a weight, $w_x \in [0, 1]$. We wish to distribute the balls of S into a set of bins $R = [m] = \{0, 1, \dots, m-1\}$. For a bin, $y \in [m]$, $X = \sum_{x \in S} w_x \cdot [h(x) = y]$ is then the total weight of the balls landing in bin y.
2. We may instead be interested in the total weight of the balls with hash values in the interval $[y_1, y_2)$ for some $y_1, y_2 \in [m]$, that is, $X = \sum_{x \in S} w_x \cdot [y_1 \leq h(x) < y_2]$.
3. More generally, we may consider a fixed *value function* $v \colon U \times R \to [0, 1]$. For each key $x \in U$, we define the random variable $X_x = v(x, h(x))$, where the randomness of X_x stems from that of $h(x)$. We write $X = \sum_{x \in U} v(x, h(x))$ for the sum of these values.

To exemplify applications, the first case is common when trying to allocate resources; the second case arises in streaming algorithms; and the third case handles the computation of a complicated statistic, X, on incoming data. In each case, we wish the variable X to be concentrated around its mean, $\mu = \mathbb{E}[X]$, according to the Chernoff-style bound of (4.3). If we had fully random hashing, this would indeed be the case. However, as discussed in Chapter 2, storing a fully random hash function is impossible in practice.

To measure the strength of the concentration of a hash-based random variable as above, we require the following definition of strong concentration.

Definition 4.1 (Strong Concentration)**.** *Let $h \colon [u] \to [m]$ be a hash function, $S \subseteq [u]$ be a set of hash keys of size $n = |S|$, and $X = X(h, S)$ be a random variable, which is completely determined by h and S. Denote by $\mu = \mathbb{E}[X]$ and $\sigma^2 = \mathrm{Var}[X]$ the expectation and variance of X. We say that X is* strongly concentrated with added error probability *$f(u, n, m)$ if for every $t > 0$,*

$$\Pr[|X - \mu| \geq t] \leq O\left(\exp\left(-\Omega(\sigma^2 \mathcal{C}(t/\sigma^2))\right)\right) + f(u, n, m). \tag{4.4}$$

Note that strong concentration resembles that of (4.3) except for the constant delay in the exponential decrease and the added error probability of $f(u, n, m)$. In Chapter 5, we will present the paper [AKK$^+$20] which introduces a very practical constant-time hash family. For any $\gamma = O(1)$ the hash family provides strong concentration with added error probability $u^{-\gamma}$ on random variables, X, as in each of the three cases described above. In Chapter 6, we will present the paper [ADK$^+$20] which demonstrate how such a hash family leads to speedups in different streaming algorithms.

Chapter 5

Hashing with Strong Concentration

This chapter presents the results of the paper "*Fast Hashing with Strong Concentration Bounds*" [AKK$^+$20] of Appendix A. What follows is an extracted and slightly modified subset of the introduction from the paper.

5.1 Introduction

Pătraşcu and Thorup have shown that Chernoff-style bounds on hash based sums can be achieved in constant time with tabulation based hashing methods; namely simple tabulation [PT12] for case 1. of Section 4.1 and twisted tabulation [PT13] for all three cases. However, their results suffer from some severe restrictions on the expected value, μ, of the sum. More precisely, the speed of these methods relies on using space small enough to fit in fast cache, and the Chernoff-style bounds [PT12, PT13] all require that μ is much smaller than the space used. For larger values of μ, Pătraşcu and Thorup [PT12, PT13] offered some weaker bounds with a deviation that was off by several logarithmic factors. It can be shown that some of these limitations are inherent to simple and twisted tabulation. For instance, they cannot reliably distribute balls into $m = 2$ bins, as in case 1. of Section 4.1, if the expected number of balls in each bin exceeds the space used.

In [AKK$^+$20], we introduce and analyse a new family of fast hash functions, *tabulation-permutation* hashing, that provides strong concentration with added added error probability $u^{-\gamma}$ for any $\gamma = O(1)$ without any restrictions on μ (see Definition 4.1). Our bounds hold for all of the three cases described in Section 4.1 and all possible inputs. Furthermore, tabulation-permutation hashing is an order of magnitude faster than any other known hash function with similar concentration bounds, and almost as fast as simple and twisted tabulation. This is demonstrated both theoretically and experimentally in [AKK$^+$20].

The main theoretical contribution of [AKK$^+$20] lies in the field of analysis of algorithms, and is in the spirit of Knuth's analysis of linear probing [Knu63], which shows strong theoretical guarantees for a very practical algorithm. We show that tabulation-permutation hashing has strong theoretical Chernoff-style concentration bounds. Moreover, on the practical side, we perform experiments, summarized in Table 5.1, demonstrating that it is comparable in speed to some of the fastest hash functions in use, none of which provide similar concentration bounds.

Concrete examples of the utility of our new hash-family are discussed in Chapter 6 which presents the paper [ADK$^+$20]. In [ADK$^+$20] it is shown that some classic streaming algorithms enjoy very substantial speed-ups when implemented using tabulation-permutation hashing; namely the original similarity estimation of Broder [Bro97] and the estimation of distinct elements of Bar-Yossef et al. [BYJK$^+$02]. The strong concentration bounds makes the use of independent repetitions unnecessary, allowing the implementations of the algorithms to be both simpler and faster. We stress that in high-volume streaming algorithms, speed is of critical importance.

Tabulation-permutation hashing builds on top of simple tabulation hashing, and to analyse it, we require a new and better understanding of the behaviour and inherent limitations of simple tabulation. Let $S \subseteq U$ and consider hashing $n = |S|$ weighted balls or keys into $m = 2^\ell$ bins using a simple tabulation function, $h\colon [u] \to [m]$, in line with case 1. of Section 4.1. In [AKK$^+$20], we prove the following new theorem on the concentration guarantees provided by simple tabulation.

Theorem 5.1 ([AKK$^+$20])**.** *Let $h\colon [u] \to [m]$ be a simple tabulation hash function with $[u] = \Sigma^c$, $c = O(1)$. Let $S \subseteq [u]$ be given of size $n = |S|$ and assign to each key/ball $x \in S$ a weight $w_x \in [0,1]$. Let $y \in [m]$, and define $X = \sum_{x\in S} w_x \cdot [h(x) = y]$ to be the total weight of the balls hashing to bin y. Then for any constant $\gamma > 0$, X is strongly concentrated with added error probability n/m^γ, where the constants of the asymptotics are determined solely by c and γ.*

In Theorem 5.1, we note that the expectation, $\mu = \mathbb{E}[X]$, and the variance, $\sigma^2 = \mathrm{Var}[X]$, are the same as if h were a fully random hash function since h is 3-independent. The bound provided by Theorem 5.1 is therefore the same as the variance based Chernoff bound (4.3) except for a constant delay in the exponential decrease and an added error probability of n/m^γ. Since $\sigma^2 \leq \mu$, Theorem 5.1 also implies the classic one-sided Chernoff bounds (4.1) and (4.2), again with the constant delay and the added error probability as above, and a leading factor of 2.

Pătraşcu and Thorup [PT12] proved an equivalent probability bound, but without weights, and, more importantly, with the restriction that the number of bins $m \geq n^{1-1/(2c)}$. In particular, this implies the restriction $\mu \leq |\Sigma|^{1/2}$. Our new bound gives Chernoff-style concentration with high probability in n for any $m \geq n^\epsilon$, $\epsilon = \Omega(1)$. Indeed, letting $\gamma' = (\gamma+1)/\epsilon$, the added error probability becomes $n/m^{\gamma'} \leq 1/n^\gamma$.

However, for small m the error probability n/m^γ is prohibitive. For instance, unbiased coin tossing, corresponding to the case $m = 2$, has an added error probability of $n/2^\gamma$ which is useless. In [AKK$^+$20], we show that it is inherently impossible to get good concentration bounds using simple tabulation hashing when the number of bins m is small. To handle all instances, including those with few bins, and to support much more general Chernoff bounds, we introduce a new hash function: tabulation-permutation hashing.

5.1.1 Tabulation-Permutation Hashing

We start by defining *tabulation-permutation hashing* from Σ^c to Σ^d with $c, d = O(1)$. A tabulation-permutation hash function $h\colon \Sigma^c \to \Sigma^d$ is given as a composition, $h = \tau \circ g$, of a simple tabulation hash function $g\colon \Sigma^c \to \Sigma^d$ and a permutation $\tau\colon \Sigma^d \to \Sigma^d$. The permutation is a coordinate-wise fully random permutation: for each $j \in \{1, \ldots, d\}$, pick a uniformly random character permutation $\tau_j : \Sigma \to \Sigma$. Now, $\tau = (\tau_1, \ldots, \tau_d)$ in the sense that for $z = (z_1, \ldots, z_d) \in \Sigma^d$, $\tau(z) = (\tau_1(z_1), \ldots, \tau_d(z_d))$. In words, a tabulation-permutation hash function hashes c characters to d characters using simple tabulation, and then randomly permutes each of the d output characters.

The main result of [AKK$^+$20] is that with tabulation-permutation hashing, we get high probability Chernoff-style bounds for the very general case 3. described in Section 4.1 dealing with arbitrary value functions.

Theorem 5.2 ([AKK$^+$20]). *Let $h\colon [u] \to [r]$ be a tabulation-permutation hash function with $[u] = \Sigma^c$ and $[r] = \Sigma^d$, $c, d = O(1)$. Let $v\colon [u] \times [r] \to [0,1]$ be a fixed value function that to each key $x \in [u]$ assigns a value $X_x = v(x, h(x)) \in [0,1]$ depending on the hash value $h(x)$ and define $X = \sum_{x \in [u]} X_x$. For any constant $\gamma > 0$, X is strongly concentrated with added* *error probability $1/u^\gamma$, where the constants of the asymptotics are determined solely by c, d, and γ.*

5.1.2 Tabulation-1Permutation

Above we introduced tabulation-permutation hashing which yields Chernoff-style bounds with an arbitrary value function. This is the same general scenario as was studied for twisted tabulation in [PT13]. However, for almost all applications we are aware of, we only need the generality of case 2. discussed in Section 4.1. Recall that in this case we are only interested in the total weight of the balls hashing to a certain interval. As it turns out, a significant simplification of tabulation-permutation hashing suffices to achieve strong concentration bounds. We call this simplification *tabulation-1permutation.* Tabulation-permutation hashing randomly permutes each of the d output characters of a simple tabulation function $g\colon \Sigma^c \to \Sigma^d$. Instead, tabulation-1permutation only permutes the most significant character.

More precisely, a tabulation-1permutation hash function $h\colon \Sigma^c \to \Sigma^d$ is a composition, $h = \tau \circ g$, of a simple tabulation function, $g\colon \Sigma^c \to \Sigma^d$, and a random permutation, $\tau\colon \Sigma^d \to \Sigma^d$, of the most significant character, $\tau(z_1, \ldots, z_d) = (\tau_1(z_1), z_2, \ldots, z_d)$ for a random character permutation $\tau_1\colon \Sigma \to \Sigma$.

This simplified scheme, needing only $c+1$ character lookups, is powerful enough for concentration within an arbitrary interval.

Theorem 5.3 ([AKK+20]). *Let $h\colon [u] \to [r]$ be a tabulation-1permutation hash function with $[u] = \Sigma^c$ and $[r] = \Sigma^d$, $c, d = O(1)$. Consider a key/ball set $S \subseteq [u]$ of size $n = |S|$ where each ball $x \in S$ is assigned a weight $w_x \in [0, 1]$. Choose arbitrary hash values $y_1, y_2 \in [r]$ with $y_1 \leq y_2$. Define $X = \sum_{x \in S} w_x \cdot [y_1 \leq h(x) < y_2]$ to be the total weight of balls hashing to the interval $[y_1, y_2)$. Then for any constant $\gamma > 0$, X is strongly concentrated with added error probability $1/u^\gamma$, where the constants of the asymptotics are determined solely by c, d, and γ.*

One application of Theorem 5.3 is in the following sampling scenario: We set $y_1 = 0$, and sample all keys with $h(x) < y_2$. Each key is then sampled with probability y_2/r, and Theorem 5.3 gives concentration on the number of samples. In [ADK+20], to be presented in Chapter 6, this is used for more efficient implementations of streaming algorithms.

	Running time (ms)			
	Computer 1		Computer 2	
Hash function	32 bits	64 bits	32 bits	64 bits
Multiply-Shift	4.2	7.5	23.0	36.5
2-Independent PolyHash	14.8	20.0	72.2	107.3
Simple Tabulation	13.7	17.8	53.1	55.9
Twisted Tabulation	17.2	26.1	65.6	92.5
Mixed Tabulation	28.6	68.1	120.1	236.6
Tabulation-1Permutation	16.0	19.3	63.8	67.7
Tabulation-Permutation	27.3	43.2	118.1	123.6
Double Tabulation	1130.1	–	3704.1	–
"Random" (100-Independent PolyHash)	2436.9	3356.8	7416.8	11352.6

Table 5.1: The time for different hash functions to hash 10^7 keys of length 32 bits and 64 bits, respectively, to ranges of size 32 bits and 64 bits. The experiment was carried out on two computers. The hash functions written in italics are those without general Chernoff-style bounds. Hash functions written in bold are the contributions of this paper. The hash functions in regular font are known to provide Chernoff-style bounds. Note that we were unable to implement double tabulation from 64 bits to 64 bits since the hash tables were too large to fit in memory.

5.1.3 Experiments and Comparisons

To better understand the real-world performance of our new hash functions in comparison with well-known and comparable alternatives, we performed some simple experiments on regular laptops, as presented in Table 5.1. We did two types of experiments.

- On the one hand we compared with potentially faster hash functions with weaker or restricted concentration bounds to see how much we lose in speed with our theoretically strong tabulation-permutation hashing. These other hashing schemes were Multiply-Shift [Die96], 2-independent PolyHash [WC81], simple-tabulation [PT12], twisted tabulation [PT13], mixed-tabulation [DKRT15]. We observed that our tabulation-permutation is very competitive in speed.

- On the other hand we compared with the fastest previously known hashing schemes with strong concentration bounds like ours (double tabulation [Tho13b], 100-independent PolyHash [WC81]). Here we observed a gain of a factor of 30 in speed.

The results of our experiments are presented in Table 5.1. We refer the reader to Appendix A for a thorough discussion of these experiments. Here, the reader will also find a theoretical comparison to other hashing schemes that are relevant to the goal of fast constant-time hashing with strong concentration bounds with high probability, i.e., bounds of the form

$$\Pr[|X-\mu| \geq t] \leq 2\exp(-\Omega(\sigma^2\mathcal{C}(t/\sigma^2))) + u^{-\gamma}.$$

5.1.4 Concluding Remarks

To sum up, in [AKK$^+$20], we have introduced a family of hash functions, tabulation-permutation hashing, which provides Chernoff-style concentration bounds for hash-based sums with a constant delay in the exponential decrease and an added error probability of $u^{-\gamma}$. We have moreover, demonstrated that tabulation-permutation is comparable in speed to some of the fastest strongly universal hashing schemes that do not provide such general concentration bounds. In the next chapter, we will provide examples of two streaming algorithms that can be sped up with the access to such a hash family.

Chapter 6

No Repetitions with Highly Concentrated Hashing

This chapter presents the results of the paper "*No Repetition: Fast Streaming with Highly Concentrated Hashing*" [ADK$^+$20] of Appendix B. What follows is an extracted and slightly modified subset of the introduction from the paper.

6.1 Introduction

To get estimators that work within a certain error bound with high probability, a common strategy is to design one that works with constant probability, and then boost the probability using independent repetitions. A classic example of this approach is the algorithm of Bar-Yossef *et al.* [BYJK$^+$02] to estimate the number of distinct elements in a stream. Using standard strongly universal hashing to process each element, we get an estimator where the probability of a too large error is, say, $1/4$. By performing r independent repetitions and taking the median of the estimators, the error probability falls exponentially in r. However, running r independent experiments increases the processing time by a factor r.

In [ADK$^+$20] we make the point that if we have a hash function with strong concentration bounds, then we get the same high probability bounds without any need for repetitions. Instead of r independent sketches, we have a single sketch that is $\Theta(r)$ times bigger, so the total space is essentially the same. However, we only apply a single hash function, processing each element in constant time regardless of r, and the overall algorithms just get simpler. Using the hashing scheme tabulation-1permutation from [AKK$^+$20] (presented in Chapter 5), we get a very fast implementation of the above streaming algorithm, suitable for online processing of high volume data streams.

To illustrate a streaming scenario where the constant in the processing time is critical, consider the Internet. Suppose we want to process packets passing through a high-end Internet router. Each application only gets very limited time to look at the packet before it is forwarded. If it is not done in time, the information is lost. Since processors and routers use some of the same technology, we never expect to have more than a few instructions available. Slowing down the Internet is typically not an option. The papers of Krishnamurthy *et al.* [KSZC03] and Thorup and Zhang [TZ12] explain in more detail how high speed hashing is necessary for their Internet traffic analysis. Incidentally, the hash function we use from [AKK$^+$20] is a bit faster than the ones from [KSZC03, TZ12], which do not provide Chernoff-style concentration bounds.

The idea is generic and can be applied to other algorithms. The paper, [ADK$^+$20] also applies it to Broder's original min-hash algorithm [Bro97] to estimate set similarity, which can now be implemented efficiently, giving the desired estimates with high probability.

6.1.1 Chebyshev vs. Chernoff for Estimating Distinct Elements and Set Similarity

Let us now be more specific about the algorithmic context. Below, we state the theoretical improvements using hashing with strong concentration bounds. We refer the reader to Appendix B for the analysis of the algorithms and for a thorough discussion of implementation and alternatives. In both the set similarity and the min-hash algorithm, we have a key universe, U, e.g., 64-bit keys, and we will use a random hash function h mapping U uniformly into $R = (0, 1]$.

Estimating the number of distinct elements The goal is to estimate the number of distinct elements in a data stream, $x_1, \ldots, x_s$, from a universe, U, to within some relative error ε with a desired small error probability δ. Let S be the set of distinct elements of the stream and assume that $|S| = n$. The algorithm from [BYJK$^+$02] uses a hash function $h : U \to (0, 1]$ and maintains the k smallest distinct hash values of the stream for some $k > 1$. The space required is thus $O(k)$, so we want k to be small. Let $x_{(k)}$ be the key having the k'th smallest hash value under h and let $h_{(k)} = h(x_{(k)})$. As in [BYJK$^+$02], we use $\hat{n} = k/h_{(k)}$ as an estimator for n (we note that [BYJK$^+$02] suggests several other estimators, but the points we will make below apply to all of them).

If h is strongly universal (see Chapter 2), the analysis from [BYJK$^+$02] shows that,

$$\Pr\left[|\hat{n} - n| > \varepsilon n\right] \leq 2/(k\varepsilon^2),$$

for $\varepsilon < 1$. With this choice of hashing, we can let let $k = 8/\varepsilon^2$ and use the median trick over $O(\log 1/\delta)$ independent repetitions of the experiments to obtain the error probability of δ.

The point made in [ADK$^+$20] is that if we replace h with a hash function which provides strong concentration bounds, we remove the need for independent repetitions. What we need is a hash function that satisfies Definition 4.1 of Chapter 4 with added error $\mathcal{E}$ when X is a random variable counting the number of keys hashing to a given *interval.* With such a hash function, the bound above gets replaced by

$$\Pr\left[|\hat{n} - n| \geq \varepsilon n\right] \leq 2\exp(-\Omega(k\varepsilon^2)) + \mathcal{E}.$$

For $\delta = \omega(\mathcal{E})$, we can just perform the experiment a single time, instead choosing $k = O(\log(1/\delta)/\varepsilon^2)$, to obtain the desired error probability of δ. Like with strongly universal hashing and the median trick, we need to store $k = O(\log(1/\delta)/\varepsilon^2)$ hash values. The big advantage is that (1) with no independent repetitions we avoid applying $r = \Theta(\log(1/\delta))$ hash functions to each key, so we basically save a factor $\Theta(\log(1/\delta))$ in speed, and (2) with independent repetitions, we are tuning the algorithm depending on ε and δ, whereas with hashing with strong concentration, we get the concentration from above for every $\varepsilon < 1$.

Estimating set-similarity We next consider Broder's [Bro97] original algorithm for Jaccard similarity $f = |A \cap B|/|A \cup B|$ between sets $A, B \subseteq U$. For the discussion, it is simpler to consider the case where we want to estimate the frequency $f = |T|/|S|$ of a subset $T \subseteq S$. Naturally, we would put $S = A \cup B$ and $T = A \cap B$ for Jaccard similarity. Again the algorithm uses a hash function $h : U \to [0, 1]$. Let $h_{(k)}$ be the kth smallest hash value from S as in the above algorithm for estimating distinct elements. For any p, let $Y^{\leq p}$ be the number of elements from T with hash value at most p. The algorithm uses $Y^{\leq h_{(k)}}$ as an estimator for fk.

Following the analysis from [Tho13a], we show in [ADK$^+$20] that

$$\Pr\left[|Y^{\leq h_{(k)}} - fk| > \varepsilon fk\right] = 2\exp(-\Omega(fk\varepsilon^2)) + O(\mathcal{E}).$$

In contrast, using strongly universal hashing, one only obtains the variance-based bound

$$\Pr\left[|Y^{\leq h_{(k)}} - fk| > \varepsilon fk\right] = O\left(\frac{1}{fk\varepsilon^2}\right).$$

Again, the difference of the two bounds shows that hashing with strong concentration removes the need for independent repetitions when aiming to obtain a desired error probability of $\delta = \omega(\mathcal{E})$.

Using Tabulation-1permutation Tabulation-1permutation *exactly* provides strong concentration bounds with added error probability $u^{-\gamma}$ on the number of keys from S that hashes to a given interval by Theorem 5.3. With this choice of hashing, we thus remove the need for independent repetitions for any $\delta = u^{-O(1)}$. Thus, using tabulation-1permutation, the two algorithms above can be efficiently implemented giving the desired estimates with high probability.

6.1.2 Concluding Remarks

We have demonstrated how hashing with strong concentration bounds leads to speedups in streaming algorithms for estimating the number of distinct elements and estimating set similarity. Such a hashing scheme essentially removes the need for making $O(\log(1/\delta))$ independent repetitions to obtain an error probability of δ. It will be interesting to investigate the quality of these improvements in practice by performing experiments. For example, one can implement the above algorithms both using (1) a fast strongly universal hash function and independent repetitions and (2) tabulation-1permutation, and compare the speed and accuracy of the two implementations.

Chapter 7

Non-Empty Bins with Simple Tabulation Hashing

This chapter presents the results of the paper "*Non-Empty Bins with Simple Tabulation Hashing*" [AT19] of Appendix C. What follows is an extracted and slightly modified subset of the introduction from the paper.

7.1 Introduction

In [AT19], we consider the balls and bins paradigm where a set $X \subseteq U$ of $|X| = n$ balls are distributed into a set of m bins according to a hash function $h : U \to [m]$. We are interested in questions relating to the distribution of $|h(X)|$, for example: What is the expected number of non-empty bins? How well is $|h(X)|$ concentrated around its mean? And what is the probability that a query ball lands in an empty bin? These questions are important in applications such as Bloom filters [Blo70] and Filter hashing [FPSS05].

When h is truly random, the situation is well understood. In this case, the probability that a bin becomes empty is $(1 - 1/m)^n$. Thus, the expected number of non-empty bins is $\mu_0 = m(1-(1-1/m)^n)$, and unsurprisingly, the number of non-empty bins turns out to be sharply concentrated around μ_0 (see for example [KMPS95] for several such concentration results). In [AT19], we study the number of non-empty bins when the hash function h is chosen to be a simple tabulation hash function. We provide estimates on the expected size of $|h(X)|$ which asymptotically match[1] those with fully random hashing on any possible input. To get a similar

[1]Here we use "asymptotically" in the classic mathematical sense to mean equal to within low order terms, not just within a constant factor.

match within the classic k-independence paradigm, we would generally need $k = \Omega((\log n)/(\log\log n))$. In [AT19] we also study to what extend $|h(X)|$ is concentrated around its mean.

Our results complement those from [PT12], which provides certain concentration bounds on the number of balls in a given bin when $n \gg m$. For example, their results imply that when $n = \omega(m \log m)$, all bins are non-empty with high probability. On the other hand, [PT12] does not provide any good bounds on the probability that a bin is non-empty when, say, $n = \Theta(m)$. Our results show that in this case, a bin is non-empty with probability $(1 - 1/m)^n \pm o(1)$, as in the fully random case. It is precisely the setting of parameters $n = \Theta(m)$ which becomes important when implementing Bloom filters [Blo70] and Filter hashing [FPSS05].

7.1.1 Main Results

We will now present some of the results of [AT19].

The expected number of non-empty bins. The first theorem of [AT19] concerns the expected number of non-empty bins when n balls are distributed into m bins. Denote by $p_0 = 1-(1-1/m)^n \le n/m$ the probability that a bin is non-empty and by $\mu_0 = mp_0$ the expected number of non-empty bins when n balls are distributed into m bins using fully random hashing.

Theorem 7.1 ([AT19])**.** *Let $X \subseteq U$ be a fixed set of $|X| = n$ balls. Let $y \in [m]$ be any bin and suppose that $h : U \to [m]$ is a simple tabulation hash function. If p denotes the probability that $y \in h(X)$ then*

$$|p - p_0| \le \frac{n^{2-1/c}}{m^2} \quad \textit{and hence} \quad |\mathbb{E}[|h(X)|] - \mu_0| \le \frac{n^{2-1/c}}{m}.$$

When n increases, the bound in Theorem 7.1 gets weaker, but then the high probability bound from [PT12] takes over. Combining the analysis of the two papers one can prove that the relative error is always small in the sense that $|\mathbb{E}[|h(X)|] - \mu_0| / \mu_0 = \tilde{O}(m^{-1/c})$, regardless of the values of m and n.

Application to Bloom filters. A Bloom filter [Blo70] is a simple data structure which space efficiently represents a set $X \subseteq U$ and supports membership queries of the form "is q in X". It uses k independent hash functions $h_0, \dots, h_{k-1} : U \to [m]$ and k arrays $A_0, \dots, A_{k-1}$, each of m bits, which are initially all 0. For each $x \in X$ and each $i \in [k]$, we update $A_i[h_i(x)] \leftarrow 1$

noting that an entry may be set to 1 several times. To answer a membership query q, we output *yes* if for each $i \in [k]$, $A_i[h_i(x)] = 1$ and *no* otherwise. If $q \in X$, we will certainly output the correct answer but if $q \notin X$ we potentially get a false positive in the case that all the bits corresponding to $(h_i(q))_{i\in[k]}$ are set to 1 by other keys in X. The probability of a false positive if $q \notin X$ is

$$\prod_{i=0}^{k-1} \Pr[h_i(q) \in h_i(X)],$$

which with fully random hashing is p_0^k. Normally, Bloom filters are proportioned for a desired low false-positive probability assuming full randomness. A slightly more general version of Theorem 7.1 (which includes the conditioning on the hash value of a query ball), implies that with filters proportioned the same way, but with the hashing implemented with as simple tabulation, we obtain a false positive probability of $(1 + o(1))p_0^k$, asymptotically the same as with fully random hashing.

Concentration of the number of non-empty bins. In [AT19] we also study the concentration of $|h(X)|$ around its mean. In the fully random setting, it was shown by Kamath et al. [KMPS95] that the concentration of $|h(X)|$ around μ_0 is sharp: For any $\lambda \geq 0$ it holds that

$$\Pr(||h(X)| - \mu_0| \geq \lambda) \leq 2\exp\left(-\frac{\lambda^2}{2\mu_0}\right),$$

which for example yields that $|h(X)| = \mu_0 \pm O(\sqrt{\mu_0 \log m})$ with high probability, that is, with probability $1 - O(m^{-\gamma})$ for any choice of $\gamma = O(1)$. Unfortunately, we cannot hope to obtain a similar concentration with simple tabulation. Aiming for high probability, the following lower bound on$|h(X)|$ is asymptotically optimal.

Theorem 7.2 ([AT19]). *Let $X \subseteq U$ be a fixed sets of $|X| = n$ keys. Let $h : U \to [m]$ be a simple tabulation hash function. Then with high probability*

$$|h(X)| \geq m\left(1 - \left(1 - \frac{1}{m}\right)^{\Omega(n)}\right)$$

It turns out however, that settling with a weaker requirement than high probability, $|h(X)|$ is somewhat concentrated around μ_0. We will not state the results in this introduction as they are of a rather technical nature. Instead, we refer the reader to Appendix C.

Applications to Filter Hashing In Filter hashing [FPSS03], we wish to store as many elements as possible of a set $X \subseteq U$ of size $|X| = n$ in d hash tables $(T_i)_{i \in [d]}$. The total number of entries in the tables is at most n and each entry can store just a single key. For $i \in [d]$, we pick independent hash functions $h_i : U \to [m_i]$ where m_i is the number of entries in T_i. The keys are allocated as follows: We first greedily store a key from $h_0^{-1}(\{y\})$ in $T_0[y]$ for each $y \in h_0(X)$. This allows us to store exactly $|h_0(X)|$ keys. Letting S_0 be the so stored keys and $X_1 = X \backslash S_0$ the remaining keys, we repeat the process, storing $|h(X_1)|$ keys in T_1 using h_1, etc.

With fully random hashing, it can be argued that at least $\Omega(\log^2(1/\varepsilon))$ filters are needed to store all but εn keys with high probability. To achieve a matching bound, [FPSS03] uses the $(12\lceil\log(4/\varepsilon)+1\rceil)$-independent PolyHash scheme from [WC81]. If we instead use simple tabulation hashing, it follows from Theorem 7.2 that we again only require $O(\log^2(1/\varepsilon))$ filters. Here, however, the hashing can be done in constant time, so we essentially save a factor of $O(\log(1/\varepsilon))$ in speed. The hash functions used in [FPSS03] depends on ε and becomes more unrealistic the smaller ε is. A further advantage of using simple tabulation is that we only need to change the number of filters, not the hashing, when ε varies.

7.1.2 Concluding Remarks

In [AT19], we have studied the distribution of the number of non-empty bins, X, when n balls are distributed into m bins using a simple tabulation hash function. We analysed both the expectation of X, μ, and to what extend X is concentrated around μ. Although, obtaining some degree of concentration, simple tabulation has some inherent limitations that prevent us from getting concentration bounds similar to those from the fully random setting. An interesting direction for future research is therefore to search for other practical hashing schemes that give a better distribution of X. For example, it is conceivable, that the bound of Theorem 7.2 should hold with high probability in the size of the universe when using the tabulation-permutation scheme from [AKK$^+$20]. Still, even with the powerful tabulation-permutation scheme, there exists sets of n balls such that when hashing to n bins, X deviates by a constant factor from μ with probability at least $|\Sigma|^{-1}$. It would be interesting to find a hashing scheme for which the distribution of non-empty bins is asymptotically the same as in the fully random setting but with an added error probability of $u^{-\gamma}$ for any $\gamma = O(1)$ — in the style of Theorem 5.2 and 5.3.

Chapter 8

(Learned) Frequency Estimation Algorithms under Zipfian Distribution

This chapter presents the results of the paper "*(Learned) Frequency Estimation Algorithms under Zipfian Distribution*" [AIV19] of Appendix D. What follows is an extracted and slightly modified subset of the introduction from the paper.

8.1 Introduction

The last few years have witnessed a rapid growth in using machine learning methods to solve "classical" algorithmic problems. For example, they have been used to improve the performance of data structures [KBC+18, Mit18], online algorithms [LV18, PSK18, GP19, Kod19, CGT+19, ADJ+20, LLMV20, Roh20, ACE+20], combinatorial optimization [KDZ+17, BDSV18, Mit20], similarity search [WLKC16, DIRW19], compressive sensing [MPB15, BJPD17] and streaming algorithms [HIKV19, IVY19, JLL+20, CGP20]. Multiple frameworks for designing and analyzing such algorithms have been proposed [ACC+11, GR17, BDV18, AKL+19]. The rationale behind this line of research is that machine learning makes it possible to adapt the behavior of the algorithms to inputs from a specific data distribution, making them more efficient or more accurate in specific applications.

In [AIV19], we focus on classic and learning-augmented streaming algorithms for frequency estimation. The frequency estimation problem is formalized as follows: given a sequence S of elements from some universe

U, construct a data structure that for any element $i \in U$ computes an estimation $\tilde{f}_i$ of f_i, the number of times i occurs in S. Many of the most popular algorithms for this problem, such as Count-Min (CM) [CM05] or Count-Sketch (CS) [CCFC02] are based on hashing. Specifically, these algorithms hash stream elements into B buckets, count the number of items hashed into each bucket, and use the bucket value as an estimate of item frequency. To improve the accuracy, the algorithms use $k > 1$ such hash functions and aggregate the answers, e.g., using the median trick. These algorithms have several useful properties: they can handle item deletions (implemented by decrementing the respective counters), and some of them (Count-Min) never underestimate the true frequencies, i.e., $\tilde{f}_i \geq f_i$.

In a recent work [HIKV19], the authors showed that the aforementioned algorithm can be improved by augmenting them with machine learning. Their approach is as follows: During the training phase, they construct a classifier (neural network) to detect whether an element is “heavy” (e.g., whether f_i is among top k frequent items). After such a classifier is trained, they scan the input stream, and apply the classifier to each element i. If the element is predicted to be heavy, it is allocated a unique bucket, so that an exact value of f_i is computed. Otherwise, the element is forwarded to a “standard” hashing data structure $\mathcal{C}$, e.g., CM or CS. To estimate $\tilde{f}_i$, the algorithm either returns the exact count f_i (if i is allocated a unique bucket) or an estimate provided by the data structure $\mathcal{C}$. An empirical evaluation, on networking and query log data sets, shows that this approach can reduce the overall estimation error.

Their paper also presents a preliminary analysis of the algorithm. Under the common assumption that the frequencies follow the Zipfian law, i.e., $f_i \propto 1/i$, for $i = 1, \ldots, n$ for some n, and further that item i is queried with probability proportional to its frequency, the expected error incurred by the learning-augmented version of CM is shown to be asymptotically lower than that of the “standard” CM. However, the exact magnitude of the gap between the error incurred by the learned and standard CM algorithms was left as an open problem. Furthermore, no analysis was presented for CS.

8.1.1 Main Results

In [AIV19], we resolve the aforementioned questions left open in [HIKV19]. Assuming that the frequencies follow a Zipfian law, we show:

- An asymptotically tight bound of $\Theta(\frac{k \log(kn/B)}{B})$ for the expected error incurred by the CM algorithm with k hash functions and a total of

B buckets. Together with a prior bound for Learned CM (Table 8.1), this shows that learning-augmentation improves the error of CM by a factor of $\Theta(\log(n)/\log(n/B))$ if the heavy hitter oracle is perfect.

- The first error bounds for CS and Learned CS (see Table 8.1). In particular, we show that for Learned CS, a single hash function as in [HIKV19] leads to an asymptotically optimal error bound, improving over standard CS by a factor of $\Theta(\log(n)/\log(n/B))$ (same as CM).

	$k = 1$	$k > 1$
Count-Min (CM)	$\Theta\left(\frac{\log n}{B}\right)$ [HIKV19]	$\Theta\left(\frac{k\cdot\log(\frac{kn}{B})}{B}\right)$
Learned Count-Min (L-CM)	$\Theta\left(\frac{\log^2(\frac{n}{B})}{B\log n}\right)$ [HIKV19]	$\Omega\left(\frac{\log^2(\frac{n}{B})}{B\log n}\right)$ [HIKV19]
Count-Sketch (CS)	$\Theta\left(\frac{\log B}{B}\right)$	$\Omega\left(\frac{k^{1/2}}{B\log k}\right)$ and $O\left(\frac{k^{1/2}}{B}\right)$
Learned Count-Sketch (L-CS)	$\Theta\left(\frac{\log\frac{n}{B}}{B\log n}\right)$	$\Omega\left(\frac{\log\frac{n}{B}}{B\log n}\right)$

Table 8.1: This table summarizes our and previously known results on the expected frequency estimation error of Count-Min (CM), Count-Sketch (CS) and their learned variants (i.e., L-CM and L-CS) that use k functions and overall space $k \times \frac{B}{k}$ under Zipfian distribution. For CS, we assume that k is odd (so that the median of k values is well defined).

The results on L-CS in Table 8.1 assume that the heavy hitter oracle is *perfect*, i.e., that it makes no mistakes when classifying the heavy items. We complement the results with an analysis of L-CS when the heavy hitter oracle may err with probability at most δ on each item. As δ varies in $[0, 1]$, we obtain a smooth trade-off between the performance of L-CS and its classic counterpart. Specifically, as long as $\delta = O(1/\log B)$, the bounds are as good as with a perfect heavy hitter oracle.

In addition to clarifying the gap between the learned and standard variants of popular frequency estimation algorithms, our results provide interesting insights about the algorithms themselves. For example, for both CM and CS, the number of hash functions k is often selected to be $\Theta(\log n)$, in order to guarantee that *every* frequency is estimated up to a certain error bound. In contrast, we show that if instead the goal is to bound the *expected* error, then setting k to a constant (strictly greater than 1) leads to the asymptotic optimal performance. The same phenomenon holds not only for a Zipfian query distribution but in fact for an arbitrary distribution on the queries.

8.1.2 Concluding Remarks

We have shown how access to an oracle that can predict whether an item of a data stream is a heavy hitter, can be used to obtain improved performances of the streaming algorithms Count-Min and Count-Sketch. The results of [AIV19] are specific to input that follows a Zipfian distribution. It is interesting whether one can obtain similar improvements for arbitrary distributions, perhaps parametrized in some properties of the distribution. The idea of augmenting classic algorithm with machine learning oracles is relatively new. Taking a step back, it is therefore interesting to study in which other settings classic algorithms can be improved using advice from a (machine learning) oracle which predicts some property of the input data.

Chapter 9

One-Way Trail Orientations

This chapter presents the results of the paper "*One-Way Trail Orientations*" [AHHR18] of Appendix E. What follows is an extracted and slightly modified subset of the introduction from the paper.

9.1 Introduction and Motivation

Suppose that the mayor of a small town decides that it will be beneficial to make all the streets of the town one-way[1]. Naturally, she wants to ensure that it is remains possible to get from any place to any other place without violating the orientations of the streets, and she asks herself whether this is possible. Modelling the road network as a graph, we can similarly ask when the edges of a graph, G, can be oriented such that the resulting directed graph is strongly connected. Robbins' theorem [Rob39] asserts that this can be done exactly when G is 2-edge connected. Moreover, if some edges of G are already oriented, the generalised Robbins' theorem by Boesch [BT80] asserts it can be done exactly when the corresponding "mixed" graph is strongly connected and the underlying graph is bridgeless. In the town analogy, this would correspond to some streets initially being one-way.

The model above assumes that every street of the city corresponds to exactly one edge in the graph. There's hardly a city in the world where this assumption holds and therefore a more natural assumption is that every street corresponds to a *trail* (informally, a potentially self-crossing path) in the graph and that the edges of each trail must be oriented consistently[2].

[1]Her motivation is that the long and narrow streets cause big inconveniences when two cars unexpectedly meet.

[2]This version of the problem was given to us through personal communication with

In [AHHR18], we consider such graphs, the edges of which are partitioned into trails. We prove that the trails can be oriented making the resulting directed graph strongly connected exactly if the initial graph is 2-edge connected (note that this is the same condition as in Robbins' theorem).

Not only do we show that the strong trail orientation problem in undirected 2-edge connected graphs always has a solution, we also provide a linear time algorithm for finding such an orientation. In doing so, we use an interesting combination of techniques that allow us to reduce to a graph with a number of 3-edge connected components that is linear in the number of edges. Using that the average size of these components is constant and that we can piece together solutions for the individual components we obtain an efficient algorithm.

Finally, we consider the generalised Robbins' theorem in this new setting by allowing some initially oriented edges and supposing that the remaining edges are partitioned into trails. We show that if each cut, (V_1, V_2), in the graph has either at least two undirected edges going between V_1 and V_2 or at least one directed edge in each direction, then it is possible to orient the trails making the resulting graph strongly connected. In fact, we show that if this condition is satisfied we may start the trail orientation process by orienting an *arbitrary* trail in an *arbitrary* direction. Although this condition is not necessary it does give a simple algorithm for finding a trail orientation if it exists. Indeed, initially the graph may contain undirected edges that are forced in one direction by some cut. Here, an edge e is *forced* if it is the single undirected edge of some cut, (V_1, V_2), and all the directed edges of the cut go from V_1 to V_2. Aiming for a strong orientation, we are then clearly forced to orient e from V_2 to V_1. For finding a trail orientation (if it exists), we can thus orient forced trails in their forced direction. If there are no forced trails we orient and arbitrary trail in an arbitrary direction.

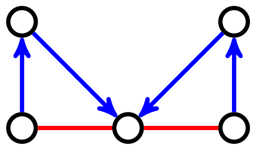

Figure 9.1: The graph is strongly connected and the underlying graph is 2-edge connected, but irrespective of the choice of orientation of the red trail, the graph will no longer be strongly connected.

Note that in the setting with mixed graph, the feasibility depends on

Professor Robert E. Tarjan.

the trail decomposition which is not the case for the other results. That the condition from the generalised Robbins' theorem is not sufficient can be seen from Figure 9.1.

9.1.1 Concluding Remarks

In [AHHR18], we presented a simple criterion for when the one-way trail orientation problem can be solved in an undirected graph. We further provided a linear time algorithm for finding a strongly connected trail orientation when it exists. We mention two directions for future work. First, our linear time algorithm for finding strongly connected trail orientations only works for undirected graphs and it does not seem to generalise to mixed graphs. It would be interesting to know whether there exists a linear time algorithm for mixed graphs too. If so, this would complete the picture of how fast any variant of the trail orientation problem can be solved. Second, our condition for when the trail orientation problem can be solved for mixed graphs is not necessary. It would be interesting to know whether there is a simple necessary and sufficient criterion like in the undirected setting.

Bibliography

[AAKR19] Anders Aamand, Mikkel Abrahamsen, Jakob B. T. Knudsen, and Peter M. R. Rasmussen. Classifying convex bodies by their contact and intersection graphs. *CoRR*, 2019. `https://arxiv.org/abs/1902.01732`. In submission.

[AAT20] Anders Aamand, Mikkel Abrahamsen, and Mikkel Thorup. Disks in curves of bounded convex curvature. *The American Mathematical Monthly*, 127(7):579–593, 2020.

[ABKU99] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations. *SIAM Journal of Computing*, 29(1):180–200, 1999. Announced at STOC'94.

[ACC$^+$11] Nir Ailon, Bernard Chazelle, Kenneth L Clarkson, Ding Liu, Wolfgang Mulzer, and C Seshadhri. Self-improving algorithms. *SIAM Journal on Computing*, 40(2):350–375, 2011.

[ACE$^+$20] Antonios Antoniadis, Christian Coester, Marek Elias, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. *arXiv preprint arXiv:2003.02144*, 2020.

[ADJ$^+$20] Spyros Angelopoulos, Christoph Dürr, Shendan Jin, Shahin Kamali, and Marc Renault. Online computation with untrusted advice. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[ADK$^+$20] Anders Aamand, Debarati Das, Evangelos Kipouridis, Jakob Bæk Tejs Knudsen, Peter M. R. Rasmussen, and Mikkel Thorup. No repetition: Fast streaming with highly concentrated hashing. *CoRR*, 2020. `arxiv.org/abs/2004.01156`. In submission.

[AHHR18] Anders Aamand, Niklas Hjuler, Jacob Holm, and Eva Rotenberg. One-way trail orientations. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, pages 6:1–6:13, 2018.

[AIV19] Anders Aamand, Piotr Indyk, and Ali Vakilian. (learned) frequency estimation algorithms under zipfian distribution. *CoRR*, 2019. `https://arxiv.org/abs/1908.05198`. In submission.

[AKK$^+$20] Anders Aamand, Jakob Bæk Tejs Knudsen, Mathias Bæk Tejs Knudsen, Peter M. R. Rasmussen, and Mikkel Thorup. Fast hashing with strong concentration bounds. *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, 2020.

[AKL$^+$19] Daniel Alabi, Adam Tauman Kalai, Katrina Ligett, Cameron Musco, Christos Tzamos, and Ellen Vitercik. Learning to prune: Speeding up repeated computations. In *Conference on Learning Theory*, 2019.

[AKŁ$^+$20] Anders Aamand, Adam Karczmarz, Jakub Łacki, Nikos Parotsidis, Peter M. R. Rasmussen, and Mikkel Thorup. Randomized decremental connectivity in linear time for less dense graphs, 2020. In submission.

[AKT18] Anders Aamand, Mathias B. T. Knudsen, and Mikkel Thorup. Power of d choices with simple tabulation. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, pages 5:1–5:14, 2018.

[AT19] Anders Aamand and Mikkel Thorup. Non-empty bins with simple tabulation hashing. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 2498–2512, 2019.

[BDSV18] Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In *International Conference on Machine Learning*, pages 353–362, 2018.

[BDV18] Maria-Florina Balcan, Travis Dick, and Ellen Vitercik. Dispersion for data-driven algorithm design, online learning, and private optimization. In *2018 IEEE 59th Annual Symposium* *on Foundations of Computer Science (FOCS)*, pages 603–614. IEEE, 2018.

[Ben62] George Bennett. Probability inequalities for the sum of independent random variables. *Journal of the American Statistical Association*, 57(297):33–45, 1962.

[Ber24] Sergei Natanovich Bernstein. On a modification of Chebyshev's inequality and of the error formula of Laplace. *Ann. Sci. Inst. Sav. Ukraine, Sect. Math.*, (1):38–49, 1924.

[BJPD17] Ashish Bora, Ajil Jalal, Eric Price, and Alexandros G Dimakis. Compressed sensing using generative models. In *International Conference on Machine Learning*, pages 537–546, 2017.

[Blo70] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.

[Bro97] Andrei Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences (SEQUENCES)*, pages 21–29, 1997.

[BT80] Frank Boesch and Ralph Tindell. Robbins's theorem for mixed multigraphs. *The American Mathematical Monthly*, 87(9):716–719, 1980.

[BYJK$^+$02] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 1–10, 2002.

[CCFC02] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.

[CGP20] Edith Cohen, Ofir Geri, and Rasmus Pagh. Composable sketches for functions of frequencies: Beyond the worst case. *arXiv preprint arXiv:2004.04772*, 2020.

[CGT$^+$19] Shuchi Chawla, Evangelia Gergatsouli, Yifeng Teng, Christos Tzamos, and Ruimin Zhang. Learning optimal search algorithms from data. *arXiv preprint arXiv:1911.01632*, 2019.

[Che52] Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23(4):493–507, 1952.

[CM05] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[CPT15] Tobias Christiani, Rasmus Pagh, and Mikkel Thorup. From independence to expansion and back again. In *Proceedings of the 47rd ACM Symposium on Theory of Computing (STOC)*, 2015.

[Die96] Martin Dietzfelbinger. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In *Proceedings of the 13th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 569–580, 1996.

[DIRW19] Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Learning sublinear-time indexing for nearest neighbor search. *arXiv preprint arXiv:1901.08544*, 2019.

[DKRT15] Søren Dahlgaard, Mathias B. T. Knudsen, Eva Rotenberg, and Mikkel Thorup. Hashing for statistics over k-partitions. In *Proc. 56th Symposium on Foundations of Computer Science*, FOCS, pages 1292–1310, 2015.

[DKRT16] Søren Dahlgaard, Mathias B. T. Knudsen, Eva Rotenberg, and Mikkel Thorup. The power of two choices with simple tabulation. In *Proc. 27. ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 1631–1642, 2016.

[DM92] Martin Dietzfelbinger and Friedhelm Meyer auf der Heide. Dynamic hashing in real time. In *Informatik, Festschrift zum 60. Geburtstag von Günter Hotz*, pages 95–119. 1992.

[DR09] Martin Dietzfelbinger and Michael Rink. Applications of a splitting trick. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 354–365, 2009.

[Dum56] A. I. Dumey. Indexing for rapid random access memory systems. *Computers and Automation*, 5(12):6–9, 1956.

[DW03] Martin Dietzfelbinger and Philipp Woelfel. Almost random graphs with simple hash functions. In *Proceedings of the 25th ACM Symposium on Theory of Computing (STOC)*, pages 629–638, 2003.

[DW07] Martin Dietzfelbinger and Christoph Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. *Theor. Comput. Sci.*, 380:47–68, 06 2007.

[FPSS03] Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul G. Spirakis. Space efficient hash tables with worst case constant access time. In *Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '03, pages 271–282, 2003.

[FPSS05] Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul Spirakis. Space efficient hash tables with worst case constant access time. *Theory of Computing Systems*, 38(2):229–248, Feb 2005.

[GP19] Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2319–2327, 2019.

[GR17] Rishi Gupta and Tim Roughgarden. A pac approach to application-specific algorithm selection. *SIAM Journal on Computing*, 46(3):992–1017, 2017.

[HIKV19] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In *International Conference on Learning Representations*, 2019.

[HT01] Torben Hagerup and Torsten Tholey. Efficient minimal perfect hashing in nearly minimal space. In *Proceedings of the 18th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 317–326, 2001.

[IVY19] Piotr Indyk, Ali Vakilian, and Yang Yuan. Learning-based low-rank approximations. In *Advances in Neural Information Processing Systems*, pages 7400–7410, 2019.

[JLL+20] Tanqiu Jiang, Yi Li, Honghao Lin, Yisong Ruan, and David P. Woodruff. Learning-augmented data stream algorithms. In *International Conference on Learning Representations*, 2020.

[KBC+18] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, pages 489–504, 2018.

[KDZ+17] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017.

[KMPS95] Anil Kamath, Rajeev Motwani, Krishna V. Palem, and Paul G. Spirakis. Tail bounds for occupancy and the satisfiability threshold conjecture. *Random Struct. Algorithms*, 7(1):59–80, 1995.

[Knu63] Donald E. Knuth. Notes on open addressing. Unpublished memorandum. See `http://citeseer.ist.psu.edu/knuth63notes.html`, 1963.

[Knu98] Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2Nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[Kod19] Rohan Kodialam. Optimal algorithms for ski rental with soft machine-learned predictions. *arXiv preprint arXiv:1903.00092*, 2019.

[KSZC03] Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang, and Yan Chen. Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the 3rd Internet Measurement Conference (IMC)*, pages 234–247, 2003.

[LLMV20] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1859–1877. SIAM, 2020.

[LV18] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *International Conference on Machine Learning*, pages 3302–3311, 2018.

[Mit18] Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *Advances in Neural Information Processing Systems*, pages 464–473, 2018.

[Mit20] Michael Mitzenmacher. Scheduling with predictions and the price of misprediction. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[MPB15] Ali Mousavi, Ankit B Patel, and Richard G Baraniuk. A deep learning approach to structured signal recovery. In *Communication, Control, and Computing (Allerton), 2015 53rd Annual Allerton Conference on*, pages 1336–1343. IEEE, 2015.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[PP08] Anna Pagh and Rasmus Pagh. Uniform hashing in constant time and optimal space. *SIAM Journal on Computing*, 38(1):85–96, 2008.

[PSK18] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In *Advances in Neural Information Processing Systems*, pages 9661–9670, 2018.

[PT12] Mihai Pătraşcu and Mikkel Thorup. The power of simple tabulation-based hashing. *Journal of the ACM*, 59(3):Article 14, 2012. Announced at STOC'11.

[PT13] Mihai Pătraşcu and Mikkel Thorup. Twisted tabulation hashing. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 209–228, 2013.

[Rob39] H. E. Robbins. A theorem on graphs, with an application to a problem of traffic control. *The American Mathematical Monthly*, 46(5):281–283, 1939.

[Roh20] Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1834–1845. SIAM, 2020.

[Sie04] Alan Siegel. On universal classes of extremely random constant-time hash functions. *SIAM Journal on Computing*, 33(3):505–543, 2004. Announced at FOCS'89.

[Tho13a] Mikkel Thorup. Bottom-k and priority sampling, set similarity and subset sums with minimal independence. In *Proc. 45th ACM Symposium on Theory of Computing (STOC)*, 2013.

[Tho13b] Mikkel Thorup. Simple tabulation, fast expanders, double tabulation, and high independence. In *54th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 90–99, 2013.

[TZ12] Mikkel Thorup and Yin Zhang. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM Journal on Computing*, 41(2):293–331, 2012. Announced at SODA'04 and ALENEX'10.

[WC81] Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.*, 22(3):265–279, 1981.

[WLKC16] Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. Learning to hash for indexing big data - a survey. *Proceedings of the IEEE*, 104(1):34–57, 2016.

[Zob70] Albert Lindsey Zobrist. A new hashing method with application for game playing. Technical Report 88, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1970.

Part II

Appendix

Appendix A

Fast Hashing with Strong Concentration Bounds

Fast Hashing with Strong Concentration Bounds[*]

Anders Aamand[†] Jakob B. T. Knudsen[†] Mathias B. T. Knudsen[‡]
Peter M. R. Rasmussen[†] Mikkel Thorup[†]

August 31, 2020

Abstract

Previous work on tabulation hashing by Pătraşcu and Thorup from STOC'11 on simple tabulation and from SODA'13 on twisted tabulation offered Chernoff-style concentration bounds on hash based sums, e.g., the number of balls/keys hashing to a given bin, but under some quite severe restrictions on the expected values of these sums. The basic idea in tabulation hashing is to view a key as consisting of $c = O(1)$ characters, e.g., a 64-bit key as $c = 8$ characters of 8-bits. The character domain Σ should be small enough that character tables of size $|\Sigma|$ fit in fast cache. The schemes then use $O(1)$ tables of this size, so the space of tabulation hashing is $O(|\Sigma|)$. However, the concentration bounds by Pătraşcu and Thorup only apply if the expected sums are $\ll |\Sigma|$.

To see the problem, consider the very simple case where we use tabulation hashing to throw n balls into m bins and want to analyse the number of balls in a given bin. With their concentration bounds, we are fine if $n = m$, for then the expected value is 1. However, if $m = 2$, as when tossing n unbiased coins, the expected value $n/2$ is $\gg |\Sigma|$ for large data sets, e.g., data sets that do not fit in fast cache.

To handle expectations that go beyond the limits of our small space, we need a much more advanced analysis of simple tabulation, plus a new tabulation technique that we call *tabulation-permutation* hashing which is at most twice as slow as simple tabulation. No other hashing scheme of comparable speed offers similar Chernoff-style concentration bounds.

[*]An extended abstract appeared at the 52nd Annual ACM Symposium on Theory of Computing (STOC'20).
[†]Basic Algorithms Research Copenhagen (BARC), University of Copenhagen.
[‡]SupWiz ApS.

Contents

1 Introduction

Chernoff's concentration bounds [12] date back to the 1950s but bounds of this types go even further back to Bernstein in the 1920s [7]. Originating from the area of statistics they are now one of the most basic tools of randomized algorithms [36]. A canonical form considers the sum $X = \sum_{i=1}^{n} X_i$ of independent random variables $X_1, \ldots, X_n \in [0,1]$. Writing $\mu = \mathbb{E}[X]$ it holds for every $\varepsilon \geq 0$ that

$$\Pr[X \geq (1+\varepsilon)\mu] \leq \exp(-\mu\, \mathcal{C}(\varepsilon)) \qquad \left[\leq\ \exp(-\varepsilon^2\mu/3) \text{ for } \varepsilon \leq 1\right], \tag{1}$$

$$\Pr[X \leq (1-\varepsilon)\mu] \leq \exp(-\mu\, \mathcal{C}(-\varepsilon)) \qquad \left[\leq\ \exp(-\varepsilon^2\mu/2) \text{ for } \varepsilon \leq 1\right]. \tag{2}$$

Here $\mathcal{C} : (-1, \infty) \to [0, \infty)$ is given by $\mathcal{C}(x) = (x+1)\ln(x+1) - x$, so $\exp(-\mathcal{C}(x)) = \frac{e^x}{(1+x)^{(1+x)}}$. Textbook proofs of (1) and (2) can be found in [36, §4][1]. Writing $\sigma^2 = \mathrm{Var}[X]$, a more general bound is

$$\Pr[|X - \mu| \geq t] \leq 2\exp(-\sigma^2\mathcal{C}(t/\sigma^2)) \qquad \left[\leq\ 2\exp(-(t/\sigma)^2/3) \text{ for } t \leq \sigma^2\right]. \tag{3}$$

Since $\sigma^2 \leq \mu$ and $\mathcal{C}(-\varepsilon) \leq 1.5\,\mathcal{C}(\varepsilon)$ for $\varepsilon \leq 1$, (3) is at least as good as (1) and (2), up to constant factors, and often better. In this work, we state our results in relation to (3), known as Bennett's inequality [6].

Hashing is another fundamental tool of randomized algorithms dating back to the 1950s [23]. A random hash function, $h : U \to R$, assigns a hash value, $h(x) \in R$, to every key $x \in U$. Here both U and R are typically bounded integer ranges. The original application was hash tables with chaining where x is placed in bin $h(x)$, but today, hash functions are ubiquitous in randomized algorithms. For instance, they play a fundamental role in streaming and distributed settings where a system uses a hash function to coordinate the random choices for a given key. In most applications, we require concentration bounds for one of the following cases of increasing generality.

1. Let $S \subseteq U$ be a set of balls and assign to each ball, $x \in S$, a weight, $w_x \in [0,1]$. We wish to distribute the balls of S into a set of bins $R = [m] = \{0, 1, \ldots, m-1\}$. For a bin, $y \in [m]$, $X = \sum_{x \in S} w_x \cdot [h(x) = y]$ is then the total weight of the balls landing in bin y.

2. We may instead be interested in the total weight of the balls with hash values in the interval $[y_1, y_2)$ for some $y_1, y_2 \in [m]$, that is, $X = \sum_{x \in S} w_x \cdot [y_1 \leq h(x) < y_2]$.

3. More generally, we may consider a fixed *value function* $v \colon U \times R \to [0,1]$. For each key $x \in U$, we define the random variable $X_x = v(x, h(x))$, where the randomness of X_x stems from that of $h(x)$. We write $X = \sum_{x \in U} v(x, h(x))$ for the sum of these values.

To exemplify applications, the first case is common when trying to allocate resources; the second case arises in streaming algorithms; and the third case handles the computation of a complicated statistic, X, on incoming data. In each case, we wish the variable X to be concentrated around its mean, $\mu = \mathbb{E}[X]$, according to the Chernoff-style bound of (3). If we had fully random hashing, this would indeed be the case. However, storing a fully random hash function is infeasible. The goal of this paper is to obtain such concentration with a practical constant-time hash function. More specifically, we shall construct hash functions that satisfy the following definition when X is a random variable as in one of the three cases above.

Definition 1 (Strong Concentration)**.** Let $h \colon [u] \to [m]$ be a hash function, $S \subseteq [u]$ be a set of hash keys of size $n = |S|$, and $X = X(h, S)$ be a random variable, which is completely determined by h and S. Denote by $\mu = \mathbb{E}[X]$ and $\sigma^2 = \mathrm{Var}[X]$ the expectation and variance of X. We say that X is *strongly concentrated with added error probability* $f(u, n, m)$ if for every $t > 0$,

$$\Pr[|X - \mu| \geq t] \leq O\left(\exp\left(-\Omega(\sigma^2\mathcal{C}(t/\sigma^2))\right)\right) + f(u, n, m). \tag{4}$$

Throughout the paper we shall prove properties of random variables that are determined by some hash function. In many cases, we would like these properties to continue to hold while conditioning the hash function on its value on some hash key.

[1]The bounds in [36, §4] are stated as working only for $X_i \in \{0,1\}$, but the proofs can easily handle any $X_i \in [0,1]$.

Definition 2 (Query Invariant)**.** Let $h\colon [u] \to [m]$ be a hash function, let $X = X(h)$ be a random variable determined by the outcome of h, and suppose that some property T is true of X. We say that the property is *query invariant* if whenever we choose $x \in [u]$ and $y \in [m]$ and consider the hash function $h' = (h|h(x) = y)$, i.e., h conditioned on $h(x) = y$, property T is true of $X' = X(h')$.

Remark. For example, consider the case (1) from above. We are interested in the random variable $X = \sum_{x\in S} w_x \cdot [h(x) = y]$. Suppose that for every choice of weights, $(w_x)_{x\in S}$, X is strongly concentrated and that this concentration is query invariant. Let $x_0 \in [u]$ be a distinguished query key. Then since for every $y_0 \in [m]$, the hash function $h' = (h|h(x_0) = y_0)$ satisfies that $X' = \sum_{x\in S} w_x \cdot [h'(x) = y_0]$ is strongly concentrated, it follows that $X'' = \sum_{x\in S} w_x \cdot [h(x) = h(x_0)]$ is strongly concentrated. Thus, h allows us to get Chernoff-style concentration on the weight of the balls landing in the same bin as x_0.

This may be generalized such that in the third case from above, the weight function may be chosen as a function of $h(x_0)$. Thus, the property of being query invariant is very powerful. It is worth noting that the constants of the asymptotics may change when conditioning on a query. Furthermore, the expected value and variance of X' may differ from that of X, but this is included in the definition.

One way to achieve Chernoff-style bounds in all of the above cases is through the classic k-independent hashing framework of Wegman and Carter [48]. The random hash function $h : U \to R$ is k-independent if for any k distinct keys $x_1, \dots, x_k \in U$, $(h(x_1), \dots, h(x_k))$ is uniformly distributed in R^k. Schmidt and Siegel [43] have shown that with k-independence, the above Chernoff bounds hold with an added error probability decreasing exponentially in k. Unfortunately, a lower bound by Siegel [44] implies that evaluating a k-independent hash function takes $\Omega(k)$ time unless we use a lot of space (to be detailed later).

Pătraşcu and Thorup have shown that Chernoff-style bounds can be achieved in constant time with tabulation based hashing methods; namely simple tabulation [38] for the first case described above and twisted tabulation [41] for all cases. However, their results suffer from some severe restrictions on the expected value, μ, of the sum. More precisely, the speed of these methods relies on using space small enough to fit in fast cache, and the Chernoff-style bounds [38, 41] all require that μ is much smaller than the space used. For larger values of μ, Pătraşcu and Thorup [38, 41] offered some weaker bounds with a deviation that was off by several logarithmic factors. It can be shown that some of these limitations are inherent to simple and twisted tabulation. For instance, they cannot even reliably distribute balls into $m = 2$ bins, as described in the first case above, if the expected number of balls in each bin exceeds the space used.

In this paper, we construct and analyse a new family of fast hash functions *tabulation-permutation* hashing that has Chernoff-style concentration bounds like (3) without any restrictions on μ. This generality is important if building a general online system with no knowledge of future input. Later, we shall give concrete examples from streaming where μ is in fact large. Our bounds hold for all of the cases described above and all possible inputs. Furthermore, tabulation-permutation hashing is an order of magnitude faster than any other known hash function with similar concentration bounds, and almost as fast as simple and twisted tabulation. We demonstrate this both theoretically and experimentally. Stepping back, our main theoretical contribution lies in the field of analysis of algorithms, and is in the spirit of Knuth's analysis of linear probing [29], which shows strong theoretical guarantees for a very practical algorithm. We show that tabulation-permutation hashing has strong theoretical Chernoff-style concentration bounds. Moreover, on the practical side, we perform experiments, summarized in Table 1, demonstrating that it is comparable in speed to some of the fastest hash functions in use, none of which provide similar concentration bounds.

When talking about hashing in constant time, the actual size of the constant is of crucial importance. First, hash functions typically execute the same instructions on all keys, in which case we always incur the worst-case running time. Second, hashing is often an inner-loop bottle-neck of data processing. Third, hash functions are often applied in time-critical settings. Thus, even speedups by a multiplicative constant are very impactful. As an example from the Internet, suppose we want to process packets passing through a high-end Internet router. Each application only gets very limited time to look at the packet before it is forwarded. If it is not done in time, the information is lost. Since processors and routers use some of the same technology, we never expect to have more than a few instructions available. Slowing down the Internet is typically not an option. The papers of Krishnamurthy et al. [30] and Thorup and Zhang [47] explain in more detail how high speed hashing is necessary for their Internet traffic analysis. Incidentally, our hash function is a bit faster than the ones from [30, 47], which do not provide Chernoff-style concentration bounds.

Concrete examples of the utility of our new hash-family may be found in [1]. In [1] it is shown that some classic streaming algorithms enjoy very substantial speed-ups when implemented using tabulation-permutation hashing; namely the original similarity estimation of Broder [8] and the estimation of distinct elements of Bar-Yossef et al. [5]. The strong concentration bounds makes the use of independent repetitions unnecessary, allowing the implementations of the algorithms to be both simpler and faster. We stress that in high-volume streaming algorithms, speed is of critical importance.

Tabulation-permutation hashing builds on top of simple tabulation hashing, and to analyse it, we require a new and better understanding of the behaviour and inherent limitations of simple tabulation, which we proceed to describe. Afterwards we break these limitations by introducing our new powerful tabulation-permutation hashing scheme.

1.1 Simple Tabulation Hashing

Simple tabulation hashing dates back to Zobrist [49]. In simple tabulation hashing, we consider the key domain U to be of the form $U = \Sigma^c$ for some character alphabet Σ and $c = O(1)$, such that each key consists of c characters of Σ. Let $m = 2^\ell$ be given and identify $[m] = \{0, 1, \ldots, m-1\}$ with $[2]^\ell$. A simple tabulation hash function, $h\colon \Sigma^c \to [m]$, is then defined as follows. For each $j \in \{1, \ldots, c\}$ store a fully random character table $h_j\colon \Sigma \to [m]$ mapping characters of the alphabet Σ to ℓ-bit hash values. To evaluate h on a key $x = (x_1, \ldots, x_c) \in \Sigma^c$, we compute $h(x) = h_1(x_1) \oplus \cdots \oplus h_c(x_c)$, where $\oplus$ denotes bitwise XOR – an extremely fast operation. With character tables in cache, this scheme is the fastest known 3-independent hashing scheme [38]. We will denote by $u = |U|$ the size of the key domain, identify $U = \Sigma^c$ with $[u]$, and always assume the size of the alphabet, $|\Sigma|$, to be a power of two. For instance, we could consider 32-bit keys consisting of four 8-bit characters. For a given computer, the best choice of c in terms of speed is easily determined experimentally once and for all, and is independent of the problems considered.

Let $S \subseteq U$ and consider hashing $n = |S|$ weighted balls or keys into $m = 2^\ell$ bins using a simple tabulation function, $h\colon [u] \to [m]$, in line with the first case mentioned above. We shall prove the theorem below.

Theorem 1. *Let $h\colon [u] \to [m]$ be a simple tabulation hash function with $[u] = \Sigma^c$, $c = O(1)$. Let $S \subseteq [u]$ be given of size $n = |S|$ and assign to each key/ball $x \in S$ a weight $w_x \in [0, 1]$. Let $y \in [m]$, and define $X = \sum_{x \in S} w_x \cdot [h(x) = y]$ to be the total weight of the balls hashing to bin y. Then for any constant $\gamma > 0$, X is strongly concentrated with added error probability n/m^γ, where the constants of the asymptotics are determined solely by c and γ. Furthermore, this concentration is query invariant.*

In Theorem 1, we note that the expectation, $\mu = \mathbb{E}[X]$, and the variance, $\sigma^2 = \mathrm{Var}[X]$, are the same as if h were a fully random hash function since h is 3-independent. This is true even when conditioning on the hash value of a query key having a specific value. The bound provided by Theorem 1 is therefore the same as the variance based Chernoff bound (3) except for a constant delay in the exponential decrease and an added error probability of n/m^γ. Since $\sigma^2 \leq \mu$, Theorem 1 also implies the classic one-sided Chernoff bounds (1) and (2), again with the constant delay and the added error probability as above, and a leading factor of 2.

Pǎtraşcu and Thorup [38] proved an equivalent probability bound, but without weights, and, more importantly, with the restriction that the number of bins $m \geq n^{1-1/(2c)}$. In particular, this implies the restriction $\mu \leq |\Sigma|^{1/2}$. Our new bound gives Chernoff-style concentration with high probability in n for any $m \geq n^\varepsilon$, $\varepsilon = \Omega(1)$. Indeed, letting $\gamma' = (\gamma + 1)/\varepsilon$, the added error probability becomes $n/m^{\gamma'} \leq 1/n^\gamma$.

However, for small m the error probability n/m^γ is prohibitive. For instance, unbiased coin tossing, corresponding to the case $m = 2$, has an added error probability of $n/2^\gamma$ which is useless. In Section 8, we will show that it is inherently impossible to get good concentration bounds using simple tabulation hashing when the number of bins m is small. To handle all instances, including those with few bins, and to support much more general Chernoff bounds, we introduce a new hash function: tabulation-permutation hashing.

1.2 Tabulation-Permutation Hashing

We start by defining *tabulation-permutation hashing* from Σ^c to Σ^d with $c, d = O(1)$. A tabulation-permutation hash function $h\colon \Sigma^c \to \Sigma^d$ is given as a composition, $h = \tau \circ g$, of a simple tabulation hash function $g\colon \Sigma^c \to \Sigma^d$ and a permutation $\tau\colon \Sigma^d \to \Sigma^d$. The permutation is a coordinate-wise fully random permutation: for each $j \in \{1, \ldots, d\}$, pick a uniformly random character permutation $\tau_j : \Sigma \to \Sigma$. Now, $\tau = (\tau_1, \ldots, \tau_d)$ in the sense that for $z = (z_1, \ldots, z_d) \in \Sigma^d$, $\tau(z) = (\tau_1(z_1), \ldots, \tau_d(z_d))$. In words, a tabulation-permutation hash function hashes c characters to d characters using simple tabulation, and then randomly permutes each of the d output characters. As is, tabulation-permutation hash functions yield values in Σ^d, but we will soon see how we can hash to $[m]$ for any $m \in \mathbb{N}$.

If we precompute tables $T_i\colon \Sigma \to \Sigma^d$, where

$$T_i(z_i) = \left(\overbrace{0, \ldots, 0}^{i-1}, \tau_i(z_i), \overbrace{0, \ldots, 0}^{d-i}\right), \quad z_i \in \Sigma,$$

then $\tau(z_1, \ldots, z_d) = T_1(z_1) \oplus \cdots \oplus T_d(z_d)$. Thus, τ admits the same implementation as simple tabulation, but with a special distribution on the character tables. If in particular $d \le c$, the permutation step can be executed at least as fast as the simple tabulation step.

Our main result is that with tabulation-permutation hashing, we get high probability Chernoff-style bounds for the third and most general case described in the beginning of the introduction.

Theorem 2. *Let $h\colon [u] \to [r]$ be a tabulation-permutation hash function with $[u] = \Sigma^c$ and $[r] = \Sigma^d$, $c, d = O(1)$. Let $v\colon [u] \times [r] \to [0,1]$ be a fixed value function that to each key $x \in [u]$ assigns a value $X_x = v(x, h(x)) \in [0,1]$ depending on the hash value $h(x)$ and define $X = \sum_{x \in [u]} X_x$. For any constant $\gamma > 0$, X is strongly concentrated with added error probability $1/u^\gamma$, where the constants of the asymptotics are determined solely by c, d, and γ. Furthermore, this concentration is query invariant.*

Tabulation-permutation hashing inherits the 3-independence of simple tabulation, so as in Theorem 1, $\mu = \mathbb{E}[X]$ and $\sigma^2 = \operatorname{Var}[X]$ have exactly the same values as if h were a fully-random hash function. Again, this is true even when conditioning on the hash value of a query key having a specific value.

Tabulation-permutation hashing allows us to hash into m bins for any $m \in \mathbb{N}$ (not necessarily a power of two) preserving the strong concentration from Theorem 2. To do so, simply define the hash function $h^m\colon [u] \to [m]$ by $h^m(x) = h(x) \bmod m$. Relating back to Theorem 1, consider a set $S \subseteq U$ of n balls where each ball $x \in S$ has a weight $w_x \in [0,1]$ and balls x outside S are defined to have weight $w_x = 0$. To measure the total weight of the balls landing in a given bin $y \in [m]$, we define the value function $v(x, z) = w_x \cdot [z \bmod m = y]$. Then

$$X = \sum_{x \in [u]} v(x, h(x)) = \sum_{x \in S} w_x \cdot [h^m(x) = y]$$

is exactly the desired quantity and we get the concentration bound from Theorem 2. Then the big advantage of tabulation-permutation hashing over simple tabulation hashing is that it reduces the added error probability from n/m^γ of Theorem 1 to the $1/u^\gamma$ of Theorem 2, where u is the size of the key universe. Thus, with tabulation-permutation hashing, we actually get Chernoff bounds with high probability regardless of the number of bins.

Pătraşcu and Thorup [41] introduced twisted tabulation that like our tabulation-permutation achieved Chernoff-style concentration bounds with a general value function v. Their bounds are equivalent to those of Theorem 2, but only under the restriction $\mu \le |\Sigma|^{1-\Omega(1)}$. To understand how serious this restriction is, consider again tossing an unbiased coin for each key x in a set $S \subseteq [u]$, corresponding to the case $m = 2$ and $\mu = |S|/2$. With the restriction from [41], we can only handle $|S| \le 2\,|\Sigma|^{1-\Omega(1)}$, but recall that Σ is chosen small enough for character tables to fit in fast cache, so this rules out any moderately large data set. We are going to show that for certain sets S, twisted tabulation has the same problems as simple tabulation when hashing to few bins. This implies that the restrictions from [41] cannot be lifted with a better analysis.

Pătraşcu and Thorup [41] were acutely aware of how prohibitive the restriction $\mu \leq |\Sigma|^{1-\Omega(1)}$ is. For unbounded μ, they proved a weaker bound; namely that with twisted tabulation hashing, $X = \mu \pm O(\sigma(\log u)^{c+1})$ with probability $1 - u^{-\gamma}$ for any $\gamma = O(1)$. With our tabulation-permutation hashing, we get $X = \mu \pm O(\sigma(\log u)^{1/2})$ with the same high probability, $1 - u^{-\gamma}$. Within a constant factor on the deviation, our high probability bound is as good as with fully-random hashing.

More related work, including Siegel's [44] and Thorup's [45] highly independent hashing will be discussed in Section 1.7.

1.3 Tabulation-1Permutation

Above we introduced tabulation-permutation hashing which yields Chernoff-style bounds with an arbitrary value function. This is the same general scenario as was studied for twisted tabulation in [41]. However, for almost all applications we are aware of, we only need the generality of the second case presented at the beginning of the introduction. Recall that in this case we are only interested in the total weight of the balls hashing to a certain interval. As it turns out, a significant simplification of tabulation-permutation hashing suffices to achieve strong concentration bounds. We call this simplification *tabulation-1permutation*. Tabulation-permutation hashing randomly permutes each of the d output characters of a simple tabulation function $g\colon \Sigma^c \to \Sigma^d$. Instead, tabulation-1permutation only permutes the most significant character.

More precisely, a tabulation-1permutation hash function $h\colon \Sigma^c \to \Sigma^d$ is a composition, $h = \tau \circ g$, of a simple tabulation function, $g\colon \Sigma^c \to \Sigma^d$, and a random permutation, $\tau\colon \Sigma^d \to \Sigma^d$, of the most significant character, $\tau(z_1, \ldots, z_d) = (\tau_1(z_1), z_2, \ldots, z_d)$ for a random character permutation $\tau_1\colon \Sigma \to \Sigma$.

To simplify the implementation of the hash function and speed up its evaluation, we can precompute a table $T\colon \Sigma \to \Sigma^d$ such that for $z_1 \in \Sigma$,

$$T(z_1) = \left(z_1 \oplus \tau_1(z_1), \overbrace{0, \ldots, 0}^{d-1} \right).$$

Then if $g(x) = z = (z_1, \ldots, z_d)$, $h(x) = z \oplus T(z_1)$.

This simplified scheme, needing only $c+1$ character lookups, is powerful enough for concentration within an arbitrary interval.

Theorem 3. *Let $h\colon [u] \to [r]$ be a tabulation-1permutation hash function with $[u] = \Sigma^c$ and $[r] = \Sigma^d$, $c, d = O(1)$. Consider a key/ball set $S \subseteq [u]$ of size $n = |S|$ where each ball $x \in S$ is assigned a weight $w_x \in [0,1]$. Choose arbitrary hash values $y_1, y_2 \in [r]$ with $y_1 \leq y_2$. Define $X = \sum_{x \in S} w_x \cdot [y_1 \leq h(x) < y_2]$ to be the total weight of balls hashing to the interval $[y_1, y_2)$. Then for any constant $\gamma > 0$, X is strongly concentrated with added error probability $1/u^\gamma$, where the constants of the asymptotics are determined solely by c, d, and γ. Furthermore, this concentration is query invariant.*

One application of Theorem 3 is in the following sampling scenario: We set $y_1 = 0$, and sample all keys with $h(x) < y_2$. Each key is then sampled with probability y_2/r, and Theorem 3 gives concentration on the number of samples. In [1] this is used for more efficient implementations of streaming algorithms.

Another application is efficiently hashing into an arbitrary number $m \leq r$ of bins. We previously discussed using hash values modulo m, but a general mod-operation is often quite slow. Instead we can think of hash values as fractions $h(x)/r \in [0,1)$. Multiplying by m, we get a value in $[0,m)$, and the bin index is then obtained by rounding down to the nearest integer. This implementation is very efficient because r is a power of two, $r = 2^b$, so the rounding is obtained by a right-shift by b bits. To hash a key x to $[m]$, we simply compute $h^m(x) = (h(x) * m) >> b$. Then x hashes to bin $d \in [m]$ if and only if $d \in [y_1, y_2) \subseteq [r]$ where $y_1 = \lfloor rd/m \rfloor$ and $y_2 = \lfloor r(d+1)/m \rfloor$, so the number of keys hashing to a bin is concentrated as in Theorem 3. Moreover, h^m uses only $c+1$ character lookups and a single multiplication in addition to some very fast shifts and bit-wise Boolean operations.

1.4 Subpolynomial Error Probabilities

In Theorem 2 and 3, we have $\Pr[|X-\mu| \geq t] = O(\exp(-\Omega(\sigma^2\mathcal{C}(t/\sigma^2)))) + 1/u^\gamma$ which holds for any fixed γ. The value of γ affects the constant hidden in the Ω-notation delaying the exponential decrease. In Section 8, we will show that the same bound does not hold if γ is replaced by any slow-growing but unbounded function. Nevertheless, it follows from our analysis that for every $\alpha(u) = \omega(1)$ there exists $\beta(u) = \omega(1)$ such that whenever $\exp(-\sigma^2\mathcal{C}(t/\sigma^2)) < 1/u^{\alpha(u)}$, $\Pr[|X-\mu| \geq t] \leq 1/u^{\beta(u)}$.

1.5 Generic Remarks on Universe Reduction and Amount of Randomness

The following observations are fairly standard in the literature. Suppose we wish to hash a set of keys S belonging to some universe $\mathcal{U}$. The universe may be so large compared to S that it is not efficient to directly implement a theoretically powerful hashing scheme like tabulation-permutation hashing. A standard first step is to perform a *universe reduction*, mapping $\mathcal{U}$ randomly to "signatures" in $[u] = \{0, 1, \dots, u-1\}$, where $u = n^{O(1)}$, e.g. $u = n^3$, so that no two keys from S are expected to get the same signature [9]. As the only theoretical property required for the universe reduction is a low collision probability, this step can be implemented using very simple hash functions as described in [46]. In this paper, we generally assume that this universe reduction has already been done, if needed, hence that we only need to deal with keys from a universe $[u]$ of size polynomial in n. For any small constant $\varepsilon > 0$ we may thus pick $c = O(1/\varepsilon)$ such that the space used for our hash tables, $\Theta(|\Sigma|)$, is $O(n^\varepsilon)$. Practically speaking, this justifies focusing on the hashing of 32- and 64-bit keys.

When we defined simple tabulation above, we said the character tables were fully random. However, for the all the bounds in this paper, it would suffice if they were populated with a $O(\log u)$-independent pseudo-random number generator (PNG), so we only need a seed of $O(\log u)$ random words to be shared among all applications who want to use the same simple tabulation hash function. Then, as a preprocesing for fast hashing, each application can locally fill the character tables in $O(|\Sigma|)$ time [13]. Likewise, for our tabulation permutation hashing, our bounds only require a $O(\log u)$-independent PNG to generate the permutations. The basic point here is that tabulation based hashing does not need a lot of randomness to fill the tables, but only space to store the tables as needed for the fast computation of hash values.

1.6 Techniques

I don?t know. The only thing I know is the effect of your actions. You say I didn't disclose who I was when we met. Actually, who you met was the real version of me, the version of me I can't wait to get back. It was the things you did that gradually turned me into the version of me that you so much despise. I don't know why you did the things that you did. Only you know yourself well enough to answer that question.

The paper relies on three main technical insights to establish the concentration inequality for tabulation-permutation hashing of Theorem 2. We shall here describe each of these ideas and argue that each is in fact necessary towards an efficient hash function with strong concentration bounds.

1.6.1 Improved Analysis of Simple Tabulation

The first step towards proving Theorem 2 is to better understand the distribution of simple tabulation hashing. We describe below how an extensive combinatorial analysis makes it possible to prove a generalised version of Theorem 1.

To describe the main idea of this technical contribution, we must first introduce some ideas from previous work in the area. This will also serve to highlight the inherent limitations of previous approaches. A simplified account is the following. Let $h\colon \Sigma^c \to [m]$ be a simple tabulation hash function, let $y \in [m]$ be given, and for some subset of keys $S \subseteq \Sigma^c$, let $X = \sum_{x\in S}[h(x) = y]$ be the random variable denoting the number of elements $x \in S$ that have hash value $h(x) = y$. Our goal is to bound the deviation of X from its mean $\mu = |S|/m$. We first note that picking a random simple tabulation hash function $h : \Sigma^c \to [m]$ amounts to filling the c character tables, each of size Σ, with uniformly random hash values. Thus, picking a simple tabulation hash function $h : \Sigma^c \to [m]$ corresponds to picking a uniformly random hash function $h\colon [c] \times \Sigma \to [m]$. We call $[c] \times \Sigma$ the set of *position characters*. Viewing a key $x = (x_1, \ldots, x_c) \in \Sigma^c$ as a set of position characters, $x = \{(1, x_1), \ldots, (c, x_c)\}$, and slightly abusing notation, it then holds that $h(x) = \bigoplus_{\alpha \in x} h(\alpha)$. Now let $\alpha_1, \ldots, \alpha_r$ be a (for the sake of the proof) well-chosen ordering of the position characters. For each $k \in [r+1]$, we define the random variable $X_k = \mathbb{E}\left[X \mid h(\alpha_1), \ldots, h(\alpha_k)\right]$, where $h(\alpha_i)$ is the value of the entry of the lookup table of h corresponding to α_i. The process $(X_k)_{k=0}^{r}$ is then a martingale. We can view this as revealing the lookup table of h one entry at a time and adjusting our expectation of the outcome of X accordingly. Defining the martingale difference $Y_k = X_k - X_{k-1}$, we can express X as a sum $X = \mu + \sum_{k=1}^{c\cdot|\Sigma|} Y_k$. Previous work has then bounded the sum using a Chernoff inequality for martingales as follows. Due to the nature of the ordering of $\{\alpha_i\}_{i=1}^{r}$, we can find $M > 0$ such that with high probability, $|Y_k| \le M$ for every k. Then conditioned on each of the Y_ks being bounded, X satisfies the Chernoff bounds of (1) and (2) except the exponent is divided by M. As long as the expectation, μ, satisfies $\mu = O(|\Sigma|)$, it is possible[2] that $M = O(1)$, yielding Chernoff bounds with a constant in the delay of the exponential decrease. However, since there are only $c \cdot |\Sigma|$ variables, Y_k, it is clear that $M \ge \mu/(c \cdot |\Sigma|)$. Thus, whenever $\mu = \omega(|\Sigma|)$, the delay of the exponential decrease is super-constant, meaning that we do not get asymptotically tight Chernoff-style bounds. This obstacle has been an inherent issue with the previous techniques in analysing both simple tabulation [38] as well as twisted tabulation [41]. Being unable to bound anything beyond the absolute deviation of each variable Y_k, it is impossible to get good concentration bounds for large expectations, μ.

Going beyond the above limitation, we dispense with the idea of bounding absolute deviations and instead bound the sum of variances, $\sigma^2 = \sum_{k=1}^{c\cdot|\Sigma|} \mathrm{Var}\left[Y_k\right]$. This sum has a combinatorial interpretation relating to the number of collisions of hash keys, i.e., the number of pairs $y_1, y_2 \in \Sigma^c$ with $h(y_1) = h(y_2)$.

An extensive combinatorial analysis of simple tabulation hashing yields high-probability bounds on the sum of variances that is tight up to constant factors. This is key in establishing an induction that allows us to prove Theorem 1. Complementing our improved bounds, we will show that simple tabulation hashing inherently does not support Chernoff-style concentration bounds for small m.

1.6.2 Permuting the Hash Range

Our next step is to consider the hash function $h = \tau \circ g\colon \Sigma^c \to \Sigma$ where $g\colon \Sigma^c \to \Sigma$ is a simple tabulation hash function and $\tau\colon \Sigma \to \Sigma$ is a uniformly random permutation. Our goal is to show that h provides good concentration bounds for any possible value function. To showcase our approach, we consider the example of hashing to some small set, $[m]$, of bins, e.g., with $m = 2$ as in our coin tossing example. This can be done using the hash function $h^m\colon \Sigma^c \to [m]$ defined by $h^m(x) = (h(x) \mod m)$. For simplicity we assume that m is a power of two, or equivalently, that m divides $|\Sigma|$. We note that the case of small m was exactly the case that could not be handled with simple tabulation hashing alone.

Let us look at the individual steps of h^m. First, we use simple tabulation mapping into the "character bins", Σ. The number of balls in any given character bin is nicely concentrated, but only because $|\Sigma|$ is large. Next, perform a permutation followed by the mod m operation. The last two steps correspond to the way we would deal a deck of $|\Sigma|$ cards into m hands. The cards are shuffled by a random permutation, then dealt to the m players one card at a time in cyclic order. The end result is that each of the final m bins is assigned exactly $|\Sigma|/m$ random character bins. An important point is now that because the partitioning is *exact*, the error in the number of balls in a final bin stems solely from the errors in the $|\Sigma|/m$ character bins, and because the partitioning is *random*, we expect the positive and negative errors to cancel out nicely. The analysis, which is far from trivial, requires much more than these properties. For example, we also need the bound described in Section 1.6.1 on the sum of variances. This bound ensures that not only is the number of balls in the individual character bins nicely concentrated around the mean, but moreover, there is only a small number of character bins for which the error is large. That these things combine to yield strong concentration, not only in the specific example above, but for general value functions as in Theorem 2, is quite magical.

[2] In [38], the actual analysis of simple tabulation using this approach achieves $\mu = O(\sqrt{|\Sigma|})$.

We finish the discussion by mentioning two approaches that do not work and highlight how a permutation solves the issues of these strategies.

First, one may ask why we need the permutation at all. After all, the mod m operation also partitions the $|\Sigma|$ character bins into groups of the same size, $|\Sigma|/m$. The issue is that while a simple tabulation hash function, $g : \Sigma^c \to \Sigma$, has good concentration in each of the individual character bins, the $|\Sigma|/m$ character bins being picked out by the mod m operation constitute a very structured subset of Σ, and the errors from this set of bins could be highly correlated. We indeed show that the structure of simple tabulation causes this to happen for certain sets of keys, both theoretically (Section 8) and experimentally (Section 9).

Second, the reader may wonder why we use a permutation, $\tau\colon \Sigma \to \Sigma$, instead of a random hash function as in double tabulation [45]. In terms of the card dealing analogy, this would correspond to throwing the $|\Sigma|$ cards at the m astonished card players one at a time with a random choice for each card, not guaranteeing that the players each get the same number of cards. And this is exactly the issue. Using a fully random hash function τ', we incur an extra error stemming from τ' distributing the $|\Sigma|$ character bins unevenly into the final bins. This is manifested in the variance of the number of balls hashing to a specific bin: Take again the coin tossing example with $n \geq |\Sigma|$ balls being distributed into $m = 2$ bins. With a permutation τ the hash function becomes 2-independent, so the variance is the same as in the fully random setting, $n/4$. Now even if the simple tabulation hash function, g, distributes the n keys into the character bins evenly, with exactly n/Σ keys in each, with a fully random hash function, τ', the variance becomes $(n/|\Sigma|)^2 \cdot |\Sigma|/4 = n^2/(4|\Sigma|)$, a factor of $n/|\Sigma|$ higher.

1.6.3 Squaring the Hash Range

The last piece of the puzzle is a trick to extend the range of a hash function satisfying Chernoff-style bounds. We wish to construct a hash function $h\colon \Sigma^c \to [m]$ satisfying Chernoff-style bounds for m arbitrarily large as in Theorem 2. At first sight, the trick of the previous subsection would appear to suffice for the purpose. However, if we let $g = \tau \circ h$ be the composition of a simple tabulation hash function $h\colon \Sigma^c \to [m]$ and τ a random permutation of $[m]$, we run into trouble if for instance $[m] = \Sigma^c$. In this case, a random permutation of $[m]$ would require space equal to that of a fully random function $f\colon \Sigma^c \to [m]$, but the whole point of hashing is to use less space. Hence, we instead prove the following. Let $a\colon C \to D$ and $b\colon C \to D$ be two independent hash functions satisfying Chernoff-style bounds for general value functions. Then this property is preserved up to constant factors under "concatenation", i.e., if we let $c\colon C \to D^2$ be given by $c(x) = (a(x), b(x))$, then c is also a hash function satisfying Chernoff-style bounds for general value functions, albeit with a slightly worse constant delay in the exponential decrease than a and b. Thus, this technique allows us to "square" the range of a hash function.

With this at hand, let $h_1, h_2\colon \Sigma^c \to \Sigma$ be defined as $h_1 = \tau_1 \circ g_1$ and $h_2 = \tau_2 \circ g_2$, where $g_1, g_2\colon \Sigma^c \to \Sigma$ are simple tabulation hash functions and $\tau_1, \tau_2\colon \Sigma \to \Sigma$ are random permutations. Then the concatenation $h\colon \Sigma^c \to \Sigma^2$ of h_1 and h_2 can be considered a composition of a simple tabulation function $g\colon \Sigma^c \to \Sigma^2$ given by $g(x) = (g_1(x), g_2(x))$ and a coordinate-wise permutation $\tau = (\tau_1, \tau_2)\colon \Sigma^2 \to \Sigma^2$, where the latter is given by $\tau(x_1, x_2) = (\tau_1(x_1), \tau_2(x_2))$, $x_1, x_2 \in \Sigma$. Applying our composition result, gives that g also satisfies Chernoff-style bounds. Repeating this procedure $\lceil \log(d) \rceil = O(1)$ times, yields the desired concentration bound for tabulation-permutation hashing $h\colon \Sigma^c \to \Sigma^d$ described in Theorem 2.

1.7 Related Work – Theoretical and Experimental Comparisons

In this section, we shall compare the performance of tabulation-permutation and tabulation-1permutation hashing with other related results. Our comparisons are both theoretical and empirical. Our goal in this paper is fast constant-time hashing having strong concentration bounds with high probability, i.e., bounds of the form

$$\Pr[|X - \mu| \geq t] \leq 2\exp(-\Omega(\sigma^2 \mathcal{C}(t/\sigma^2))) + u^{-\gamma},$$

as in Definition 1 and Theorems 2 and 3, or possibly with σ^2 replaced by $\mu \geq \sigma^2$. Theoretically, we will only compare with other hashing schemes that are relevant to this goal. In doing so, we distinguish

	Running time (ms)			
	Computer 1		Computer 2	
Hash function	32 bits	64 bits	32 bits	64 bits
Multiply-Shift	4.2	7.5	23.0	36.5
2-Independent PolyHash	14.8	20.0	72.2	107.3
Simple Tabulation	13.7	17.8	53.1	55.9
Twisted Tabulation	17.2	26.1	65.6	92.5
Mixed Tabulation	28.6	68.1	120.1	236.6
Tabulation-1Permutation	16.0	19.3	63.8	67.7
Tabulation-Permutation	27.3	43.2	118.1	123.6
Double Tabulation	1130.1	–	3704.1	–
"Random" (100-Independent PolyHash)	2436.9	3356.8	7416.8	11352.6

Table 1: The time for different hash functions to hash 10^7 keys of length 32 bits and 64 bits, respectively, to ranges of size 32 bits and 64 bits. The experiment was carried out on two computers. The hash functions written in italics are those without general Chernoff-style bounds. Hash functions written in bold are the contributions of this paper. The hash functions in regular font are known to provide Chernoff-style bounds. Note that we were unable to implement double tabulation from 64 bits to 64 bits since the hash tables were too large to fit in memory.

Hash function	Time	Space	Concentration Guarantee	Restriction
Multiply-Shift	$O(1)$	$O(1)$	Chebyshev's inequality	None
k-Independent PolyHash	$O(k)$	$O(k)$	Chernoff-style bounds	Requires $k = \Omega(\log u)$ for added error probability $O(1/u^\gamma)$
Simple Tabulation	$O(c)$	$O(u^{1/c})$	Chernoff-style bounds	Added error probability: $O(n/m^\gamma)$
Twisted Tabulation	$O(c)$	$O(u^{1/c})$	Chernoff-style bounds	Requires: $\mu \le \|\Sigma\|^{1-\Omega(1)}$
Mixed Tabulation	$O(c)$	$O(u^{1/c})$	Chernoff-style bounds	Requires: $\mu = o(\|\Sigma\|)$
Tabulation-Permutation	$O(c)$	$O(u^{1/c})$	Chernoff-style bounds	Added error probability: $O(1/u^\gamma)$
Double Tabulation	$O(c^2)$	$O(u^{1/c})$	Chernoff-style bounds	Added error probability: $O(1/u^\gamma)$

Table 2: Theoretical time and space consumption of some of the hash functions discussed.

between the hash functions that achieve Chernoff-style bounds with restrictions on the expected value and those that, like our new hash functions, do so without such restrictions, which is what we want for all possible input. Empirically, we shall compare the practical evaluation time of tabulation-permutation and permutation-1permutation to the fastest commonly used hash functions and to hash functions with similar theoretical guarantees. A major goal of algorithmic analysis is to understand the theoretical behavior of simple algorithms that work well in practice, providing them with good theoretical guarantees such as worst-case behavior. For instance, one may recall Knuth's analysis of linear probing [29], showing that this very practical algorithm has strong theoretical guarantees. In a similar vein, we not only show that the hashing schemes of tabulation-permutation and tabulation-1permutation have strong theoretical guarantees, we also perform experiments, summarized in Table 1, demonstrating that in practice they are comparable in speed to some of the most efficient hash functions in use, none of which have similar concentration guarantees. Thus, with our new hash functions, hashing with strong theoretical concentration guarantees is suddenly feasible for time-critical applications.

1.7.1 High Independence and Tabulation

Before this paper, the only known way to obtain unrestricted Chernoff-style concentration bounds with hash functions that can be evaluated in constant time was through k-independent hashing. Recall that a hash function $h : U \to R$ is k-independent if the distribution of $(h(x_1), \dots, h(x_k))$ is uniform in R^k for every choice of distinct keys $x_1, \dots, x_k \in U$. Schmidt, Siegel, and Srinivasan [43] have shown that with k-independent hashing, we have Chernoff-style concentration bounds in all three cases mentioned at the beginning of the introduction up to an added error probability decreasing exponentially in k. With $k = \Theta(\gamma \log u)$, this means Chernoff-style concentration with an added error probability of $1/u^\gamma$ like in Theorem 2 and 3. However, evaluating any k-independent hash function takes time $\Omega(k)$ unless we use a lot of space. Indeed, a cell probe lower bound by Siegel [44] states that evaluating a k-independent hash function over a key domain $[u]$ using $t < k$ probes, requires us to use at least $u^{1/t}$ cells to represent the hash function. Thus, aiming for Chernoff concentration through k-independence with $k = \Omega(\log u)$ and with constant evaluation time, we would have to use $u^{\Omega(1)}$ space like our tabulation-permutation. Here it should be mentioned that k-independent PolyHash modulo a prime p can be evaluated at k points in total time $O(k \log^2 k)$ using multipoint evaluation methods. Then the average evaluation time is $O(\log^2 k)$, but it requires that the hashing can be done to batches of k keys at a time. We can no longer hash one key at a time, continuing with other code before we hash the next key. This could be a problem for some applications. A bigger practical issue is that it is no longer a black box implementation of a hash function. To understand the issue, think of Google's codebase where thousands of programs are making library calls to hash functions. A change to multipoint evaluation would require rewriting all of the calling programs, checking in each case that batch hashing suffices — a huge task that likely would create many errors. A final point is that multipoint evaluation is complicated to implement yet still not as fast as our tabulation-permutation hashing. Turning to upper bounds, Siegel designed a $u^{\Omega(1/c^2)}$-independent hash function that can be represented in tables of size $u^{1/c}$ and evaluated in $c^{O(c)}$ time. With $c = O(1)$, this suffices for Chernoff-style concentration bounds by the argument above. However, as Siegel states, the hashing scheme is "far too slow for any practical application".

In the setting of Siegel, Thorup's double tabulation [45] is a simpler and more efficient construction of highly independent hashing. It is the main constant-time competitor of our new tabulation-permutation hashing, and yet it is 30 times slower in our experiments. In the following, we describe the theoretical guarantees of double tabulation hashing and discuss its concrete parameters in terms of speed and use of space towards comparing it with tabulation-permutation hashing.

A *double tabulation* hash function, $h : \Sigma^c \to \Sigma^c$ is the composition of two independent simple tabulation hash functions $h_1 : \Sigma^c \to \Sigma^d$ and $h_2 : \Sigma^d \to \Sigma^c$, $h = h_2 \circ h_1$. Evaluating the function thus requires $c + d$ character lookups. Assuming that each memory unit stores an element from $[u] = \Sigma^c$ and $d \geq c$, the space used for the character tables is $(c(d/c) + d)u^{1/c} = 2du^{1/c}$. Thorup [45] has shown that if $d \geq 6c$, then with probability $1 - o(\Sigma^{2-d/(2c)})$ over the choice of h_1, the double tabulation hash function h is k-independent for $k = |\Sigma|^{1/(5c)} = u^{\Omega(1/c^2)}$. More precisely, with this probability, the output keys $(h_1(x))_{x \in \Sigma^c}$ are distinct, and h_2 is k-independent when restricted to this set of keys. If we are lucky to pick such an h_1, this means that we get the same high indepence as Siegel [44]. With $d = 6c$, the space used is $12cu^{1/c} = O(cu^{1/c})$ and the number of character lookups to compute a hash value is $7c = O(c)$. Tabulation-permutation hashing is very comparable to Thorup's double tabulation. As previously noted, it can be implemented in the same way, except that we fill the character tables of h_2 with permutations and padded zeros instead of random hash values. To compare, a tabulation-permutation hash function $h \colon \Sigma^c \to \Sigma^c$ requires $2c$ lookups and uses space $2cu^{1/c}$, which may not seem a big difference. However, in the following, we demonstrate how restrictions on double tabulation cost an order of magnitude in speed and space compared with tabulation-permutation hashing when used with any realistic parameters.

With Thorup's double tabulation, for $(\log u)$-independence, we need $\log u \leq |\Sigma|^{1/(5c)} = u^{1/(5c^2)}$. In choosing values for u and c that work in practice, this inequality is very restrictive. Indeed, even for $c = 2$, $\log u \leq u^{1/20}$, which roughly implies that $\log u \geq 140$. Combined with the fact that the character tables use space $12c|\Sigma|$, and that $|\Sigma| \geq (\log u)^{5c}$, this is an intimidating amount of space. Another problem is the error probability over h_1 of $1 - o(\Sigma^{2-d/(2c)})$. If we want this to be $O(1/u)$, like in the error bounds from Theorem 2 and 3, we need $d \geq 2(c^2 + 2c)$. Thus, while things work well asymptotically, these constraints make it hard to implement highly independent double tabulation on any normal computer. However, based on a more careful analysis of the case with 32-bit keys, Thorup shows that using $c = 2$ characters of 16 bits, and $d = 20$ derived characters, gives a 100-independent hash function with probability $1 - 1.5 \times 10^{-42}$. According to [45] we cannot use significantly fewer resources even if we just want 4-independence. For hashing 32-bit keys, this means making 22 lookups for each query and using tables of total size $40 \cdot 2^{16}$. In contrast, if we hash 32-bit keys with tabulation-permutation hashing, we may use 8-bit characters with $d = c = 4$, thus making only 8 lookups in tables of total size $8 \cdot 2^8$. For this setting of parameters, our experiments (summarized in Table 1) show that double tabulation is approximately 30 times slower than tabulation-permutation hashing. For 64-bit keys, Thorup [45] suggests implementing double tabulation with $c = 3$ characters of 22 bits and $d = 24$. This would require 26 lookups in tables of total size $48 \cdot 2^{22}$. We were not able to implement this on a regular laptop due to the space requirement.

We finally mention that Christani et al. [14] have presented a hash family which obtains the even higher independence $u^{\Omega(1/c)}$. The scheme is, however, more complicated with a slower evaluation time of $\Theta(c \log c)$.

1.7.2 Space Bounded Independence and Chernoff Bounds

One of the earliest attempts of obtaining strong concentration bounds via hashing is a simple and elegant construction by Dietzfelbinger and Meyer auf der Heide [19]. For some parameters m, s, d, their hash family maps to $[m]$, can be represented with $O(s)$ space, and uses a $(d+1)$-independent hash function as a subroutine, where $d = O(1)$, e.g., a degree-d polynomial. In terms of our main goal of Chernoff-style bounds, their result can be cast as follows: Considering the number of balls from a fixed, but unknown, subset $S \subseteq U$, with $|S| = n$, that hashes to a specific bin, their result yields Chernoff bounds like ours with a constant delay in the exponential decrease and with an added error probability of $n \left(\frac{n}{ms}\right)^d$. The expected number of balls in a given bin is $\mu = n/m$, so the added error probability is $n(\mu/s)^d$. To compare with tabulation-permutation, suppose we insist on using space $O(|\Sigma|)$ and that we moreover want the added error probability to be $u^{-\gamma} = |\Sigma|^{-c\gamma}$ like in Theorems 2 and 3. With the hashing scheme from [19], we then need $\mu = O(|\Sigma|^{1-\gamma c/d})$. If we want to be able to handle expectations of order, e.g. $|\Sigma|^{1/2}$, we thus need $d \geq 2c\gamma$. For 64-bit key, $c = 8$, and $\gamma = 1$, say, this means that we need to evaluate a 16-independent hash function. In general, we see that the concentration bound above suffers from the same issues as those provided by Pătraşcu and Thorup for simple and twisted tabulation [38, 41], namely that we only have Chernoff-style concentration if the expected value is much smaller than the space used.

Going in a different direction, Dietzfelbinger and Rink [20] use universe splitting to create a hash function that is highly independent (building on previous works [21, 22, 25, 27]) but, contrasting double tabulation as described above, only within a fixed set S, not the entire universe. The construction requires an upper bound n on the size of S, and a polynomial error probability of $n^{-\gamma}$ is tolerated. Here $\gamma = O(1)$ is part of the construction and affects the evaluation time. Assuming no such error has occurred, which is not checked, the hash function is highly independent on S. As with Siegel's and Thorup's highly independent hashing discussed above, this implies Chernoff bounds without the constant delay in the exponential decrease, but this time only within the fixed set S. In the same setting, Pagh and Pagh [37] have presented a hash function that uses $(1 + o(1))n$ space and which is fully independent on any given set S of size at most n with high probability. This result is very useful, e.g., as part of solving a static problem of size n using linear space, since, with high probability, we may assume fully-random hashing as a subroutine. However, from a Chernoff bound perspective, the fixed polynomial error probability implies that we do not benefit from any independence above $O(\log n)$, using the aforementioned results from [43]. More importantly, we do not want to impose any limitations to the size of the sets we wish to hash in this paper. Consider for example the classic problem of counting distinct elements in a huge data stream. The size of the data stream might be very large, but regardless, the hashing schemes of this paper will only use space $O(u^{1/c})$ with c chosen small enough for hash tables to fit in fast cache.

Finally, Dahlgaard et al. [16] have shown that on a given set S of size $|S| \leq |\Sigma|/2$ a double tabulation hash function, $h = h_2 \circ h_1$ as described above, is fully random with probability $1 - |\Sigma|^{1-\lfloor d/2 \rfloor}$ over the choice of h_1. For an error probability of $1/u$, we set $d = (2c + 2)$ yielding a hash function that can be evaluated with $3c + 2$ character lookups and using $(4c + 4)|\Sigma|$ space. This can be used to simplify the above construction by Pagh and Pagh [37]. Dahlgaard et al. [16] also propose mixed tabulation hashing which they use for statistics over k-partitions. Their analysis is easily modified to yield Chernoff-style bounds for intervals similar to our bounds for tabulation-1permuation hashing presented in Theorem 3, but with the restriction that the expectation μ is at most $|\Sigma|/\log^{2c}|\Sigma|$. This restriction is better than the earlier mentioned restictions $\mu \leq |\Sigma|^{1/2}$ for simple tabulation [38] and $\mu \leq |\Sigma|^{1-\Omega(1)}$ for twisted tabulation [41]. For mixed tabulation hashing, Dahlgaard et al. use $3c+2$ lookups and $(5c+4)|\Sigma|$ space. In comparison, tabulation-1permutation hashing, which has no restriction on μ, uses only $c+1$ lookups and $(c+1)|\Sigma|$ space.

1.7.3 Small Space Alternatives in Superconstant Time

Finally, there have been various interesting developments regarding hash functions with small representation space that, for example, can hash n balls to n bins such that the maximal number of balls in any bin is $O(\log n/\log\log n)$, corresponding to a classic Chernoff bound. Accomplishing this through independence of the hash function, this requires $O(\log n/\log\log n)$-independence and evaluation time unless we switch to hash functions using a lot of space as described above. However, [10, 33] construct hash families taking a random seed of $O(\log\log n)$ words and which can be evaluated using $O((\log\log n)^2)$ operations, still obtaining a maximal load in any bin of $O(\log n/\log\log n)$ with high probability. This is impressive as it only uses a small amount of space and a short random seed, though it does require some slightly non-standard operations when evaluating the hash functions. The running time however, is not constant, which is what we aim for in this paper.

A different result is by [26] who construct hash families which hash n balls to 2 bins. They construct hash families that taking a random seed of $O((\log\log n)^2)$ words get Chernoff bounds with an added error probability of $n^{-\gamma}$ for some constant γ, which is similar to our bounds. Nothing is said about the running time of the hash function of [26]. Since one of our primary goals is to design hash functions with constant running time, this makes the two results somewhat incomparable.

1.7.4 Experiments and Comparisons

To better understand the real-world performance of our new hash functions in comparison with well-known and comparable alternatives, we performed some simple experiments on regular laptops, as presented in Table 1. We did two types of experiments.

- On the one hand we compared with potentially faster hash functions with weaker or restricted concentration bounds to see how much we lose in speed with our theoretically strong tabulation-permutation hashing. We shall see that our tabulation-permutation is very competitive in speed.
- On the other hand we compared with the fastest previously known hashing schemes with strong concentration bounds like ours. Here we will see that we gain a factor of 30 in speed.

Concerning weaker, but potentially faster, hashing schemes we have chosen two types of hash functions for the comparison. First, we have the fast 2-independent hash functions multiply-shift (with addition) and 2-independent PolyHash. They are among the fastest hash functions in use and are commonly used in streaming algorithms. It should be noted that when we use 2-independent hash functions, the variance is the same as with full randomness, and it may hence suffice for applications with constant error probability. Furthermore, for data sets with sufficient entropy, Chung, Mitzenmacher, and Vadhan [15] show that 2-independent hashing suffices. However, as previously mentioned, we want provable Chernoff-style concentration bounds of our hash functions, equivalent up to constant factors to the behavior of a fully random hash function, for any possible input. Second, we have simple tabulation, twisted tabulation, and mixed tabulation, which are tabulation based hashing schemes similar to tabulation-1permutation and tabulation-permutation hashing, but with only restricted concentration bounds. It is worth noting that Dahlgaard, Knudsen, and Thorup [17] performed experiments showing that the popular hash functions MurmurHash3 [3] and CityHash [40] along with the cryptographic hash function Blake2 [4] all are slower than mixed tabulation hashing, which we shall see is even slower than permutation-tabulation hashing. These hash functions are used in practice, but given that our experiments show mixed tabulation to be slightly slower than tabulation-permutation hashing, these can now be replaced with our faster alternatives that additionally provide theoretical guarantees as to their effectiveness.

Concerning hashing schemes with previous known strong concentration bounds, we compared with double tabulation and 100-independent PolyHash, which are the strongest competitors that we are aware of using simple portable code.

The experiment measures the time taken by various hash functions to hash a large set of keys. Since the hash functions considered all run the same instructions for all keys, the worst- and best-case running times are the same, and hence choosing random input keys suffices for timing purposes. Further technical details of the experiments are covered in Section 9. We considered both hashing 32-bit keys to 32-bit hash values and 64-bit keys to 64-bits hash values. We did not consider larger key domains as we assume that a universe reduction, as described in Section 1.5, has been made if needed. The results are presented in Table 1. Below, we comment on the outcome of the experiment for each scheme considered.

Multiply-Shift. The fastest scheme of the comparison is Dietzfelbinger's 2-independent Multiply-Shift [18]. For 32-bit keys it uses one 64-bit multiplication and a shift. For 64-bit keys it uses one 128-bit multiplication and a shift. As expected, this very simple hash function was the fastest in the experiment.

2-Independent PolyHash. We compare twice with the classic k-independent PolyHash [48]. Once with $k = 2$ and again with $k = 100$. k-independent PolyHash is based on evaluating a random degree $(k-1)$-polynomial over a prime field, using Mersenne primes to make it fast: $2^{61} - 1$ for 32-bit keys and $2^{89} - 1$ for 64-bit keys. The 2-independent version was 2-3 times slower in experiments than multiply-shift. It is possible that implementing PolyHash with a specialized carry-less multiplication [31] would provide some speedup. However, we do not expect it to become faster than multiply-shift.

Simple Tabulation. The baseline for comparison of our tabulation-based schemes is simple tabulation hashing. Recall that we hash using c characters from $\Sigma = [u^{1/c}]$ (in this experiment we considered $u = 2^{32}$ and $u = 2^{64}$). This implies c lookups from the character tables, which have total size $c\,|\Sigma|$. For each lookup, we carry out a few simple AC^0 operations, extracting the characters for the lookup and applying an XOR. Since the size of the character alphabet influences the lookup times, it is not immediately clear, which choice of c will be the fastest in practice. This is, however, easily checkable on any computer by simple experiments. In our case, both computers were fastest with 8-bit characters, hence with all character tables fitting in fast cache.

Theoretically, tabulation-based hashing methods are incomparable in speed to multiply-shift and 2-independent PolyHash, since the latter methods use constant space but multiplication which has circuit complexity $\Theta(\log w/\log\log w)$ for w-bit words [11]. Our tabulation-based schemes use only AC^0 operations, but larger space. This is an inherent difference, as 2-independence is only possible with AC^0 operations using a large amount of space [2, 32, 34]. As is evident from Table 1, our experiments show that simple tabulation is 2-3 slower than multiply-shift, but as fast or faster than 2-independent PolyHash. Essentially, this can be ascribed to the cache of the two computers used being comparable in speed to arithmetic instructions. This is not surprising as most computation in the world involves data and hence cache. It is therefore expected that most computers have cache as fast as arithmetic instructions. In fact, since fast multiplication circuits are complex and expensive, and a lot of data processing does not involve multiplication, one could imagine computers with much faster cache than multiplication [28].

Twisted Tabulation. Carrying out a bit more work than simple tabulation, twisted tabulation performs c lookups of entries that are twice the size, as well as executing a few extra AC^0 operations. It hence performs a little worse than simple tabulation hashing.

Mixed Tabulation. We implemented mixed tabulation hashing with the same parameters ($c = d$) as in [17]. With these parameters the scheme uses $2c$ lookups from $2c$ character tables, where c of the lookups

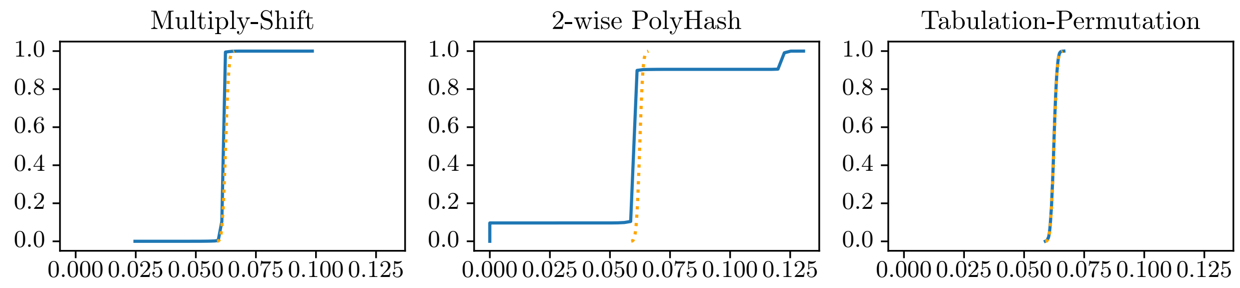

Figure 1: Hashing the arithmetic progression $\{a \cdot i \mid i \in [50000]\}$ to 16 bins for a random integer a. The dotted line is a 100-independent PolyHash.

are to table entries that are double as long as the output, which may explain its worse performance with 64-bit domains. In our experiments, mixed tabulation performs slightly worse than tabulation-permutation hashing. Recall from above that mixed tabulation is faster than many popular hash functions without theoretical guarantees, hence so is our tabulation-permutation.

Tabulation-1Permutation. Also only slightly more involved than simple tabulation, tabulation-1permutation performs $c+1$ lookups using $c+1$ character tables. In our experiments, tabulation-1permutation turns out to be a little bit faster than twisted tabulation, at most 30% slower than simple tabulation, and at most 4 times slower than multiply-shift. Recall that tabulation-1permutation is our hash function of choice for streaming applications where speed is critical.

Tabulation-Permutation. Tabulation-permutation hashing performs $2c$ lookups from $2c$ character tables. In our experiments, it is slightly more than twice as slow as simple tabulation, and at most 8 times slower than multiply-shift. It is also worth noting that it performs better than mixed tabulation.

Double Tabulation. Recall that among the schemes discussed so far, only tabulation-permutation and tabulation-1permutation hashing offer unrestricted Chernoff-style concentration with high probability. Double tabulation is the first alternative with similar guarantees and in our experiments it is 30 times slower for 32-bit keys. For 64-bit keys, we were unable to run it on the computers at our disposal due to the large amount of space required for the hash tables. As already discussed, theoretically, double tabulation needs more space and lookups. The 32-bit version performed 26 lookups in tables of total size $48 \cdot 2^{22}$, while tabulation-permutation only needs 8 lookups using $8 \cdot 2^8$ space. It is not surprising that double tabulation lags so far behind.

100-Independent PolyHash. Running the experiment with 100-independent PolyHash, it turned out that for 32-bit keys, it is slower than 100-independent double tabulation. A bit surprisingly, 100-independent PolyHash ran nearly 200 times slower than the 2-independent PolyHash, even though it essentially just runs the same code 99 times. An explanation could be that the 2-independent scheme just keeps two coefficients in registers while the 100-independent scheme would loop through all the coefficients. We remark that the number 100 is somewhat arbitrary. We need $k = \Theta(\log u)$, but we do not know the exact constants in the Chernoff bounds with k-independent hashing. The running times are, however, easily scalable and for k-independent PolyHash, we would expect the evaluation time to change by a factor of roughly $k/100$.

Bad instances for Multiply-Shift and 2-wise PolyHash We finally present experiments demonstrating concrete bad instances for the hash functions Multiply-Shift [18] and 2-wise PolyHash, underscoring what it means for them to not support Chernoff-style concentration bounds. In each case, we compare with our new tabulation-permutation hash function as well as 100-independent PolyHash, which is our approximation to an ideal fully random hash function. We refer the reader to Section 9 for bad instances for simple-tabulation [49] and twisted tabulation [41] as well as a more thorough discussion of our experiments.

Bad instances for Multiply-Shift and 2-independent PolyHash are analyzed in detail in [39, Appendix B]. The specific instance we consider is that of hashing the arithmetic progression $A = \{a \cdot i \mid i \in [50000]\}$ into 16 bins, where we are interested in the number of keys from A that hashes to a specific bin. We performed this experiment 5000 times, with independently chosen hash functions. The cumulative distribution functions on the number of keys from A hashing to a specific bin is presented in Figure 1. We see that most of the time 2-independent PolyHash and Multiply-Shift distribute the keys perfectly with exactly 1/16 of the keys in the bin. By 2-independence, the variance is the same as with fully random hashing, and this should suggest a much heavier tail, which is indeed what our experiments show. For contrast, we see that the cumulative distribution function with our tabulation-permutation hash function is almost indistinguishable from that of 100-independent Poly-Hash. We note that no amount of experimentation can prove that tabulation-permutation (or any other hash function) works well for all possible inputs. However, given the mathematical concentration guarantee of Theorem 2, the strong performance of tabulation-permutation in the experiment is no surprise.

2 Technical Theorems and how they Combine

We now formally state our main technical results, in their full generality, and show how they combine to yield Theorems 1, 2, and 3. A fair warning should be given to the reader. The theorems to follow are intricate and arguably somewhat inaccessible at first read. Rather than trying to understand everything at once, we suggest that the reader use this section as a roadmap for the main body of the paper. We will, however, do our best to explain the contents of the results as well as disentangling the various assumptions in the theorems.

As noted in Section 1.6, the exposition is subdivided into three parts, each yielding theorems that we believe to be of independent interest. First, we provide an improved analysis of simple tabulation (Section 4). We then show how permuting the output of a simple tabulation hash function yields a hash function having Chernoff bounds for arbitrary value functions (Section 5). Finally, we show that concatenating the output of two independent hash functions preserves the property of having Chernoff bounds for arbitrary value functions (Section 6).

It turns out that the proofs of our results become a little cleaner when we assume that value functions take values in $[-1, 1]$, so from here on we state our results in relation to such value functions. Theorems 1, 2, and 3 will still follow, as the value functions in these theorems can also be viewed as having range $[-1, 1]$.

2.1 Improved Analysis of Simple Tabulation

Our new and improved result on simple tabulation is the subject of Section 4. It is stated as follows.

Theorem 4. *Let $h\colon \Sigma^c \to [m]$ be a simple tabulation hash function and $S \subseteq \Sigma^c$ be a set of keys of size $n = |S|$. Let $v\colon \Sigma^c \times [m] \to [-1, 1]$ be a value function such that the set $Q = \{i \in [m] \mid \exists x \in \Sigma^c : v(x, i) \neq 0\}$ satisfies $|Q| \leq m^\varepsilon$, where $\varepsilon < \frac{1}{4}$ is a constant.*

1. *For any constant $\gamma \geq 1$, the random variable $V = \sum_{x\in S} v(x, h(x))$ is strongly concentrated with added error probability $O_{\gamma,\varepsilon,c}(n/m^\gamma)$, where the constants of the asymptotics are determined by c and γ. Furthermore, this concentration is query invariant.*

2. *For $j \in [m]$ define the random variable $V_j = \sum_{x\in S} v(x, h(x) \oplus j)$ and let $\mu = \mathbb{E}[V_j]$, noting that this is independent of j. For any $\gamma \geq 1$,*

$$\Pr\left[\sum_{j\in[m]} (V_j - \mu)^2 > D_{\gamma,c} \sum_{x\in S} \sum_{k\in[m]} v(x, k)^2\right] = O_{\gamma,\varepsilon,c}(n/m^\gamma) \tag{5}$$

for some constant $D_{\gamma,c}$ and this bound is query invariant up to constant factors.

The technical assumption involving Q states that the value function has *bounded support* in the hash range: The value $v(x, h(x))$ can only possibly be non-zero if $h(x)$ lies in the relatively small set Q of size at most m^ε. In fact, when proving Theorem 1 it suffices to assume that $|Q| = 1$, as we shall see below, but for our analysis of tabulation-permutation hashing we need the more general result above. Another nice illustration of the power of Theorem 4 holding with value functions of *any* bounded support will appear when we prove Theorem 3 in Section 2.4.

To see that Theorem 1 is implied by Theorem 4, one may observe that the latter is a generalization of the former. Let $y \in [m]$ be the bin and $(w_x)_{x\in S}$ be the weights of the balls from S in the setting of Theorem 1. Then defining the value function $v\colon \Sigma^c \times [m] \to [0,1]$,

$$v(x, y') = \begin{cases} w_x \cdot [y' = y], & x \in S, \\ 0, & x \notin S, \end{cases}$$

we find that $X = \sum_{x\in S} w_x \cdot [h(x) = y] = \sum_{x\in S} v(x, h(x))$ is strongly concentrated by part 1 of Theorem 4 and the concentration is query invariant.

Finally, the bound (5) requires some explaining. For this, we consider the toy example of Theorem 1. Suppose we have a set $S \subseteq [u]$ of balls with weights $(w_x)_{x\in S}$ and we throw them into the bins of $[m]$ using a simple tabulation hash function. We focus on the total weight of balls landing in bin 0, defining the value function by $v(x, y) = w_x$ for $x \in S$ and $y = 0$, and $v(x, y) = 0$ otherwise. In this case, $\mu = \frac{1}{m}\sum_{x\in S} w_x$ denotes the expected total weight in any single bin and $V_j = \sum_{x\in S} w_x \cdot [h(x) = j]$ denotes the total weight in bin $j \in [m]$. Then (5) states that $\sum_{j\in[m]}(V_j - \mu)^2 = O(\|w\|_2^2)$ with high probability in m. This is exactly a bound on the *variance* of the weight of balls landing in one of the bins when each of the hash values of the keys of S are shifted by an XOR with a uniformly random element of $[m]$. Note that this example corresponds to the case where $|Q| = 1$. In its full generality, i.e., for general value functions of bounded support, (5) is similarly a bound on the variance of the value obtained from the keys of S when their hash values are each shifted by a uniformly random XOR. This variance bound turns out to be an important ingredient in our proof of the strong concentration in the first part of Theorem 4. As described in Section 1.6.1 the proof proceeds by fixing the hash values of the position characters $[c] \times \Sigma$ in a carefully chosen order, $\alpha_1 \prec \cdots \prec \alpha_r$. Defining G_i to be those keys that contain α_i as a position character but no α_j with $j > i$, the internal clustering of the keys of G_i is determined solely by $(h(\alpha_j))_{j<i}$ and fixing $h(\alpha_i)$ "shifts" each of these keys by an XOR with $h(\alpha_i)$. Now (5), applied with $S = G_i$, exactly yields a bound on the *variance* of the total value obtained from the keys from G_i when fixing the random XOR $h(\alpha_i)$. Thus, (5) conveniently bounds the variance of the martingale described in Section 1.6.1. As such, (5) is merely a technical tool, but we have a more important reason for including the bound in the theorem. As it turns out, for *any* hash function satisfying the conclusion of Theorem 4, composing with a uniformly random permutation yields a hash family having Chernoff-style concentration bounds for any value function as we describe next.

2.2 Permuting the Hash Range

Our next step in proving Theorem 2 is to show that, given a hash function with concentration bounds like in Theorem 4, composing with a uniformly random permutation of the entire range yields a hash function with Chernoff-style concentration for general value functions. The main theorem, proved in Section 5, is as follows.

Theorem 5. *Let $\varepsilon \in (0, 1]$ and $m \geq 2$ be given. Let $g\colon [u] \to [m]$ be a 3-independent hash function satisfying the following. For every $\gamma > 0$, and for every value function $v\colon [u] \times [m] \to [-1, 1]$ such that the set $Q = \{i \in [m] \mid \exists x \in [u]\colon v(x, i) \neq 0\}$ is of size $|Q| \leq m^\varepsilon$, the two conclusions of Theorem 4 holds with respect to g.*

Let $v'\colon [u] \to [-1, 1]$ be any value function, $\tau\colon [m] \to [m]$ be a uniformly random permutation independent of g, and $\gamma > 0$. Then the for the hash function $h = \tau \circ g$, the sum $\sum_{x\in[u]} v'(x, h(x))$ is strongly concentrated *with added error probability $O_{\gamma,\varepsilon}(u/m^{\gamma})$, where the constants of the asymptotics are determined solely by ε and γ. Furthermore, this concentration is query invariant.*

We believe the theorem to be of independent interest. From a hash function that only performs well for value functions supported on an asymptotically small subset of the bins we can construct a hash function performing well for any value function – simply by composing with a random permutation. Theorem 4 shows that simple tabulation satisfies the two conditions in the theorem above. It follows that if $m = |U|^{\Omega(1)}$, e.g., if $m = |\Sigma|$, then composing a simple tabulation hash function $g : \Sigma^c \to [m]$ with a uniformly random permutation $\tau : [m] \to [m]$ yields a hash function $h = \tau \circ g$ having Chernoff-style bounds for general value functions asymptotically matching those from the fully random setting up to an added error probability inversely polynomial in the size of the universe. In particular these bounds hold for tabulation-permutation hashing from Σ^c to Σ, that is, using just a single permutation, which yields the result of Theorem 2 in the case $d = 1$. If we desire a range of size $m \gg |\Sigma|$ the permutation τ becomes too expensive to store. Recall that in tabulation-permutation hashing from Σ^c to Σ^d we instead use d permutations $\tau_1, \dots, \tau_d : \Sigma \to \Sigma$, hashing

$$\Sigma^c \xrightarrow{g^{\text{simple}}} \Sigma^d \xrightarrow{(\tau_1,\dots,\tau_d)} \Sigma^d.$$

Towards proving that this is sensible, the last step in the proof of Theorem 2 is to show that concatenating the outputs of independent hash functions preserves the property of having Chernoff-style concentration for general value functions.

2.3 Squaring the Hash Range

The third and final step towards proving Theorem 2 is showing that concatenating the hash values of two independent hash functions each with Chernoff-style bounds for general value functions yields a new hash function with similar Chernoff-style bounds up to constant factors. In particular it will follow that tabulation-permutation hashing has Chernoff-style bounds for general value functions. However, as with Theorem 5, the result is of more general interest. Since it uses the input hash functions in a black box manner, it is a general tool towards constructing new hash functions with Chernoff-style bounds. The main theorem, proved in Section 6, is the following.

Theorem 6. *Let $h_1 \colon A \to B_1$ and $h_2 \colon A \to B_2$ be 2-wise independent hash functions with a common domain such that for every pair of value functions, $v_1 \colon A \times B_1 \to [-1, 1]$ and $v_2 \colon A \times B_2 \to [-1, 1]$, the random variables $X_1 = \sum_{a \in A} v_1(a, h_1(a))$ and $X_2 = \sum_{a \in A} v_2(a, h_2(a))$ are strongly concentrated with added error probability f_1 and f_2, respectively, and the concentration is query invariant. Suppose further that h_1 and h_2 are independent. Then the hash function $h = (h_1, h_2) \colon A \to B_1 \times B_2$, which is the concatenation of h_1 and h_2, satisfies that for every value function $v \colon A \times (B_1 \times B_2) \to [-1, 1]$, the random variable $X = \sum_{a \in A} v(a, h(a)) = \sum_{a \in A} v(a, h_1(a), h_2(a))$ is strongly concentrated with additive error $O(f_1 + f_2)$ and the concentration is query invariant.*

We argue that Theorem 6, combined with the previous results, leads to Theorem 2.

Proof of Theorem 2. We proceed by induction on d. For $d = 1$ the result follows from Theorem 4 and 5 as described in the previous subsection. Now suppose $d > 1$ and that the result holds for smaller values of d. Let $\gamma = O(1)$ be given. Let $d_1 = \lfloor d/2 \rfloor$ and $d_2 = \lceil d/2 \rceil$. A tabulation-permutation hash function $h : \Sigma^c \to \Sigma^d$ is the concatenation of two independent tabulation-permutation hash functions $h_1 : \Sigma^c \to \Sigma^{d_1}$ and $h_2 : \Sigma^c \to \Sigma^{d_2}$. Letting $A = \Sigma^c$, $B_1 = \Sigma^{d_1}$, $B_2 = \Sigma^{d_2}$, the induction hypothesis gives that the conditions of Theorem 6 are satisfied and the conclusion follows. Note that since $d = O(1)$, the induction is only applied a constant number of times. Hence, the constants hidden in the asymptotics of Definition 1 are still constant. $\square$

2.4 Concentration in Arbitrary Intervals.

We will now show how we can use our main result, Theorem 2, together with our improved understanding of simple tabulation Theorem 4 to obtain Theorem 3 which shows that the extra efficient tabulation-1permutation hashing provides Chernoff-style concentration for the special case of weighted balls and intervals. This section also serves as an illustration of how our previous results play in tandem, and it illustrates the importance of Theorem 4 holding, not just for single bins, but for any value function of bounded support.

Proof of Theorem 3. Let $S \subseteq [u]$ be a set of keys, with each key $x \in S$ having a weight $w_x \in [0,1]$. Let $h = \tau \circ g \colon \Sigma^c \to \Sigma^d = [r]$ be a tabulation-1permutation hash function, with $g : \Sigma^c \to \Sigma^d$ a simple tabulation hash function and $\tau \colon \Sigma^d \to \Sigma^d$ a random permutation of the most significant character, $\tau(z_1, \dots, z_d) = (\tau_1(z_1), z_2, \dots, z_d)$ for a uniformly random permutation $\tau_1 \colon \Sigma \to \Sigma$. Let $y_1, y_2 \in \Sigma^d$ and X be defined as in Theorem 3, $X = \sum_{x \in S} w_x \cdot [y_1 \leq h(x) < y_2]$. Set $\mu = \mathbb{E}[X]$, and $\sigma^2 = \mathrm{Var}[X]$. For simplicity we assume that $|I| \geq r/2$. Otherwise, we just apply the argument below with I replaced by $[r] \setminus I = [0, y_1) \cup [y_2, r)$, which we view as an interval in the cyclic ordering of $[r]$. We will partition $I = [y_1, y_2)$ into a constant number of intervals in such a way that our previous results yield Chernoff style concentration bound on the total weight of keys landing within each of these intervals. The desired result will follow.

To be precise, let $t > 0$ and $\gamma = O(1)$ be given. Let $P_1 = \{x \in \Sigma \mid \forall y \in \Sigma^{d-1} : (x, y) \in I\}$ and $I_1 = \{(x_1, \dots, x_d) \in \Sigma^d \mid x_1 \in P_1\}$. Whether or not $h(x) \in I_1$ for a key $x \in \Sigma^c$ depends solely on the most significant character of $h(x)$. With $X_1 = \sum_{x \in S} w_x \cdot [h(x) \in I_1]$, $\mu_1 = \mathbb{E}[X_1]$, and $\sigma_1^2 = \mathrm{Var}[X_1]$, we can therefore apply Theorem 2 to obtain that for any $t' > 0$ and $\gamma' = O(1)$,

$$\Pr[|X_1 - \mu_1| \geq t'] \leq C \exp(-\Omega(\sigma_1^2 \mathcal{C}(t'/\sigma_1^2))) + 1/u^{\gamma'} \leq C \exp(-\Omega(\sigma^2 \mathcal{C}(t'/\sigma^2))) + 1/u^{\gamma'}, \tag{6}$$

for some constant C. Here we used that $\sigma_1^2 \leq \sigma^2$ as $|I_1| \leq |I| \leq |\Sigma^d|/2$. Next, let $d_1 = \lg |\Sigma|$ and $d_2, \dots, d_\ell \in \mathbb{N}$ be such that for $2 \leq i \leq \ell$, it holds that $2^{d_i} \leq (2^{d_1 + d_2 + \cdots + d_i})^{1/4}$, and further $2^{d_1 + d_2 + \cdots + d_\ell} = |\Sigma|^d$. We may assume that u and hence $|\Sigma|$ is larger than some constant as otherwise the bound in Theorem 3 is trivial. It is then easy to check that we may choose ℓ and the $(d_i)_{2 \leq i \leq \ell}$ such that $\ell = O(\log d) = O(1)$. We will from now on consider elements of Σ^d as numbers written in binary or, equivalently, bit strings of length $d' := d_1 + \cdots + d_\ell$. For $i = 1, \dots, \ell$ we define a map $\rho_i : \Sigma^d \to [2]^{d_1 + \cdots + d_i}$ as follows. If $x = b_1 \dots b_{d'} \in [2]^{d'}$, then $\rho_i(x)$ is the length $d_1 + \cdots + d_i$ bit string $b_1 \dots b_{d_1 + \cdots + d_i}$ Set $J_1 = I$. For $i = 2, \dots, \ell$ we define $J_i \subseteq I$ and $I_i \subseteq I$ recursively as follows. First, we let $J_i = J_{i-1} \setminus I_{i-1}$. Second, we define I_i to consist of those elements of $x \in J_i$ such that if $y \in \Sigma^c$ has $\rho_i(y) = \rho_i(x)$, then $y \in J_i$. In other words, I_i consists of those elements of J_i that remain in J_i when the least significant $d_{i+1} + \cdots + d_\ell$ bits of x are changed in an arbitrary manner. It is readily checked that for $i = 1, \dots, \ell$, I_i is a disjoint union of two (potentially empty) intervals $I_i = I_i^{(1)} \cup I_i^{(2)}$ such that for each $j \in \{1, 2\}$ and $x, y \in I_i^{(j)}$, $\rho_i(x) = \rho_i(y)$. Moreover, the sets $(I_i)_{i=1}^{\ell}$ are pairwise disjoint and $I = \bigcup_{i=1}^{\ell} I_i$.

We already saw in (6) that we have Chernoff-style concentration for the total weight of balls landing in I_1. We now show that the same is true for $I_i^{(j)}$ for each $i = 2, \dots, \ell$ and $j \in \{0, 1\}$. So let such an i and j be fixed. Note that whether or not $h(x) \in I_i^{(j)}$, for a key $x \in \Sigma^c$, depends solely on the most significant $d_1 + \cdots + d_i$ bits of $h(x)$. Let $h' : \Sigma^c \to [2]^{d_1 + \cdots + d_i}$ be defined by $h'(x) = \rho_i(h(x))$. Then h' is itself a simple tabulation hash function and $h'(x)$ is obtained by removing the $d_{i+1} + \cdots + d_\ell$ least significant bits of $h(x)$. Letting $I' = \rho_i(I_i^{(j)})$, it thus holds that $h(x) \in I_i^{(j)}$ if and only if $h'(x) \in I'$. Let now $X_i^{(j)} = \sum_{x \in S} w_x \cdot [h(x) \in I_i^{(j)}]$, $\mu_i^{(j)} = \mathbb{E}\left[X_i^{(j)}\right]$, and $\sigma_1^2 = \mathrm{Var}\left[X_i^{(j)}\right] \leq \sigma^2$. As $|I'| \leq 2^{d_i} \leq (2^{d_1 + \cdots + d_i})^{1/4}$, we can apply Theorem 4 to conclude that for $t' > 0$ and $\gamma' = O(1)$,

$$\Pr[|X_i^{(j)} - \mu_i^{(j)}| \geq t'] \leq C \exp(-\Omega(\sigma_1^2 \mathcal{C}(t'/\sigma_1^2))) + 1/u^{\gamma'} \leq C \exp(-\Omega(\sigma^2 \mathcal{C}(t'/\sigma^2))) + 1/u^{\gamma'}. \tag{7}$$

Now applying (6) and (7) with $t' = t/(2\ell - 1)$ and $\gamma' = \gamma + \frac{\log(2\ell)}{\log u} = O(1)$, it follows that

$$\begin{aligned}\Pr[|X - \mu| \geq t] \leq & \Pr[|X_1 - \mu_1| \geq t'] + \sum_{i=2}^{\ell} \sum_{j=1}^{2} \Pr[|X_i^{(j)} - \mu_i^{(j)}| \geq t'] \leq 2C\ell \exp(-\Omega(\sigma^2 \mathcal{C}(t'/\sigma^2))) + 2\ell/u^{\gamma'} \\ = & O(\exp(-\Omega(\sigma^2\, \mathcal{C}(t/\sigma^2)))) + 1/u^{\gamma},\end{aligned}$$

as desired. □

3 Preliminaries

Before proceeding, we establish basic definitions and describe results from the literature which we will use.

3.1 Notation

Throughout the paper, we use the following general notation.

- We let $[n]$ denote the set $\{0, 1, \ldots, n-1\}$.
- For a statement or event Q we let $[Q]$ be the indicator variable on Q, i.e.,

$$[Q] = \begin{cases} 1, & Q \text{ occurred or is true,} \\ 0, & \text{otherwise.} \end{cases}$$

- Whenever $Y_0, \ldots, Y_{n-1} \in \mathbb{R}$ are variables and $i \in [n+1]$, we shall denote by $Y_{<i}$ the sum $\sum_{j<i} Y_j$. Likewise, whenever $A_0, \ldots, A_{n-1}$ are sets and $i \in [n+1]$, we shall denote by $A_{<i}$ the set $\bigcup_{j<i} A_j$.
- Suppose we have a hash function $h\colon A \to B$ with domain A and range B. We shall often associate weight and value functions with h as follows.
  - A function $w\colon A \to \mathbb{R}$ is called a *weight function*, corresponding to the idea that every ball or key $x \in A$ has an associated weight, $w(x) \in \mathbb{R}$. Occasionally, we shall write w_x for $w(x)$.
  - A function $v\colon A \times B \to \mathbb{R}$ is called a *value function*, with the interpretation that a key $x \in A$ yields a value $v(x, h(x))$ depending on the bin/hash value $h(x) \in B$.

  For weight functions $w : A \to \mathbb{R}$, a subset of balls, $S \subset A$, and a bin $y_0 \in B$, we will be interested in sums of the form $W = \sum_{x \in S} w(x)[h(x) = y_0]$, i.e., the total weight of the balls in S that are hashed to bin y_0. Defining the value function $v : A \times B \to \mathbb{R}$ by $v(x, y) = w(x)[y = y_0]$, W is exactly equal to $\sum_{x \in S} v(x, h(x))$, i.e., the total value obtained by the balls in S. From this perspective, value functions are more general objects than weight functions.

3.2 Probability Theory and Martingales

In the following, we introduce the necessary notions of probability theory. A note of caution is in order. The paper at hand relies on results from the theory of martingales to arrive at its conclusion. Working with martingales, we shall require probability theoretic notions of a fairly general and abstract character. For an introduction to measure and probability theory, see, for instance, [42].

For the most basic notation, let $(\Omega, \mathcal{F}, \Pr)$ be a probability space.

- Let $X_1, \ldots, X_n : \Omega \to \mathbb{R}$ be $\mathcal{F}$-measurable random variables. We denote by $\mathcal{G} = \sigma(X_1, \ldots, X_n) \subset \mathcal{F}$ the smallest σ-algebra such that $X_1, \ldots, X_n$ are all $\mathcal{G}$-measurable. We say that $\mathcal{G}$ is the *sigma algebra generated by* $X_1, \ldots, X_n$. Intuitively, $\sigma(X_1, \ldots, X_n)$ represents the collective information regarding the outcome of the joint distribution $(X_1, \ldots, X_n)$.

- Let $X : \Omega \to \mathbb{R}$ be an $\mathcal{F}$-measurable random variable, and let $\mathcal{G}$ be a σ-algebra with $\mathcal{G} \subset \mathcal{F}$. If $\mathbb{E}[|X|] < \infty$, we may define the random variable $\mathbb{E}[X \mid \mathcal{G}]$ to be the *conditional expectation* of X given $\mathcal{G}$. It is important to note that $\mathbb{E}[X \mid \mathcal{G}]$ is $\mathcal{G}$-measurable. In the context of the above notation, $\mathbb{E}[X \mid \sigma(X_1, \ldots, X_n)] = \mathbb{E}[X \mid X_1, \ldots, X_n]$ is the expectation of X as a function of the outcomes of $X_1, \ldots, X_n$.

We proceed to discuss martingales and martingale differences. For convenience we shall assume all random variables to be bounded, i.e., whenever X is a random variable, we assume that there exists a constant $M \geq 0$ such that $|X| \leq M$ almost surely.

Definition 3 (Filtration)**.** Let $(\Omega, \mathcal{P}(\Omega), \Pr)$ be a finite measure space. A sequence of σ-algebras, $(\mathcal{F}_i)_{i=0}^r$, is a *filtration* of $(\Omega, \mathcal{P}(\Omega), \Pr)$ if $\{\emptyset, \Omega\} = \mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \cdots \subseteq \mathcal{F}_r = \mathcal{P}(\Omega)$. We shall usually omit explicit reference to the background space.

Definition 4 (Adapted Sequence)**.** Let $(\mathcal{F}_i)_{i=0}^r$ be a filtration. A sequence of random variables $(X_i)_{i=0}^r$ is *adapted* to $(\mathcal{F}_i)_{i=0}^r$ if for every $i \in [r+1]$, X_i is $\mathcal{F}_i$-measurable. In that case, we say that $(X_i, \mathcal{F}_i)$ is an *adapted sequence.*

Definition 5 (Martingale)**.** A *martingale* is an adapted sequence, $(X_i, \mathcal{F}_i)$, satisfying that for every $i \in \{1, \ldots, r\}$, $\mathbb{E}[X_i \mid \mathcal{F}_{i-1}] = X_{i-1}$.

Definition 6 (Martingale Difference)**.** A *martingale difference* is a an adapted sequence, $(Y_i, \mathcal{F}_i)_{0=1}^r$, such that $Y_0 = 0$ almost surely and for every $i \in \{1, \ldots, r\}$, $\mathbb{E}[Y_i \mid \mathcal{F}_{i-1}] = 0$.

If $(X_i, \mathcal{F}_i)_{i=0}^r$ is a martingale, we may define the sequence of random variables $(Y_i)_{i=0}^r$ by $Y_0 = 0$ and $Y_i = X_i - X_{i-1}$ for $i = 1, \ldots, r$. Then $(Y_i, \mathcal{F}_i)_{i=0}^r$ is a martingale difference. Conversely, if $(Y_i, \mathcal{F}_i)_{i=0}^r$ is a martingale difference, a martingale $(X_i, \mathcal{F}_i)_{i=0}^r$ can be constructed by letting $X_i = Y_{<i+1} = \sum_{j \leq i} Y_j$. Under this correspondence, martingales and martingale differences are in a sense two sides of the same coin.

Concluding the section, we describe canonical constructions of a martingale and a martingale difference, respectively, that we shall use later on.

- Let X be a random variable and consider a filtration $(\mathcal{F}_i)_{i=0}^r$. We may define a martingale from X with respect to $(\mathcal{F}_i)_{i=0}^r$ by defining the sequence of random variables $(X_i)_{i=0}^r$ by $X_i = \mathbb{E}[X \mid \mathcal{F}_i]$ for each $i \in [r+1]$. Clearly, $\mathbb{E}[X_i \mid \mathcal{F}_{i-1}] = \mathbb{E}[X \mid \mathcal{F}_{i-1}] = X_{i-1}$, so $(X_i, \mathcal{F}_i)_{i=0}^r$ is indeed a martingale.

  We shall apply this construction in the following situation. Suppose we have random variables $Z_1, \ldots, Z_r$ taking values in the measure spaces $A_1, \ldots, A_r$ and denote by Z the joint distribution $(Z_1, \ldots, Z_r)$. For some function $f \colon A_1 \times \ldots A_r \to \mathbb{R}$, we wish to assess the value of $f(Z)$. We may then define the filtration $\mathcal{F}_i = \sigma(Z_1, \ldots, Z_i)$ for $i \in [r+1]$ and set $X_i = \mathbb{E}[f(Z) \mid \mathcal{F}_i]$. This yields a martingale $(X_i, \mathcal{F}_i)_{i=0}^r$ with $X_0 = \mathbb{E}[f(Z)]$ and $X_r = f(Z)$. This is known as a Doob martingale and the construction will be used in Section 5 to prove Theorem 5.

- Let $(Z_i, \mathcal{F}_i)_{i=0}^r$ be an adapted sequence and define $Y_0 = 0$ and $Y_i = Z_i - \mathbb{E}[Z_i \mid \mathcal{F}_{i-1}]$ for $i \in \{1, \ldots, r\}$. Then $(Y_i, \mathcal{F}_i)_{i=1}^r$ is a martingale difference. This construction is applied in Section 4 to prove Theorem 4.

3.3 Martingale Concentration Inequalities

In applications of probability theory, we often consider a sequence of random variables $X_0, \ldots, X_r$. If we are lucky, the random variables are independent, pair-wise independent, or a derivative thereof. It is unfortunately often the case, however, that there is no such independence notion that apply to $X_0, \ldots, X_r$. One reason that martingales have been as successful as they are, is that frequently, one may instead impose a martingale structure on the variables, and martingales satisfy many of the same theorems that independent variables do. In this exposition, we shall consider sums of the form $X = \sum_{i=0}^r X_i$ where the X_i are far from independent, yet we would like X to satisfy Chernoff-style bounds.

To this end, we state a martingale version of Bennett's inequality due to Fan et al [24]. The reader may note the similarity to Eq. (3).

Definition 7. We denote by $\mathcal{C} : (-1, \infty) \to [0, \infty)$ the function given by $\mathcal{C}(x) = (x+1)\ln(x+1) - x$.

Theorem 7 (Fan et al. [24])**.** *Let $\sigma > 0$ be given. Let $(X_i, \mathcal{F}_i)_{i=0}^r$ be a martingale difference such that almost surely $|X_i| \leq 1$ for all $i \in \{1, \ldots, r\}$ and $\sum_{i=1}^r \mathbb{E}\left[X_i^2 \mid \mathcal{F}_{i-1}\right] \leq \sigma^2$. Writing $X = \sum_{i=1}^r X_i$, it holds for any $t \geq 0$ that*

$$\Pr\left[X \geq t\right] \leq e^t \cdot \left(\frac{\sigma^2}{\sigma^2 + t}\right)^{\sigma^2 + t} .$$

Simple calculations yield the following corollary.

Corollary 8. *Suppose that $(X_i, \mathcal{F}_i)_{i=0}^r$ is a martingale difference and there exist $M, \sigma \geq 0$ such that $|X_i| \leq M$ for all $i \in \{1, \ldots, r\}$ and $\sum_{i=1}^r \mathbb{E}\left[X_i^2 \mid \mathcal{F}_{i-1}\right] \leq \sigma^2$. Define $X = \sum_{i=1}^r X_i$. For any $t \geq 0$ it holds that*

$$\Pr\left[X \geq t\right] \leq \exp\left(-\frac{\sigma^2}{M^2}\mathcal{C}\left(\frac{tM}{\sigma^2}\right)\right) ,$$

where $\mathcal{C}(x) = (x+1)\ln(x+1) - x$.

Finally, we present three lemmas describing the asymptotic behavior of $\mathcal{C}$. We omit the proofs of the first two since the results are standard and follow by elementary calculus.

Lemma 9. *For any $x \geq 0$*

$$\frac{1}{2}x\ln(x+1) \leq \mathcal{C}(x) \leq x\ln(x+1) .$$

For any $x \in [0, 1]$

$$\frac{1}{3}x^2 \leq \mathcal{C}(x) \leq \frac{1}{2}x^2 ,$$

where the right hand inequality holds for all $x \geq 0$.

Lemma 10. *For any $a \geq 0$. If $b \geq 1$ then*

$$b\mathcal{C}(a) \leq \mathcal{C}(ab) \leq b^2\mathcal{C}(a) .$$

If $0 \leq b \leq 1$ then

$$b^2\mathcal{C}(a) \leq \mathcal{C}(ab) \leq b\mathcal{C}(a) .$$

Note that as a corollary, if $b = \Theta(1)$ and $a \geq 0$, then $\mathcal{C}(ba) = \Theta(\mathcal{C}(a))$. The final lemma shows that the bound of Corollary 8 only gets worse when σ^2 or M is replaced by some larger number.

Lemma 11. *Let $a \geq 0$ be given. On $\mathbb{R}^+$, the following two functions are decreasing*

$$x \mapsto x\mathcal{C}\left(\frac{a}{x}\right) ,$$
$$x \mapsto \frac{\mathcal{C}(ax)}{x^2} .$$

Proof. Let $0 < x \leq y$ be given. We then observe that the first function is indeed decreasing since by the first bound of Lemma 10, $x\mathcal{C}(a/x) = x\mathcal{C}\left((a/y) \cdot (y/x)\right) \geq y\mathcal{C}\left(a/y\right)$. That the second function is decreasing follows from a similar argument. □

4 Analysis of Simple Tabulation

In this section, we analyze the simple tabulation hashing scheme. The section is divided in three parts. First, there will be an introductory section regarding simple tabulation hashing and associated notation. Second, we shall prove the sum of squares result (Eq. (5)). The final section presents a proof of Theorem 4. In order to make the exposition slightly simpler and more accessible, we postpone the argument that our concentration bounds are query invariant to Section 7.

4.1 Simple Tabulation Basics

Simple tabulation hashing as introduced by Zobrist [49] is defined as follows.

Definition 8 (Simple Tabulation Hashing)**.** Let Σ be an alphabet, $c \geq 1$ an integer, and $m = 2^k, k > 0$, a power of two. A simple tabulation hash function, $h\colon \Sigma^c \to [m]$, is a random variable taking values in the set of functions from Σ^c to $[m]$, chosen with respect to the following distribution. For each $j \in \{1, \dots, c\}$, let $h_j\colon \Sigma \to [m]$ be a fully random hash function, in other words, a uniformly random function from Σ to $[m]$. We evaluate h on the key $x = (x_1, \dots, x_c) \in \Sigma^c$ by computing $h(x) = \bigoplus_{j=1}^{c} h_j(x_j)$, where $\oplus$ denotes bitwise XOR.

Now, towards analyzing simple tabulation hashing, we add the following notation.

Definition 9 (Position Character)**.** Let Σ be an alphabet and $c \geq 1$ an integer. We call an element $\alpha = (a, y) \in \{1, \dots, c\} \times \Sigma$ a *position character* of Σ^c.

Let $h\colon \Sigma^c \to [m]$ be a simple tabulation hash function. We may consider a key $x = (x_1, \dots, x_c) \in \Sigma^c$ as a set of c position characters, $\{(1, x_1), \dots, (c, x_c)\} \subseteq \{1, \dots, c\} \times \Sigma$. Recall that $h(x) = \bigoplus_{i=1}^{c} h_i(x_i)$ for uniformly random functions $h_i\colon \Sigma \to [m]$. For a position character $\alpha = (a, y) \in \{1, \dots, c\} \times \Sigma$, we may overload notation and write $h(\alpha) = h_a(y)$. Extending this, for a set of position characters $A = \{\alpha_1, \dots, \alpha_n\} \subseteq \{1, \dots, c\} \times \Sigma^c$, $h(A) = \bigoplus_{i=1}^{n} h(\alpha_i)$. Note that this agrees with the correspondence between keys of Σ^c and sets of position characters mentioned before, since for $x = (x_1, \dots, x_c) \in \Sigma^c$, $h(x) = h(\{(1, x_1), \dots, (c, x_c)\})$. If finally $A, B \subset \{1, \dots, c\} \times \Sigma$ are sets of position characters we write $A \oplus B$ for the symmetric difference between A and B, i.e., $A \oplus B = (A \setminus B) \cup (B \setminus A)$. We note that for a simple tabulation hash function h, $h(A \oplus B) = h(A) \oplus h(B)$.

Definition 10 (Projection Onto an Index)**.** Let $c \geq 1$ be an integer and $i \in \{1, \dots, c\}$ be given. We denote by $\pi_i\colon \Sigma^c \to \{1, \dots, c\} \times \Sigma$ the projection onto the ith coordinate given by $\pi_i(x_1, \dots, x_c) = (i, x_i)$, i.e., projecting a key x to its ith position character. We extend this to sets of keys, such that for $S \subseteq \Sigma^c$, $\pi_i(S) = \{\pi_i(x) \mid x \in S\}$.

The following lemma by Thorup and Zhang [47] describes the independence of sets of position characters of Σ^c under a simple tabulation function $h\colon \Sigma^c \to [2^r]$. We provide a proof for completeness.

Lemma 12 (Thorup and Zhang [47])**.** *Let $h\colon \Sigma^c \to [2^r]$ be a simple tabulation hash function. For each $i \in \{1, \dots, t\}$, let $s_i \subseteq \{1, \dots, c\} \times \Sigma$ be a set of position characters of Σ^c. Let $j \in \{1, \dots, t\}$. If every subset of indices $B \subseteq \{1, \dots, t\}$ containing j satisfies $\bigoplus_{i \in B} s_i \neq \emptyset$, then the distribution of $h(s_j)$ is independent of the joint distribution $(h(s_i))_{i \neq j}$.*

Proof. Let $\mathbb{F}_2$ be the field $\mathbb{Z}/2\mathbb{Z}$ and V the $\mathbb{F}_2$-vector space $\mathbb{F}_2^{\{1,\dots,c\}\times\Sigma}$. For a set of position characters A, we define $v_A \in V$ as follows: For $(a, y) \in \{1, \dots, c\} \times \Sigma$ we let $v_A(a, y) = 1$ if and only if $(a, y) \in A$, and $v_A(a, y) = 0$ otherwise. Picking a random simple tabulation hash function $h : \Sigma^c \to [2^r]$ is equivalent to picking a random *linear* function $h' : V \to [2^r]$. Here $[2^r]$ is identified with the $\mathbb{F}_2$-vector space $\mathbb{F}_2^r$. Indeed, $(v_{\{\alpha\}})_{\alpha \in \{1,\dots,c\}\times\Sigma}$ forms a basis for V, and choosing a random linear map $h' : V \to [2^r]$ can be done by picking independent and uniformly random values for h' on the basis elements, and extending by linearity. To define h from h', we simply put $h(x) = \bigoplus_{\alpha \in x} h'(v_{\{\alpha\}})$ for a key $x \in \Sigma$ viewed as a set of position characters. Conversely, a simple tabulation hash function $h : \Sigma^c \to [2^r]$ uniquely extends to a linear map $h' : V \to [2^r]$. Now under this identification, the condition in the lemma is equivalent to v_{s_j} being linearly independent of the vectors $(v_{s_i})_{i \neq j}$. As h' is a random linear map, it follows by elementary linear algebra that $h'(v_{s_j}) = h(s_j)$ is independent of the joint distribution $(h'(v_{s_i}))_{i \neq j} = (h(s_i))_{i \neq j}$, as desired. □

4.2 Bounding the Sum of Squared Deviations

In the following section we shall prove the bound (5) of Theorem 4 from Section 2.1, stated independently here as Theorem 16. It is a technical, albeit crucial, step on the way to proving Theorem 4 itself. The foundation of the proof of Theorem 16 is a series of combinatorial observations regarding simple tabulation hashing.

Recall from Section 1.6.1 our general proof strategy when proving concentration bounds for simple tabulation hashing. For a set of keys $S \subseteq \Sigma^c$ to be hashed, we fix an ordering of the position characters of Σ^c. We then fix the hash table entries corresponding to the position characters one at a time according to this ordering. Crucial to the success of this strategy is fixing an ordering where each position character "decides" only a small part of the final outcome.

Definition 11 (Group of Keys)**.** Let $S \subseteq \Sigma^c$ be a set of keys and $A = \{\alpha \in x \mid x \in S\}$ be the set of position characters of the keys of S. For an enumeration or ordering of the position characters of A as $\{\alpha_1, \dots, \alpha_r\} = A$, we denote by $G_i \subseteq S$ the ith *group of keys* with respect to S and the ordering of the position characters. The set is given by $G_i = \{x \in S \mid \{\alpha_i\} \subseteq x \subseteq \{\alpha_1, \dots, \alpha_i\}\}$.

Put in other words, let $\prec$ denote the ordering on A, let x be a key of S, and let $\beta_1, \dots, \beta_c$ be the position characters of x such that $\beta_1 \prec \beta_2 \prec \cdots \prec \beta_c$, i.e., β_c is last in the ordering of A. Then $x \in G_i$ if and only if $\alpha_i = \beta_c$. In relation to simple tabulation, this has the following meaning. In the proof, we shall fix the values $h(\alpha_j)$ one at a time starting at $j = 1$ and ending at $j = r$. For every $x \in G_i$, the value of $h(x)$ is then undecided before $h(\alpha_i)$ is known, but is known once $h(\alpha_1), \dots, h(\alpha_i)$ are all fixed. In analyzing the contribution of each group to the final outcome of the process, we start by proving a generalization of a result from [38]. It says that if we assign each key a weight, it is always possible to choose the ordering of the position characters such that the total weight of each group is relatively small. The original lemma simply assigned weight 1 to every key.

Lemma 13. *Let $S \subseteq \Sigma^c$ be given and let $A = \{\alpha \in x \mid x \in S\}$ be the position characters of the keys of S. Let $w\colon \Sigma^c \to \mathbb{R}_{\geq 0}$ be a weight function. Then there exists an ordering of the position characters, $\{\alpha_1, \dots, \alpha_r\} = A$ such that for every $i \in \{1, \dots, r\}$, the group $G_i = \{x \in S \mid \{\alpha_i\} \subseteq x \subseteq \{\alpha_1, \dots \alpha_i\}\}$ satisfies*

$$\sum_{x \in G_i} w(x) \leq \left(\max_{x \in S} w(x)\right)^{1/c} \left(\sum_{x \in S} w(x)\right)^{1-1/c} .$$

Proof. We define the ordering recursively and backwards as $\alpha_r, \dots, \alpha_1$. Let $T_i = A \setminus \{\alpha_{i+1}, \dots, \alpha_r\}$ and $S_i = \{x \in S \mid x \subseteq T_i\}$. We prove that we can find an $\alpha_i \in T_i$ such that

$$G_i = \{x \in S_i \mid \alpha_i \in x\} \ ,$$

satisfies

$$\sum_{x \in G_i} w(x) \leq \left(\max_{x \in S_i} w(x)\right)^{1/c} \left(\sum_{x \in S_i} w(x)\right)^{1-1/c} ,$$

which will establish the claim. Let B_k be the set of position characters at position k contained in T_i, i.e., $B_k = \{(k, y) \in T_i\} = \pi_k(T_i)$. Then as $\prod_{k=1}^{c} |B_k| \geq |S_i|$, we have $|B_k| \geq |S_i|^{1/c}$ for some k.

Since each key of S_i contains at most one position character from B_k, we can choose α_i such that

$$\sum_{x \in G_i} w(x) \leq \frac{\sum_{x \in S_i} w(x)}{|B_k|} \leq \frac{\sum_{x \in S_i} w(x)}{|S_i|^{1/c}} \leq \left(\max_{x \in S_i} w(x)\right)^{1/c} \left(\sum_{x \in S_i} w(x)\right)^{1-1/c} .$$

□

Suppose we have keys $x_1, \dots, x_t \in \Sigma^c$. It follows as a corollary of Lemma 12 that with a simple tabulation hash function $h\colon \Sigma^c \to [m]$, the values $h(x_1), \dots, h(x_t)$ are completely independent if and only if there does not exist a subset of indices $B \subseteq \{1, \dots, t\}$ with $\bigoplus_{i \in B} x_i = \emptyset$. In this vein, it turns out to be natural, given sets of keys $A_1, \dots, A_\ell \subseteq \Sigma^c$, to bound the number of tuples $x_1 \in A_1, \dots, x_\ell \in A_\ell$ with $\bigoplus_{i=1}^{\ell} x_i = \emptyset$. This is the content of Lemma 3 of [16]. We prove the following generalization of this result, which deals with weighted keys. Note that the statements would be identical if each key was assigned the weight 1.

Lemma 14. *Let $\ell \in \mathbb{N}$ be even, $w_1, \ldots, w_\ell \colon \Sigma^c \to \mathbb{R}$ be weight functions, and $A_1, \ldots, A_\ell \subseteq \Sigma^c$ be sets of keys. Then*

$$\sum_{\substack{x_1 \in A_1, \ldots, x_\ell \in A_\ell \\ \bigoplus_{k=1}^{\ell} x_k = \emptyset}} \prod_{k=1}^{\ell} w_k(x_k) \le ((\ell-1)!!)^c \cdot \prod_{k=1}^{\ell} \sqrt{\sum_{x \in A_k} w_k(x)^2}.$$

Proof. For every $(x_1, \ldots, x_\ell) \in A_1 \times \cdots \times A_\ell$ satisfying $\bigoplus_{k=1}^{\ell} x_k = \emptyset$ we have $\bigoplus_{k=1}^{\ell} \{\pi(x_k, c)\} = \emptyset$. This implies that each character in the c-th position occurs an even number of times in $(x_1, \ldots, x_\ell)$. Thus, for any such tuple we can partition the indices $1, \ldots, \ell$ into pairs $(i_1, j_1), \ldots, (i_{\ell/2}, j_{\ell/2})$ satisfying $\pi(x_{i_k}, c) = \pi(x_{j_k}, c)$ for every $k \in \{1, \ldots, \ell\}$. Fix such a partition and let $X \subseteq A_1 \times \cdots \times A_\ell$ be the set

$$X = \{(x_1, \ldots, x_\ell) \in A_1 \times \ldots \times A_\ell \mid \forall k \in \{1, \ldots, \ell/2\} \colon \pi(x_{i_k}, c) = \pi(x_{j_k}, c)\}.$$

We proceed by induction on c.

For $c = 1$, $\pi(x_{i_k}, c) = \pi(x_{j_k}, c)$ implies $x_{i_k} = x_{j_k}$ such that

$$X = \{(x_1, \ldots, x_\ell) \in A_1 \times \ldots \times A_\ell \mid \forall k \in \{1, \ldots, \ell/2\} \colon x_{i_k} = x_{j_k}\}.$$

Thus, by the Cauchy-Schwartz inequality,

$$\begin{aligned}
\sum_{(x_1, \ldots, x_\ell) \in X} \prod_{k=1}^{\ell} w_k(x_k) &= \prod_{k=1}^{\ell/2} \sum_{x \in A_{i_k} \cap A_{j_k}} w_{i_k}(x) w_{j_k}(x) \\
&\le \prod_{k=1}^{\ell/2} \left(\sqrt{\sum_{x \in A_{i_k}} w_{i_k}(x)^2} \cdot \sqrt{\sum_{x \in A_{j_k}} w_{j_k}(x)^2} \right) \\
&\le \prod_{k=1}^{\ell} \sqrt{\sum_{x \in A_k} w_k(x)^2}.
\end{aligned}$$

Since this is true for any partition into pairs, $(i_1, j_1), \ldots, (i_{\ell/2}, j_{\ell/2})$, there are exactly $(\ell-1)!!$ such partitions, and every term in the original sum is counted by some partition, we get the desired bound for $c = 1$.

Let $c > 1$ and assume that the statement holds when each key has $< c$ characters. For each $a \in \Sigma$ and $k \in \{1, \ldots, \ell\}$ define the set

$$A_k[a] = \{x \in A_k \mid \pi(x, c) = a\}.$$

Fixing the last character of each pair in our partition by picking $a_1, \ldots, a_{\ell/2} \in \Sigma$ and considering the sets $A_{i_k}[a_k]$ and $A_{j_k}[a_k]$, we can consider the keys of $\prod_{k=1}^{\ell/2} A_{i_k}[a_k] \times A_{j_k}[a_k]$ as only having $c - 1$ characters, which allows us to apply the induction hypothesis. This yields

$$\begin{aligned}
\sum_{\substack{(x_1,\dots,x_\ell)\in X\\ \bigoplus_{k=1}^{\ell} x_k=\emptyset}} \prod_{k=1}^{\ell} w_k(x_k) &= \sum_{(a_k)_{k=1}^{\ell/2}\in\Sigma^{\ell/2}} \left(\sum_{\substack{(x_{i_k},x_{j_k})_{k=1}^{\ell/2}\in\prod_{k=1}^{\ell/2} A_{i_k}[a_k]\times A_{j_k}[a_k]\\ \bigoplus_{k=1}^{\ell} x_k=\emptyset}} \prod_{k=1}^{\ell/2} w_{i_k}(x_{i_k})w_{j_k}(x_{j_k}) \right)\\
&\le \sum_{(a_k)_{k=1}^{\ell/2}\in\Sigma^{\ell/2}} \left(((\ell-1)!!)^{c-1}\cdot \prod_{k=1}^{\ell/2}\left(\sqrt{\sum_{x\in A_{i_k}[a_k]} w_{i_k}(x)^2}\cdot\sqrt{\sum_{x\in A_{j_k}[a_k]} w_{j_k}(x)^2}\right)\right)\\
&= ((\ell-1)!!)^{c-1}\cdot\prod_{k=1}^{\ell/2}\left(\sum_{a\in\Sigma}\left(\sqrt{\sum_{x\in A_{i_k}[a]} w_{i_k}(x)^2}\cdot\sqrt{\sum_{x\in A_{j_k}[a]} w_{j_k}(x)^2}\right)\right)\\
&\le ((\ell-1)!!)^{c-1}\cdot\prod_{k=1}^{\ell/2}\left(\sqrt{\sum_{a\in\Sigma}\sum_{x\in A_{i_k}[a]} w_{i_k}(x)^2}\cdot\sqrt{\sum_{a\in\Sigma}\sum_{x\in A_{j_k}[a]} w_{j_k}(x)^2}\right)\\
&= ((\ell-1)!!)^{c-1}\cdot\prod_{k=1}^{\ell/2}\left(\sqrt{\sum_{x\in A_{i_k}} w_{i_k}(x)^2}\cdot\sqrt{\sum_{x\in A_{j_k}} w_{j_k}(x)^2}\right),
\end{aligned}$$

where the last inequality follows from the Cauchy-Schwartz inequality. Since the indices can be partitioned into pairs in $(\ell-1)!!$ ways, the same argument as in the induction start yields

$$\sum_{\substack{x_1\in A_1,\dots,x_\ell\in A_\ell\\ \bigoplus_{k=1}^{\ell} x_k=\emptyset}} \prod_{k=1}^{\ell} w_k(x_k) \le ((\ell-1)!!)^c\cdot\prod_{k=1}^{\ell}\sqrt{\sum_{x\in A_k} w_k(x)^2},$$

which was the desired conclusion. □

The following rather technical lemma bounds the moments of collisions between sets of keys. However, we shall dwell on it for a moment as it reflects considerations that will come up repeatedly going forward. Consider a simple tabulation function $h\colon\Sigma^c\to[m]$ and a value function $v\colon\Sigma^c\times[m]\to\mathbb{R}$. Hashing the keys of some subsets $A_1,\dots,A_n\subseteq\Sigma^c$ into $[m]$ using h, we are interested in the sums $X_i=\sum_{x\in A_i} v(x,h(x))$ for $1\le i\le n$ and, in particular, in properties of the joint distribution $(X_1,\dots,X_n)$. Here, the actual values of X_i are not as important as how much X_i deviates form its mean. For $1\le i\le n$, We thus consider the variables

$$Y_i = X_i-\mathbb{E}[X_i] = \sum_{x\in A_i}\sum_{b\in[m]} v(x,b)\left([h(x)=b]-\frac{1}{m}\right),$$

and for a level of generality required for proving the main theorems of this section, we consider the *shifted* variables

$$Y_i^{(j)} = \sum_{x\in A_i}\sum_{b\in[m]} v(x,b)\left([h(x)=j\oplus b]-\frac{1}{m}\right),$$

for $j\in[m]$, corresponding to shifting the hash function h by $j\in[m]$.

Lemma 15. *Let $h\colon\Sigma^c\to[m]$ be a simple tabulation hash function and $v\colon\Sigma^c\times[m]\to\mathbb{R}$ a value function. Let $Q=\{i\in[m]\mid \exists x\in\Sigma^c: v(x,i)\neq 0\}$ be the support of v and write $\ell=|Q|$. Let $n\in\mathbb{N}$ and $A_1,\dots,A_n\subseteq\Sigma^c$. For every $i\in\{1,\dots,n\}$ and $j\in[m]$ define the random variable*

$$Y_i^{(j)} = \sum_{x\in A_i}\sum_{b\in Q} v(x,b)\left([h(x)=j\oplus b]-\frac{1}{m}\right),$$

and set

$$T = \sum_{\substack{j_1,\dots,j_n \in [m] \\ \bigoplus_{j=1}^n j_k = 0}} \prod_{k=1}^n Y_k^{(j_k)} \, .$$

Then for every constant $t \in \mathbb{N}$,

$$\left|\mathbb{E}\left[T^t\right]\right| = O_{t,n,c}\left(\ell^{tn/2} \prod_{k=1}^n \left(\sum_{x \in A_k} \sum_{b \in Q} v(x,b)^2\right)^{t/2}\right) \, .$$

Proof. We rewrite T as follows

$$\begin{aligned}
T &= \sum_{\substack{j_1,\dots,j_n \in [m] \\ \bigoplus_{k=1}^n j_k = 0}} \prod_{k=1}^n Y_k^{(j_k)} \\
&= \sum_{\substack{j_1,\dots,j_n \in [m] \\ \bigoplus_{k=1}^n j_k = 0}} \left(\prod_{k=1}^n \sum_{x \in A_k} \sum_{b \in Q} v(x,b)\left([h(x) = j_k \oplus b] - \frac{1}{m}\right)\right) \\
&= \sum_{(x_1,\dots,x_n) \in A_1 \times \dots \times A_n} \sum_{b_1,\dots b_n \in Q} \left(\sum_{\substack{j_1,\dots,j_n \in [m] \\ \bigoplus_{k=1}^n j_k = 0}} \left(\prod_{k=1}^n \left(v(x_k,b_k)\left([h(x_k) = j_k \oplus b_k] - \frac{1}{m}\right)\right)\right)\right) \\
&= \sum_{(x_1,\dots,x_n) \in A_1 \times \dots \times A_n} \sum_{b_1,\dots b_n \in Q} \left(\left(\prod_{k=1}^n v(x_k,b_k)\right) \cdot \left(\sum_{\substack{j_1,\dots,j_n \in [m] \\ \bigoplus_{k=1}^n j_k = 0}} \left(\prod_{k=1}^n \left([h(x_k) = j_k \oplus b_k] - \frac{1}{m}\right)\right)\right)\right) \\
&= \sum_{(x_1,\dots,x_n) \in A_1 \times \dots \times A_n} \sum_{b_1,\dots b_n \in Q} \left(\left(\prod_{k=1}^n v(x_k,b_k)\right) \cdot \left(\left[\bigoplus_{k=1}^n h(x_k) = \bigoplus_{k=1}^n b_k\right] - \frac{1}{m}\right)\right)
\end{aligned}$$

Here the last equality is derived by observing that for fixed $(x_1,\dots,x_n) \in A_n \times \dots \times A_n$ and fixed $b_1,\dots b_n \in Q$,

$$\sum_{\substack{j_1,\dots,j_n \in [m] \\ \bigoplus_{k=1}^n j_k = 0}} \left(\prod_{k=1}^n \left([h(x_k) = j_k \oplus b_k] - \frac{1}{m}\right)\right) = \sum_{\substack{j_1,\dots,j_n \in [m] \\ \bigoplus_{k=1}^n j_k = 0}} \left(\sum_{B \subseteq \{1,\dots,n\}} (-m)^{-(n-|B|)} \prod_{k \in B} [h(x_k) = j_k \oplus b_k]\right)$$

and since for $\emptyset \subseteq B \subsetneq \{1,\dots,n\}$ there are exactly $m^{n-|B|-1}$ tuples $(j_1,\dots,j_n) \in [m]^n$ satisfying $j_k \oplus b_k = h(x_k)$ for every $k \in B$ and $\bigoplus_{k=1}^n j_k = 0$, we get

$$\begin{aligned}
\sum_{\substack{j_1,\dots,j_n \in [m] \\ \bigoplus_{k=1}^n j_k = 0}} \left(\prod_{k=1}^n \left([h(x_k) = j_k \oplus b_k] - \frac{1}{m}\right)\right) &= \sum_{\substack{j_1,\dots,j_n \in [m] \\ \bigoplus_{k=1}^n j_k = 0}} \left(\prod_{k=1}^n [h(x_k) = j_k \oplus b_k]\right) + \frac{1}{m} \sum_{B \subsetneq \{1,\dots,n\}} (-1)^{n-|B|} \\
&= \left[\bigoplus_{k=1}^n h(x_k) = \bigoplus_{k=1}^n b_k\right] + \frac{1}{m} \sum_{B \subsetneq \{1,\dots,n\}} (-1)^{n-|B|}.
\end{aligned}$$

By the principle of inclusion-exclusion, the last term is $-\frac{1}{m}$, which concludes the rearrangement.

Write $S = A_1 \times \ldots \times A_n$ and let $f\colon S \to \mathbb{R}$ be the function

$$f(x_1,\ldots,x_n) = \sum_{b_1,\ldots b_n \in Q} \left(\left(\prod_{k=1}^{n} v(x_k,b_k) \right) \cdot \left(\left[\bigoplus_{k=1}^{n} h(x_k) = \bigoplus_{k=1}^{n} b_k \right] - \frac{1}{m} \right) \right) .$$

By the above rearrangement, we have $T^t = \sum_{(s_i)_{i\in[t]} \in S^t} \prod_{i=1\in[t]} f(s_i)$, such that,

$$\mathbb{E}\left[T^t\right] = \sum_{(s_i)_{i=1}^t \in S^t} \mathbb{E}\left[\prod_{i=1}^{t} f(s_i)\right] .$$

Now, for a t-tuple $(s_i)_{i=1}^t \in S^t$, we overload notation by for a subset $T \subseteq \{1,\ldots,t\}$ defining $\bigoplus_{i\in T} s_i = \bigoplus_{i\in T}\bigoplus_{j=1}^{n}(s_i)_j$, where we still think of the keys $(s_i)_j$ as sets of input characters, and where $\oplus$ is the symmetric difference. Let $(s_i)_{i=1}^t \in S^t$ and let $T_1,\ldots,T_r \subseteq \{1,\ldots,t\}$ be all subsets of indices satisfying $\bigoplus_{i\in T_j} s_i = \emptyset, 1 \le j \le t$. If for some $i \in \{1,\ldots,t\}$, $i \notin \bigcup_{j=1}^{r} T_i$ then by Lemma 12, $h(s_i)$ is independent of the joint distribution $(h(s_j))_{j\neq i}$ and uniformly distributed in $[m]$. It follows that $f(s_i)$ is independent of the joint distribution $(f(s_j))_{j\neq i}$. Since it further holds that $\mathbb{E}[f(s_i)] = 0$, this implies

$$\mathbb{E}\left[\prod_{j=1}^{t} f(s_j)\right] = \mathbb{E}[f(s_i)] \cdot \mathbb{E}\left[\prod_{j\neq i} f(s_j)\right] = 0 .$$

Hence, we shall only sum over the t-tuples $(s_i)_{i=1}^t \in S^t$ satisfying that there exist subsets of indices $T_1,\ldots,T_r \subseteq \{1,\ldots,t\}$ such that $\bigoplus_{i\in T_j} s_i = \emptyset$ for every $j \in \{1,\ldots,r\}$ and $\bigcup_{j=1}^{r} T_j = \{1,\ldots,t\}$.

Fix such subsets $T_1,\ldots,T_r \subseteq \{1,\ldots,t\}$ and for $i \in \{1,\ldots,r\}$ let $B_i = T_i \setminus \left(\bigcup_{j<i} T_j\right)$. Then we can write

$$\begin{aligned} &\sum_{\substack{(s_i)_{i=1}^t \in S^t \\ \forall j\in\{1,\ldots,r\}\colon \bigoplus_{i\in T_j} s_i = \emptyset}} \prod_{i=1}^{t} f(s_i) \\ &= \sum_{\substack{(s_i)_{i\in\{1,\ldots,t\}\setminus B_r} \in S^{t-|B_r|} \\ \forall j\in\{1,\ldots,r-1\}\colon \bigoplus_{i\in T_j} s_i = \emptyset}} \prod_{i\in\{1,\ldots,t\}\setminus B_r} f(s_i) \sum_{\substack{(s_i)_{i\in B_r} \in S^{|B_r|} \\ \bigoplus_{i\in B_r} s_i = \bigoplus_{i\in T_r\setminus B_r} s_i}} \prod_{i\in B_r} f(s_i) \end{aligned} \tag{8}$$

Now fix $(s_i)_{i\in\{1,\ldots,t\}\setminus B_r} \in S^{t-|B_r|}$ such that for all $j \in \{1,\ldots,r-1\}$ it holds that $\bigoplus_{i\in T_j} s_i = \emptyset$. We wish to upper bound the inner sum in (8) for this choice of $(s_i)_{i\in\{1,\ldots,t\}\setminus B_r}$. In order to do this, observe that for $s = (x_1,\ldots,x_n) \in S$ we always have

$$|f(s)| \le \sum_{b_1,\ldots b_n \in Q} \prod_{k\in[n]} |v(x_k,b_k)| = \prod_{k=1}^{n} \left| \sum_{b\in Q} v(x_k,b) \right| \le \prod_{k=1}^{n} \sqrt{\ell \sum_{b\in Q} v(x_k,b)^2} ,$$

by the QA-inequality. We now wish to combine this bound with Lemma 14 to obtain a bound on the inner sum in (8). For this, we define let $\ell = |T_r|n$ and define sets of keys $F_1,\ldots,F_\ell$ and weight functions $w_1,\ldots,w_\ell : \Sigma^c \to \mathbb{R}$ as follows. Enumerate $T_r = \{i_1,\ldots,i_{|T_r|}\}$ such that $\{i_1,\ldots,i_{|B_r|}\} = B_r$. Now for $0 \le k < |B_r|$ and $1 \le j \le n$ we define $F_{kn+j} = A_j$. We further define the weight function $w_{kn+j} : \Sigma^c \to \mathbb{R}$ by $w_{kn+j}(x) = \sqrt{\ell \sum_{b\in Q} v(x,b)^2}$ for $x \in \Sigma^c$. Observe that these weight functions are all identical. Secondly, for $|B_r| \le k < |T_r|$ and $1 \le j \le n$, we define $F_{kn+j} = \{s_{i_k}(j)\}$, and $w_{kn+j} : \Sigma^c \to \mathbb{R}$ by $w_{kn+j}(x) = 1$ for all $x \in \Sigma^c$. Then,

$$\left| \sum_{\substack{(s_i)_{i\in B_r} \in S^{|B_r|} \\ \bigoplus_{i\in B_r} s_i = \bigoplus_{i\in T_r\setminus B_r} s_i}} \prod_{i\in B_r} f(s_i) \right| \le \sum_{\substack{x_1\in B_1,\ldots,x_\ell\in B_\ell \\ \bigoplus_{k=1}^{\ell} x_k = \emptyset}} \prod_{k=1}^{\ell} w_k(x_k) \le (n|T_r|-1)!!)^c \prod_{k=1}^{n} \left(\ell \cdot \sum_{x\in A_k} \sum_{b\in Q} v(x,b)^2 \right)^{|B_r|/2} ,$$

where the last inequality follows from Lemma 14. Note that this upper bound does not depend on the choice of $(s_i)_{i\in\{1,\dots,t\}\setminus B_r} \in S^{t-|B_r|}$ in the outer sum in (8). Repeating this argument another $r-1$ times, and using that $\{1,\dots,t\}$ is the disjoint union of $B_1,\dots,B_r$, we obtain that

$$\left| \sum_{\substack{(s_i)_{i=1}^t \in S^t \\ \forall j\in\{1,\dots,r\}:\ \bigoplus_{i\in T_j} s_i=\emptyset}} \prod_{i=1}^t f(s_i) \right| \le ((nt-1)!!)^{cr} \cdot \prod_{k=1}^n \left(\ell \sum_{x\in A_k}\sum_{b\in Q} v(x,b)^2 \right)^{t/2} .$$

Since there are at most 2^{2^t} ways of choosing r and the subsets $T_1,\dots,T_r$ and since $r\le 2^t$, summing over these choices yields

$$\begin{aligned} \left|\mathbb{E}\left[T^t\right]\right| &\le 2^{2^t}((nt-1)!!)^{cr} \cdot \prod_{k=1}^n \left(\ell \sum_{x\in A_k}\sum_{b\in Q} v(x,b)^2 \right)^{t/2} \\ &\le O_{t,n,c}\left(\ell^{nt/2} \prod_{k=1}^n \left(\sum_{x\in A_k}\sum_{b\in Q} v(x,b)^2 \right)^{t/2} \right) . \end{aligned}$$

□

We are now ready to prove the main theorem of the subsection, a bound on the sum of squared deviations of the value function from its deviation when shifting by every $j\in[m]$, the second part of Theorem 4. As described in Section 2.1, this bound is an important ingredient in the proof of the first part of Theorem 4. Namely, in our inductive proof, it bounds the variance of the value obtained from the keys of one of the groups G_i when the keys from this group are shifted by a uniformly random XOR with $h(\alpha_i)$.

Theorem 16. *Let $h\colon \Sigma^c \to [m]$ be a simple tabulation hash function and $S\subseteq \Sigma^c$ a set of keys. Let $v\colon \Sigma^c\times[m]\to[-1,1]$ be a value function such that the set $Q=\{i\in[m]\mid \exists x\in\Sigma^c : v(x,i)\neq 0\}$ satisfies $|Q|\le m^\varepsilon$, where $\varepsilon<\frac14$ is a constant. For $j\in[m]$ define the random variable $V_j=\sum_{x\in S} v(x,h(x)\oplus j)$ and let $\mu=\mathbb{E}[V_j]$, noting that this is independent of j. For any $\gamma\ge 1$,*

$$\Pr\left[\sum_{j\in[m]} (V_j-\mu)^2 > C_\gamma^c \sum_{x\in S}\sum_{k\in[m]} v(x,k)^2 \right] = O_{\gamma,\varepsilon,c}(n/m^\gamma) \tag{9}$$

where $C_\gamma = 3\cdot 2^6\cdot\gamma^2$ and this bound is query invariant up to constant factors.

Proof. First, note that we may write

$$V_j-\mu = \sum_{x\in S}\sum_{k\in Q} v(x,k)[h(x)=j\oplus k] - \frac{1}{m}\sum_{x\in S}\sum_{k\in Q} v(x,k) = \sum_{x\in S}\sum_{k\in Q} v(x,k)\left([h(x)=j\oplus k]-\frac{1}{m}\right) \tag{10}$$

Now, define $v'(x)=\sum_{k\in Q} v(x,k)^2$ and for $X\subseteq\Sigma^c$ we let $v'(X)=\sum_{x\in X} v'(x)$ and define $v'_\infty(X)=\max_{x\in X} v'(x)$. Now applying Lemma 13 with respect to v' we get position characters $\alpha_1,\dots,\alpha_r$ with corresponding groups $G_1,\dots,G_r$, such that, $\cup_{i=1}^r G_i = S$ and for every $i\in\{1,\dots,r\}$, $v'(G_i)\le v'(S)^{1-1/c} v'_\infty(S)^{1/c}$. For $i\in\{1,\dots,r\}, j\in[m]$ we define the random variables

$$X_i^{(j)} = \sum_{x\in G_i}\sum_{k\in Q} v(x,k)\left([h(x\setminus\alpha_i)=j\oplus k]-\frac{1}{m}\right), \qquad Y_i^{(j)} = X_i^{(j\oplus h(\alpha_i))},$$

where we recall that $x \setminus \alpha_i$ denotes the set containing the position characters of x except α_i. Notice that by (10), $V_j - \mu = \sum_{i\in[r]} Y_i^{(j)}$. Writing $V = \sum_{j\in[m]}(V_j-\mu)^2 = \sum_{j\in[m]}\left(\sum_{i\in[r]} Y_i^{(j)}\right)^2$, the statement we wish to prove is

$$\Pr\left[V > C_\gamma^c v'(S)\right] \le O_{\gamma,c}\left(|S|\, m^{-\gamma}\right) .$$

We proceed by induction on c. The induction start, $c = 1$, and the induction step are almost identical, so we carry them out in parallel. Note that when $c = 1$ each group has size at most 1, i.e. $|G_i| \le 1$ for every $i \in \{1, \dots, r\}$.

Let $\gamma \ge 1$ be fixed. We write

$$V = \underbrace{\sum_{j\in[m]}\sum_{i=1}^{r}\left(Y_i^{(j)}\right)^2}_{V_1} + \underbrace{\sum_{j\in[m]}\sum_{i=1}^{r} Y_i^{(j)} Y_{<i}^{(j)}}_{V_2} \tag{11}$$

and bound V_1 and V_2 separately starting with V_1.

Interchanging summations, $V_1 = \sum_{i=1}^r \sum_{j\in[m]}\left(Y_i^{(j)}\right)^2$. In the case $c = 1$, let $i \in \{1, \dots, r\}$ be given. If $|G_i| = 0$, $\sum_{j\in[m]}\left(Y_i^{(j)}\right)^2 = 0$. If on the other hand $G_i = \{x_i\}$ for some $x_i \in \Sigma^c$,

$$\begin{aligned}
\sum_{j\in[m]}\left(Y_i^{(j)}\right)^2 &= \sum_{j\in[m]}\left(\sum_{k\in Q} v(x_i,k)\left([h(x_i) = j \oplus k] - \frac{1}{m}\right)\right)^2 \\
&= \sum_{j\in[m]}\left(\sum_{k\in[m]} v(x_i,k)\left([h(x_i)\oplus j = k] - \frac{1}{m}\right)\right)^2 \\
&= \sum_{j\in[m]}\left(\sum_{k\in[m]} v(x_i,k)\left([j = k] - \frac{1}{m}\right)\right)^2 \\
&= \sum_{j\in[m]}\left(v(x_i,j) - \frac{1}{m}\sum_{k\in[m]} v(x_i,k)\right)^2 \\
&\le \sum_{j\in[m]} v(x_i,j)^2
\end{aligned}$$

where the last inequality follows from the inequality $\mathbb{E}\left[(X - \mathbb{E}[X])^2\right] \le \mathbb{E}\left[X^2\right]$. Thus, we always have $V_1 \le v'(S) \le \frac{C_\gamma^c}{2} v'(S)$. In the case $c > 1$ we observe that the keys of G_i have a common position character. Hence, we can apply the induction hypothesis on the keys of G_i with the remaining $c-1$ position characters to conclude that

$$\Pr\left[\sum_{j\in[m]}\left(Y_i^{(j)}\right)^2 > C_\gamma^{c-1} v'(G_i)\right] \le O_{\gamma,c}(|G_i| m^{-\gamma}) .$$

By a union bound,

$$\Pr\left[V_1 > \frac{C_\gamma^c}{2} v'(S)\right] \le \Pr\left[V_1 > C_\gamma^{c-1} v'(S)\right] \le \sum_{i\in[r]} O_{\gamma,c}(|G_i| m^{-\gamma}) = O_{\gamma,c}(|S| m^{-\gamma}) . \tag{12}$$

Next we proceed to bound V_2. For $0 \le i \le r$ define $Z_i = \sum_{j\in[m]} Y_i^{(j)} Y_{<i}^{(j)}$ with $Z_0 = 0$ and $\mathcal{F}_i = \sigma((h(\alpha_j))_{j=1}^i)$ with $\mathcal{F}_0 = \{\emptyset, \Omega\}$. As $Y_{<i}^{(j)}$ is $\mathcal{F}_{i-1}$ measurable for $j \in [m]$ it holds that

$$\mathbb{E}\left[Z_i \mid \mathcal{F}_{i-1}\right] = \sum_{j\in[m]} \mathbb{E}\left[Y_i^{(j)} \,\middle|\, \mathcal{F}_{i-1}\right] Y_{<i}^{(j)} = 0 ,$$

and so $(Z_i, \mathcal{F}_i)_{i=0}^r$ is a martingale difference. We will define a modified martingale difference $(Z'_i, \mathcal{F}_i)_{i=0}^r$ recursively as follows: We define the events A_i, B_i and C_i for $i \in \{1, \dots, r\}$ as

$$\begin{aligned}
A_i &= \bigcap_{k=1}^{i} \left(\sum_{j\in[m]} \left(Y_k^{(j)}\right)^2 \le C_\gamma^{c-1} v'(G_k) \right) , \\
B_i &= \bigcap_{k=1}^{i} \left(\mathrm{Var}\left[Z_k \mid \mathcal{F}_{k-1} \right] \le m^{-1/2} v'(G_k) v'(G_{<k}) \right) , \\
C_i &= \left(\max_{1\le k\le i} \{Z'_{<k}\} \le \frac{C_\gamma^c}{2} v'(S) \right) .
\end{aligned}$$

Finally, we let $Z'_i = [A_i \cap B_i \cap C_i] \cdot Z_i$. Clearly $B_i, C_i \in \mathcal{F}_{i-1}$. To see that this is also the case for A_i we note that for $k \le i$,

$$\sum_{j\in[m]} (Y_k^{(j)})^2 = \sum_{j\in[m]} (X_k^{(j\oplus h(\alpha_k))})^2 = \sum_{j\in[m]} (X_k^{(j)})^2 ,$$

and as each $X_k^{(j)}$ is $\mathcal{F}_{k-1}$-measurable it follows that $A_i \in \mathcal{F}_{i-1}$. Now, as $[A_i \cap B_i \cap C_i]$ is $\mathcal{F}_{i-1}$-measurable,

$$\mathbb{E}\left[Z'_i \mid \mathcal{F}_{i-1} \right] = [A_i \cap B_i \cap C_i] \mathbb{E}\left[Z_i \mid \mathcal{F}_{i-1} \right] = 0,$$

which implies that $(Z'_i, \mathcal{F}_i)_{i=0}^r$ is a martingale difference.

If A_r, B_r, and C_r all occur then $\sum_{i=1}^r Z_i = \sum_{i=1}^r Z'_i$. In particular

$$\Pr\left[V_2 > \frac{C_\gamma^c}{2} v'(S) \right] = \Pr\left[\sum_{i\in[r]} Z_i > \frac{C_\gamma^c}{2} v'(S) \right] \le \Pr\left[\sum_{i=1}^{r} Z'_i > \frac{C_\gamma^c}{2} v'(S) \right] + \Pr\left[A_r^c \cup B_r^c \cup C_r^c \right] .$$

If C_r does not occur then $\sum_{i\in[r]} Z'_i > \frac{C_\gamma^c}{2} v'(S)$ so a union bound yields

$$\Pr\left[V_2 \ge \frac{C_\gamma^c}{2} v'(S) \right] \le 2\Pr\left[\sum_{i=1}^{r} Z'_i > \frac{C_\gamma^c}{2} v'(S) \right] + \Pr\left[A_r^c \right] + \Pr\left[B_r^c \right] . \tag{13}$$

We now wish to apply Corollary 8 to the martingale difference $(Z'_i, \mathcal{F}_i)_{i=0}^r$. Thus, we have to bound $|Z'_i|$ as well as the conditional variances $\mathrm{Var}\left[Z'_i \mid \mathcal{F}_{i-1} \right]$. For the bound on Z'_i, observe that by the Cauchy-Schwarz inequality,

$$|Z'_i| = [A_i \cap B_i \cap C_i] \left| \sum_{j\in[m]} Y_i^{(j)} Y_{<i}^{(j)} \right| \le [A_i \cap B_i \cap C_i] \sqrt{\sum_{j\in[m]} \left(Y_i^{(j)}\right)^2} \sqrt{\sum_{j\in[m]} \left(Y_{<i}^{(j)}\right)^2} .$$

If A_i occurs we obtain

$$\sum_{j\in[m]} \left(Y_i^{(j)}\right)^2 \le C_\gamma^{c-1} v'(G_i) \le C_\gamma^{c-1} v'(S)^{1-1/c} v'_\infty(S)^{1/c} ,$$

by Lemma 13 and if A_i, B_i, and C_i all occur then

$$\sum_{j\in[m]} \left(Y_{<i}^{(j)}\right)^2 = \sum_{j\in[m]} \sum_{k<i} \left(Y_k^{(j)}\right)^2 + 2Z'_{<i} \le C_\gamma^{c-1} v'(G_{<i}) + 2C_\gamma^c v'(G_{<i}) \le 3C_\gamma^c v'(S) .$$

In conclusion

$$|Z'_i| \le C_\gamma^{c-1/2} \sqrt{3} v'(S)^{1-1/(2c)} v'_\infty(S)^{1/(2c)} .$$

For the bound on the conditional variance note that if B_i occurs then $\mathrm{Var}\left[Z_i \mid \mathcal{F}_{i-1}\right] \le m^{-1/2} v'(G_k) v'(G_{<k})$ and thus,

$$\mathrm{Var}\left[Z_i' \mid \mathcal{F}_{i-1}\right] = [A_i][B_i][C_i]\mathrm{Var}\left[Z_i \mid \mathcal{F}_{i-1}\right] \le m^{-1/2} v'(G_k) v'(G_{<k}) \ .$$

It follows that $\sum_{i=1}^{r} \mathrm{Var}\left[Z_i' \mid \mathcal{F}_{i-1}\right] \le m^{-1/2} v'(S)^2$. Letting

$$\sigma^2 = m^{-1/2} v'(S)^2$$

and

$$M = C_\gamma^{c-1/2}\sqrt{3} v'(S)^{1-1/(2c)} v'_\infty(S)^{1/(2c)}$$

in Corollary 8 we thus obtain

$$\Pr\left[\sum_{i=1}^{r} Z_i' > \frac{C_\gamma^c}{2} v'(S)\right] \le \exp\left(- \frac{v'(S)^{1/c}}{3C_\gamma^{2c-1} \cdot \sqrt{m} \cdot v'_\infty(S)^{1/c}} \mathcal{C}\left(\frac{(C\gamma)^{2c-1/2} \cdot \sqrt{3} \cdot \sqrt{m} \cdot v'_\infty(S)^{1/2c}}{2v'(S)^{1/2c}} \right)\right) \ .$$

Applying Lemma 10 first with $b = \left(\frac{v'_\infty(S)}{v'(S)}\right)^{1/(2c)} \le 1$ and then with $b = \sqrt{m} > 1$ yields

$$\frac{v'(S)^{1/c}}{3C_\gamma^{2c-1}\sqrt{m} v'_\infty(S)^{1/c}} \mathcal{C}\left(\frac{C_\gamma^{2c-1/2}\sqrt{3}\sqrt{m} v'_\infty(S)^{1/2c}}{2v'(S)^{1/2c}}\right) \ge \frac{1}{3C_\gamma^{2c-1}} \mathcal{C}\left(\frac{C_\gamma^{2c-1/2}\sqrt{3}\sqrt{m}}{2}\right) \ .$$

We then use Lemma 9 to get

$$\frac{1}{3C_\gamma^{2c-1}} \mathcal{C}\left(\frac{C_\gamma^{2c-1/2}\sqrt{3}\sqrt{m}}{2}\right) \ge \frac{\sqrt{C_\gamma}}{\sqrt{3}\cdot 4} \log\left(1 + \frac{(C\gamma)^{2c-1/2}\sqrt{3}\sqrt{m}}{2}\right) \ge \frac{\sqrt{C_\gamma}}{\sqrt{3}\cdot 8}\log(m) = \gamma \log(m) \ ,$$

where we have used that $C_\gamma = 3 \cdot 8 \cdot \gamma^2$ and $\gamma \ge 1$. Combining this we get that

$$\Pr\left[\sum_{i=1}^{r} Z_i' > \frac{C_\gamma^c}{2} v'(S)\right] \le m^{-\gamma} \ . \tag{14}$$

It thus suffices to bound the probabilities $\Pr[A_{r-1}^c]$ and $\Pr[B_{r-1}^c]$. For A_{r-1}^c, if $c = 1$ the discussion from the bound on V_1 proves that A_{r-1}^c never occurs. If $c > 1$, the inductive hypothesis on the groups G_i and a union bound yields

$$\Pr\left[A_{r-1}^c\right] = O_{\gamma,\epsilon,c}\left(\sum_{i=1}^{r} |G_i| m^{-\gamma}\right) = O(|S| m^{-\gamma}) \ . \tag{15}$$

For B_{r-1}^c, we can for each $i \in \{1, \ldots, r\}$ write

$$\begin{aligned}
\mathrm{Var}\left[Z_i \mid \mathcal{F}_{i-1}\right] &= \mathbb{E}\left[\left(\sum_{j\in[m]} X_i^{(j\oplus h(\alpha_i))} Y_{<i}^{(j)}\right)^2 \,\middle|\, \mathcal{F}_{i-1}\right] \\
&= \frac{1}{m}\sum_{k\in[m]} \left(\sum_{j\in[m]} X_i^{(j\oplus k)} Y_{<i}^{(j)}\right)^2 \\
&= \frac{1}{m}\sum_{k\in[m]} \sum_{(j_1,j_2)\in[m]^2} Y_i^{(j_1\oplus k)} Y_i^{(j_2\oplus k)} Y_{<i}^{(j_1)} Y_{<i}^{(j_2)} \\
&= \frac{1}{m} \sum_{\substack{(j_1,j_2,j_3,j_4)\in[m]^4 \\ j_1\oplus j_2\oplus j_3\oplus j_4 = 0}} Y_i^{(j_1)} Y_i^{(j_2)} Y_{<i}^{(j_3)} Y_{<i}^{(j_4)}
\end{aligned}$$

Call this quantity T_i. It follows from Lemma 15 and Markov's inequality that

$$\Pr\left[T_i \geq m^{-1/2} v'(G_k) v'(G_{<k})\right] \leq \frac{\mathbb{E}\left[T_i^{2\gamma/(1-4\varepsilon)}\right]}{m^{\gamma/(1-4\varepsilon)}\left(v'(G_k)v'(G_{<k})\right)^{2\gamma/(1-4\varepsilon)}} \leq O_{\gamma,\varepsilon,c}(m^{-\gamma}).$$

Thus, $\Pr[B_{r-1}^c] = O(|S| m^{-\gamma})$ by a union bound.

Combining equations (11)-(15) we conclude that indeed $\Pr\left[V \geq C_\gamma^c v'(S)\right] = O_{\gamma,\varepsilon,c}(|S| m^{-\gamma})$. □

4.3 Establishing the Concentration Bound

With the results of the previous subsection at hand, we proceed to prove the first part of Theorem 4. We show that for a value function of support bounded in size by m^ε for some $\varepsilon < 1/4$, simple tabulation supports Chernoff-style bounds with added error probability inversely polynomial in m. For convenience, we restate the first part of Theorem 4 as Theorem 17. The statement is equivalent to Theorem 4 but for precision, we have chosen to write out the statement more explicitly.

Theorem 17. *Let $h\colon \Sigma^c \to [m]$ be a simple tabulation hash function and $S \subseteq \Sigma^c$ be a set of keys. Let $v\colon \Sigma^c \times [m] \to [-1,1]$ be a value such that the set $Q = \{i \in [m] \mid \exists x \in \Sigma^c : v(x,i) \neq 0\}$ satisfies $|Q| \leq m^\varepsilon$, where $\varepsilon < \frac{1}{4}$ is a constant. Define the random variable $W = \sum_{x\in S} v(x, h(x))$ and write $\mu = \mathbb{E}[W]$ and $\sigma^2 = \mathrm{Var}[W]$. Then for any constant $\gamma \geq 1$,*

$$\Pr\left[|W-\mu| \geq C_{\gamma,c} t\right] \leq 2\exp\left(-\sigma^2 \mathcal{C}\left(\frac{t}{\sigma^2}\right)\right) + O_{\gamma,\varepsilon,c}\left(|S|\, m^{-\gamma}\right) ,$$

where $C_{\gamma,c} = \left(1+\frac{1}{\gamma}\right)^{3^{\frac{c(c-1)}{2}}} (Cc\gamma)^{3c}$ for some large enough universal constant C.

Proof. First, akin to the proof of Theorem 16, we may write

$$V = W - \mu = \sum_{x\in S}\sum_{k\in Q} v(x,k)\left([h(x) = j \oplus k] - \frac{1}{m}\right),$$

and note that

$$\mathrm{Var}[V] = \mathrm{Var}[W] = \sum_{x\in S}\left(\sum_{k\in Q}\frac{1}{m}v(x,k)^2 - \left(\sum_{k\in Q}\frac{1}{m}v(x,k)\right)^2\right).$$

We proceed by induction on c. For $c = 1$ we have full randomness and it follows immediately from Corollary 8 that

$$\Pr\left[|V| \geq t\right] \leq 2\exp\left(-\sigma^2\mathcal{C}\left(\frac{t}{\sigma^2}\right)\right) .$$

Now assume that $c > 1$ and inductively that the result holds for smaller values of c. We define $v'(x) = \sum_{k\in Q} v(x,k)^2$ and for $X \subseteq \Sigma^c$ we let $v'(X) = \sum_{x\in X} v'(x)$ and define $v'_\infty(X) = \max_{x\in X} v'(x)$. Now, applying Lemma 13 with respect to $w = v'$ we get position characters $\alpha_1, \ldots, \alpha_r$ with corresponding groups $G_1, \ldots, G_r$, such that $\cup_{i=1}^r G_i = S$ and for every $i \in \{1,\ldots,r\}$, $v'(G_i) \leq v'(S)^{1-1/c} v'_\infty(S)^{1/c}$. For a bin $j \in [m]$ and an $i \in \{1,\ldots,r\}$ we again define

$$X_i^{(j)} = \sum_{x\in G_i}\sum_{k\in Q} v(x,k)\cdot\left([h(x\setminus\alpha_i) = j\oplus k] - \frac{1}{m}\right) , \qquad Y_i = X_i^{(h(\alpha_i))} .$$

Note that $\sum_{i=1}^r Y_i = V$. For $i \in \{1,\ldots,r\}$ we define the σ-algebra $\mathcal{F}_i = \sigma((h(\alpha_j))_{j=1}^i)$. We furthermore define $Y_0 = 0$ and $\mathcal{F}_0 = \{\emptyset, \Omega\}$. As Y_i is $\mathcal{F}_i$-measurable for $i \in [r+1]$ and $\mathbb{E}[Y_i \mid \mathcal{F}_{i-1}] = 0$ for $i \in \{1,\ldots,r\}$, $(Y_i, \mathcal{F}_i)_{i=0}^r$ is a martingale difference. Furthermore, for $i \in \{1,\ldots,r\}$,

$$\mathrm{Var}[Y_i \mid F_{i-1}] = \frac{1}{m}\sum_{j\in[m]}\left(X_i^{(j)}\right)^2 .$$

According to Theorem 16 there exists a constant $K = 3 \cdot 2^6 \cdot \gamma^2$ such that

$$\Pr\left[\sum_{j\in[m]} \left(X_i^{(j)}\right)^2 > K^{c-1} v'(G_i)\right] \le O_{\gamma,\varepsilon,c}\left(|G_i|\, m^{\gamma}\right) \ . \tag{16}$$

For $i \in \{1,\dots,r\}$ we define the events

$$A_i = \bigcap_{k\le i}\left(\sum_{j\in[m]} (X_k^{(j)})^2 \le K^{c-1} v'(G_k)\right) \ ,$$
$$B_i = \left(\max_{\substack{k\le i\\ j\in[m]}} |X_k^{(j)}| \le C_{\gamma+1,c-1} M\right) \ ,$$

for some M to be specified later. We define $Z_i = [A_i \cap B_i] Y_i$ for $i \in \{1,\dots,r\}$ and $Z_0 = 0$. As both $A_i, B_i \in \mathcal{F}_{i-1}$ we have that $\mathbb{E}[Z_i \mid \mathcal{F}_{i-1}] = [A_i \cap B_i]\mathbb{E}[Y_i \mid \mathcal{F}_{i-1}] = 0$ for $\{1,\dots,r\}$ so $(Z_i, \mathcal{F}_i)_{i=0}^r$ is a martingale difference. By definition of A_i and B_i it moreover holds for $i \in \{1,\dots,r\}$ that

$$|Z_i| \le C_{\gamma+1,c-1} M \quad \text{and} \quad \operatorname{Var}[Z_i \mid \mathcal{F}_{i-1}] \le \frac{K^{c-1} v'(G_i)}{m} \ .$$

Setting $\sigma_0^2 = \frac{K^{c-1} v'(S)}{m}$ and applying Corollary 8 we obtain

$$\Pr\left[\left|\sum_{i=1}^r Z_i\right| \ge t\right] \le 2\exp\left(-\frac{\sigma_0^2}{C_{\gamma+1,c-1}^2 M^2}\mathcal{C}\left(\frac{t C_{\gamma+1,c-1} M}{\sigma_0^2}\right)\right) \ . \tag{17}$$

If A_{r-1} and B_{r-1} both occur then $\sum_{i=1}^r Z_i = \sum_{i=1}^r Y_i$ so it must hold that

$$\Pr[|V| \ge t] \le \Pr\left[\left|\sum_{i=1}^r Z_i\right| \ge t\right] + \Pr\left[A_{r-1}^c\right] + \Pr\left[B_{r-1}^c\right] \ .$$

We may assume that $m > 1$, i.e., the number of bins exceeds one, and then by the Cauchy-Schwarz inequality,

$$\begin{aligned}\sigma^2 = \sum_{x\in S}\left(\sum_{k\in Q}\frac{1}{m} v(x,k)^2 - \left(\sum_{k\in Q}\frac{1}{m} v(x,k)\right)^2\right) &\ge \sum_{x\in S}\left(\sum_{k\in Q}\frac{1}{m} v(x,k)^2 - \frac{1}{m^{2(1-\varepsilon)}}\sum_{k\in Q}\frac{1}{m^{\varepsilon}} v(x,k)^2\right)\\ &= \frac{v'(S)}{m}\left(1 - \frac{1}{m^{1-\varepsilon}}\right)\\ &\ge \frac{v'(S)}{3m}\\ &\ge \frac{\sigma_0^2}{3K^{c-1}}\end{aligned}$$

so using (17) we obtain

$$\Pr[|V| \ge C_{\gamma,c} t] \le 2\exp\left(-\frac{3K^{c-1}\sigma^2}{C_{\gamma+1,c-1}^2 M^2}\mathcal{C}\left(\frac{C_{\gamma,c}\cdot t \cdot C_{\gamma+1,c-1} M}{3K^{c-1}\sigma^2}\right)\right) + \Pr\left[A_{r-1}^c\right] + \Pr\left[B_{r-1}^c\right] \ . \tag{18}$$

By (16) and a union bound $\Pr\left[A_{r-1}^c\right] \le O(|S| m^{-\gamma})$. For bounding $\Pr\left[B_{r-1}^c\right]$ we use the induction hypothesis on the groups, concluding that for $i \in \{1,\dots,r\}$ and $j \in [m]$,

$$\Pr\left[\left|X_i^{(j)}\right| > C_{\gamma+1,c-1} M\right] \le 2\exp\left(-\sigma_i^2 \mathcal{C}\left(\frac{M}{\sigma_i^2}\right)\right) + O(|G_i| m^{-\gamma-1}) \ ,$$

where $\sigma_i^2 = \operatorname{Var}\left[Y_i^{(j)}\right] \le v'(G_i)/m$. By the initial assumption on the groups, this implies $\sigma_i^2 \le v'(S)^{1-1/c}v'_\infty(S)/m$ and we denote the latter quantity τ^2. Combining with Lemma 11 we obtain by a union bound that

$$\Pr\left[B_{r-1}^c\right] \le 2\,|S|\,m \exp\left(-\tau^2\mathcal{C}\left(\frac{M}{\tau^2}\right)\right) + O(|S| m^{-\gamma}) .$$

We fix M to be the unique real number with $\tau^2\mathcal{C}\left(\frac{M}{\tau^2}\right) = (\gamma+1)\log(m)$. With this choice of M, $\Pr\left[B_{r-1}^c\right] \le O(|S|\,m^{-\gamma})$, so by (18) it suffices to show that

$$\frac{3K^{c-1}\sigma^2}{C_{\gamma+1,c-1}^2 M^2}\mathcal{C}\left(\frac{C_{\gamma,c}\cdot t\cdot C_{\gamma+1,c-1}M}{3K^{c-1}\sigma^2}\right) \ge \min\left\{\sigma^2\mathcal{C}\left(\frac{t}{\sigma^2}\right), \gamma\log(m)\right\} . \tag{19}$$

First since $\frac{C_{\gamma,c}C_{\gamma+1,c-1}}{3K^{c-1}} \ge 1$ Lemma 10 give us that

$$\frac{3\left(K\gamma\right)^{c-1}\sigma^2}{C_{\gamma+1,c-1}^2 M^2}\mathcal{C}\left(\frac{C_{\gamma,c}\cdot t\cdot C_{\gamma+1,c-1}M}{3\left(K\gamma\right)^{c-1}\sigma^2}\right) \ge \frac{C_{\gamma,c}}{C_{\gamma+1,c-1}}\frac{\sigma^2}{M^2}\mathcal{C}\left(\frac{tM}{\sigma^2}\right) .$$

Now by definition of $C_{\gamma,c}$ and $C_{\gamma+1,c-1}$ we get that

$$\frac{C_{\gamma,c}}{C_{\gamma+1,c-1}} = \frac{\left(1+\frac{1}{\gamma}\right)^{3^{\frac{c(c-1)}{2}}}\left(Cc\gamma\right)^{3c}}{\left(1+\frac{1}{\gamma+1}\right)^{3^{\frac{(c-1)(c-2)}{2}}}\left(Cc(\gamma+1)\right)^{3(c-1)}} \ge (Cc\gamma)^3 .$$

So we have reduced the problem to showing that

$$(Cc\gamma)^3\frac{\sigma^2}{M^2}\mathcal{C}\left(\frac{tM}{\sigma^2}\right) \ge \min\left\{\sigma^2\mathcal{C}\left(\frac{t}{\sigma^2}\right), \gamma\log(m)\right\} .$$

For that we have to check a couple of cases.

Case 1. $\frac{tM}{\sigma^2} \le 1$**:** Using Lemma 9 twice and the fact that $(Cc\gamma)^3 \ge \frac{3}{2}$, we get that

$$(Cc\gamma)^3\sigma^2\mathcal{C}\left(\frac{tM}{\sigma^2}\right) \ge \frac{(Cc\gamma)^3}{3}\frac{t^2}{\sigma^2} \ge \sigma^2\mathcal{C}\left(\frac{t}{\sigma^2}\right) .$$

Case 2. $v'(S) \le m^{(1-\varepsilon/c)\left(1+\frac{1}{2c-1}\right)}$**:** We then get that

$$\tau^2 \le \frac{v'(S)^{1-1/c}v'_\infty(S)}{m} \le \frac{m^{(1-\varepsilon/c)\left(1-\frac{1}{2c-1}\right)}}{m^{1-\varepsilon/c}} = m^{-\frac{1-\varepsilon/c}{2c-1}} .$$

Now we note that $M \le 12\gamma c$ since

$$\tau^2\mathcal{C}\left(\frac{12\gamma c}{\tau^2}\right) \ge 12\gamma c\log\left(1+\frac{12\gamma c}{\tau^2}\right)/2 \ge 6\gamma c\log(1/\tau^2) \ge 6\gamma c\frac{1-\varepsilon/c}{2c-1}\log(m) \ge (\gamma+1)\log(m) ,$$

where we have used that $\varepsilon \le \frac{1}{4}$ and $\gamma \ge 1$.

We then get that

$$(Cc\gamma)^3\frac{\sigma^2}{M^2}\mathcal{C}\left(\frac{tM}{\sigma^2}\right) \ge \frac{(Cc\gamma)^3}{M}\sigma^2\mathcal{C}\left(\frac{t}{\sigma^2}\right) \ge \frac{(Cc\gamma)^3}{12c\gamma}\sigma^2\mathcal{C}\left(\frac{t}{\sigma^2}\right) \ge \sigma^2\mathcal{C}\left(\frac{t}{\sigma^2}\right) .$$

Case 3. $\frac{tM}{\sigma^2} > 1$ **and** $v'(S) > m^{(1-\varepsilon/c)\left(1+\frac{1}{2c-1}\right)}$**:** We see that $M \le \max\left\{6\gamma\log(m), \sqrt{6\gamma\log(m)}\tau\right\}$ since

$$\tau^2\mathcal{C}\left(\frac{\max\left\{6\gamma\log(m), \sqrt{6\gamma\log(m)}\tau\right\}}{\tau^2}\right) \ge \max\left\{\frac{6\gamma\log(m)}{3}, \frac{6\gamma\log(m)}{3}\right\} \ge (\gamma+1)\log(m)\ ,$$

where we have used Lemma 9 and that $\gamma \ge 1$. Now we have that $\sigma^2 \ge \frac{v'(S)}{3m} > m^{\frac{1-2\varepsilon}{2c-1}}/3$ and $\tau^2 \le \frac{v'(S)^{1-1/c}v'_\infty(S)^{1/c}}{m}$. Combining this we get that

$$\begin{aligned}
\frac{\sigma^2}{M^2} &\ge \min\left\{\frac{\sigma^2}{36\gamma^2\log(m)^2}, \frac{\sigma^2}{6\gamma\log(m)\tau^2}\right\} \\
&\ge \min\left\{\frac{m^{\frac{1-2\varepsilon}{2c-1}}}{108\gamma^2\log(m)^2}, \left(\frac{v'(S)}{v'_\infty(S)}\right)^{1/c}\cdot\frac{1}{3\cdot 6\gamma\log(m)}\right\} \\
&\ge \min\left\{\frac{m^{\frac{1-2\varepsilon}{2c-1}}}{108\gamma^2\log(m)^2}, m^{\frac{1-2\varepsilon}{2c}}\cdot\frac{1}{18\gamma\log(m)}\right\} \\
&\ge \frac{m^{\frac{1}{4c}}}{108\gamma^2\log(m)^2} \\
&\ge \frac{\log(m)}{108\cdot 4^3c^3\gamma^2}\ ,
\end{aligned}$$

where have used that $\varepsilon < \frac{1}{4}$ and that $\frac{m^{\frac{1}{4c}}}{\log(m)^2} \ge \frac{\log(m)}{4^3c^3}$. Now we get that

$$(Cc\gamma)^3\frac{\sigma^2}{M^2}\mathcal{C}\left(\frac{tM}{\sigma^2}\right) \ge (Cc\gamma)^3\frac{\log(m)}{108\cdot 4^3c^3\gamma^2}/3 \ge \gamma\log(m)\ .$$

□

5 General Value Functions – Arbitrary Bins

The goal of this section is to prove Theorem 5, the second step towards Theorem 2. Again, we postpone the argument that our concentration bounds are query invariant to Section 7. Recall that Theorem 5 is concerned with a hash function of the form $h = \tau \circ g$, where $g : \Sigma^c \to [m]$ is a simple tabulation hash function and τ is a uniformly random permutation. Our goal is to prove that for any value function $v\colon \Sigma^c \times [m] \to [-1, 1]$, the sum $\sum_{x\in\Sigma^c} v(x, h(x))$ is strongly concentrated with high probability in m. This result follows by combining the distributional properties of g with the randomness of τ.

We start out by proving a lemma. The lemma describes properties we need g to possess for the final composition with τ to yield Chernoff-style concentration.

Lemma 18. *Let $m \ge 2$ be an integer and $C, T \in \mathbb{R}^+$ positive reals. Furthermore, let $\mathcal{V}\colon [m]\times[m] \to \mathbb{R}$ be a value function satisfying $\sum_{i\in[m]} \mathcal{V}(i,j) = 0$ for every $j \in [m]$ and such that*

$$\max_{i,j\in[m]} |\mathcal{V}(i,j)| \le M := \max\left\{C, \frac{\sigma^2}{T}\right\}\ ,$$

where $\sigma^2 = \frac{1}{m}\sum_{i\in[m]}\sum_{j\in[m]} \mathcal{V}(i,j)^2$. If $\tau\colon [m] \to [m]$ is a uniformly random permutation, then the random variable $Z = \sum_{i\in[m]} \mathcal{V}(\tau(i), i)$ satisfies

$$\Pr\left[|Z| \ge Dt\right] \le 4\left(\exp\left(-\sigma^2\mathcal{C}\left(\frac{t}{\sigma^2}\right)\right) + \exp\left(-\frac{T^2}{2\sigma^2}\right)\right)\ ,$$

where $D = \max\{8C, 12\}$ is a universal constant depending on C.

Proof. We define $Y_1 = \sum_{i=0}^{\lceil m/2 \rceil - 1} \mathcal{V}(\tau(i), i)$ and $Y_2 = \sum_{i=\lceil m/2 \rceil}^{m-1} \mathcal{V}(\tau(i), i)$. Since $Z = Y_1 + Y_2$ it follows that if $Z > Dt$ then there exists $i \in \{1, 2\}$ such that $Y_i \geq \frac{D}{2} t$. It suffices to show that

$$\Pr\left[Y_i \geq \frac{D}{2} t\right] \leq \exp\left(-\sigma^2 \mathcal{C}\left(\frac{t}{\sigma^2}\right)\right) + \exp\left(-\sigma^2 \mathcal{C}\left(\frac{T}{\sigma^2}\right)\right) , \tag{20}$$

for $i \in \{1, 2\}$. A union bound over i then yields a bound on $\Pr[Z \geq Dt]$. Since we may instead consider the value function $-\mathcal{V}$, the same argument yields a bound on $\Pr[Z \leq -Dt]$, which concludes the proof.

Thus, we shall prove (20) for Y_1 – the proof is completely analogous for Y_2. Define the filtration $(\mathcal{F}_i)_{i=0}^{\lceil m/2 \rceil}$ by $\mathcal{F}_i = \sigma\left((\tau(j))_{j \in [i]}\right)$ and let $X_i = \mathbb{E}\left[Y_1 \mid \mathcal{F}_i\right]$ such that $(X_i, \mathcal{F}_i)_{i=0}^{\lceil m/2 \rceil}$ is a martingale, $X_0 = \mathbb{E}\left[Y_1\right]$, and $X_{\lceil m/2 \rceil} = Y_1$. Towards applying Corollary 8, we bound $|X_i - X_{i-1}|$ and $\sum_{i=1}^{\lceil m/2 \rceil} \operatorname{Var}\left[X_i - X_{i-1} \mid \mathcal{F}_{i-1}\right]$.

First, we bound $|X_i - X_{i-1}|$. We start by writing

$$\begin{aligned} X_i - X_{i-1} &= \mathbb{E}\left[Y_1 \mid \mathcal{F}_i\right] - \mathbb{E}\left[Y_1 \mid \mathcal{F}_{i-1}\right] \\ &= \mathcal{V}(\tau(i-1), i-1) - \mathbb{E}\left[\mathcal{V}(\tau(i-1), i-1) \mid \mathcal{F}_{i-1}\right] + \sum_{k=i}^{\lceil m/2 \rceil - 1} \left(\mathbb{E}\left[\mathcal{V}(\tau(k), k) \mid \mathcal{F}_i\right] - \mathbb{E}\left[\mathcal{V}(\tau(k), k) \mid \mathcal{F}_{i-1}\right]\right) . \end{aligned}$$

Now, note that for $k \geq i$,

$$\mathbb{E}\left[\mathcal{V}(\tau(k), k) \mid \mathcal{F}_i\right] = -\frac{1}{m-i} \sum_{j=0}^{i-1} \mathcal{V}(\tau(j), k) ,$$

since $\sum_{\ell \in [m]} \mathcal{V}(\ell, k) = 0$, and furthermore,

$$\mathbb{E}\left[\mathcal{V}(\tau(k), k) \mid \mathcal{F}_{i-1}\right] = -\frac{1}{m-i} \left(\mathbb{E}\left[\mathcal{V}(\tau(i-1), k) \mid \mathcal{F}_{i-1}\right] + \sum_{j=0}^{i-2} \mathcal{V}(\tau(j), k)\right) .$$

Hence, it follows that

$$\begin{aligned} X_i - X_{i-1} = {} & \mathcal{V}(\tau(i-1), i-1) - \mathbb{E}\left[\mathcal{V}(\tau(i-1), i-1) \mid \mathcal{F}_{i-1}\right] \\ & - \frac{1}{m-i} \sum_{k=i}^{\lceil m/2 \rceil - 1} \left(\mathcal{V}(\tau(i-1), k) - \mathbb{E}\left[\mathcal{V}(\tau(i-1), k) \mid \mathcal{F}_{i-1}\right]\right) . \end{aligned}$$

Since $|\mathcal{V}(i, j)| \leq M$ for all $i, j \in [m]$, it follows that $|X_i - X_{i-1}| \leq 4M$.

Second, we bound $\operatorname{Var}\left[X_i - X_{i-1} \mid \mathcal{F}_{i-1}\right]$. To this end, observe that

$$\begin{aligned} \operatorname{Var}\left[X_i - X_{i-1} \mid \mathcal{F}_{i-1}\right] &= \operatorname{Var}\left[\mathcal{V}(\tau(i-1), i-1) - \frac{1}{m-i} \sum_{k=i}^{\lceil m/2 \rceil - 1} \mathcal{V}(\tau(i-1), k) \,\middle|\, \mathcal{F}_{i-1}\right] \\ &\leq 2\left(\operatorname{Var}\left[\mathcal{V}(\tau(i-1), i-1) \mid \mathcal{F}_{i-1}\right] + \frac{1}{m-i} \sum_{k=i}^{\lceil m/2 \rceil - 1} \operatorname{Var}\left[\mathcal{V}(\tau(i-1), k) \mid \mathcal{F}_{i-1}\right]\right) , \end{aligned}$$

where the inequality follows from the fact that $2\operatorname{Cov}\left(A, B \mid \mathcal{H}\right) \leq \operatorname{Var}\left[A \mid \mathcal{H}\right] + \operatorname{Var}\left[B \mid \mathcal{H}\right]$, for any random variables A and B and any sigma algebra $\mathcal{H}$. For any $k \in [m]$,

$$\begin{aligned}\mathrm{Var}\left[\mathcal{V}(\tau(i-1),k) \mid \mathcal{F}_{i-1}\right] &\le \mathbb{E}\left[\mathcal{V}(\tau(i-1),k)^2 \mid \mathcal{F}_{i-1}\right] \\ &= \frac{1}{m-i+1}\sum_{j\in[m]\setminus\tau([i-1])}\mathcal{V}(j,k)^2 \\ &\le \frac{1}{m-i+1}\sum_{j\in[m]}\mathcal{V}(j,k)^2 \\ &\le \frac{2}{m}\sum_{j\in[m]}\mathcal{V}(j,k)^2 ,\end{aligned}$$

where the last inequality follows from the fact that $i \le \lceil m/2 \rceil$. Hence,

$$\begin{aligned}\mathrm{Var}\left[X_i - X_{i-1} \mid \mathcal{F}_{i-1}\right] &\le 2\left(\mathrm{Var}\left[\mathcal{V}(\tau(i-1),i-1) \mid \mathcal{F}_{i-1}\right] + \frac{1}{m-i}\sum_{k=i}^{\lceil m/2\rceil-1}\mathrm{Var}\left[\mathcal{V}(\tau(i-1),k) \mid \mathcal{F}_{i-1}\right]\right) \\ &\le \frac{4}{m}\sum_{j\in[m]}\mathcal{V}(j,i)^2 + \frac{2}{m-i}\cdot\frac{2}{m}\sum_{k=i}^{\lceil m/2\rceil-1}\sum_{j\in[m]}\mathcal{V}(j,k)^2 \\ &\le \frac{4}{m}\sum_{j\in[m]}\mathcal{V}(j,i)^2 + \frac{16}{m^2}\sum_{k\in[m]}\sum_{j\in[m]}\mathcal{V}(j,k)^2 ,\end{aligned}$$

again using that $i \le \lceil m/2 \rceil$. We now see that

$$\begin{aligned}\sum_{i=1}^{\lceil m/2\rceil}\mathrm{Var}\left[X_i - X_{i-1} \mid \mathcal{F}_{i-1}\right] &\le \sum_{i=1}^{\lceil m/2\rceil}\left(\frac{4}{m}\sum_{j\in[m]}\mathcal{V}(j,i)^2 + \frac{16}{m^2}\sum_{k\in[m]}\sum_{j\in[m]}\mathcal{V}(j,k)^2\right) \\ &\le \sum_{i\in[m]}\left(\frac{4}{m}\sum_{j\in[m]}\mathcal{V}(j,i)^2 + \frac{16}{m^2}\sum_{k\in[m]}\sum_{j\in[m]}\mathcal{V}(j,k)^2\right) \\ &\le 20\sigma^2 .\end{aligned}$$

The assumption on $\mathcal{V}$ implies that $\mathbb{E}\left[\mathcal{V}(\tau(i),i)\right] = 0$ for each $i \in [m]$, so also $\mathbb{E}\left[Y_1\right] = 0$. Applying Corollary 8 then yields,

$$\Pr\left[Y_1 \ge \frac{D}{2}t\right] \le \exp\left(-\frac{20\sigma^2}{(4M)^2}\mathcal{C}\left(\frac{(D/2)t\cdot 4M}{20\sigma^2}\right)\right) = \exp\left(-\frac{5\sigma^2}{4M^2}\mathcal{C}\left(\frac{DMt}{10\sigma^2}\right)\right).$$

The goal is now to show that

$$\frac{5\sigma^2}{4M^2}\mathcal{C}\left(\frac{DMt}{10\sigma^2}\right) \ge \min\left\{\sigma^2\mathcal{C}\left(\frac{t}{\sigma^2}\right), \frac{T^2}{2\sigma^2}\right\}. \tag{21}$$

Because if this is the case, then as desired

$$\Pr\left[Y_1 \ge \frac{D}{2}t\right] \le \exp\left(-\min\left\{\sigma^2\mathcal{C}\left(\frac{t}{\sigma^2}\right), \frac{T^2}{2\sigma^2}\right\}\right) \le \exp\left(-\sigma^2\mathcal{C}\left(\frac{t}{\sigma^2}\right)\right) + \exp\left(-\frac{T^2}{2\sigma^2}\right).$$

We check (21) by cases. This completes the proof.

Case 1. $M \le \frac{10}{D}$**:** In this case, $\frac{DM}{10} \le 1$. Thus, by Lemma 10,

$$\frac{5\sigma^2}{4M^2}\mathcal{C}\left(\frac{DMt}{10\sigma^2}\right) \ge \frac{D^2}{80}\sigma^2\mathcal{C}\left(\frac{t}{\sigma^2}\right) \ge \sigma^2\mathcal{C}\left(\frac{t}{\sigma^2}\right) ,$$

using that $D \ge 12 \ge \sqrt{80}$.

Case 2. $\frac{10}{D} \leq M \leq C$**:** In this case, $\frac{DM}{10} \geq 1$. Thus, by Lemma 10,

$$\frac{5\sigma^2}{4M^2}\mathcal{C}\left(\frac{DMt}{10\sigma^2}\right) \geq \frac{D}{8M}\sigma^2\mathcal{C}\left(\frac{t}{\sigma^2}\right) \geq \frac{D}{8C}\sigma^2\mathcal{C}\left(\frac{t}{\sigma^2}\right) \geq \sigma^2\mathcal{C}\left(\frac{t}{\sigma^2}\right) ,$$

using that $D \geq 8C$.

Case 3. $M \leq \frac{\sigma^2}{T}$**:** In this case, recall that $D \geq 12$ such that $\frac{D}{10} \geq 1$ and we may apply Lemma 10, yielding

$$\frac{5\sigma^2}{4M^2}\mathcal{C}\left(\frac{DMt}{10\sigma^2}\right) \geq \frac{5}{4}\frac{T^2}{\sigma^2}\mathcal{C}\left(\frac{D}{10}\frac{t}{T}\right) \geq \frac{D}{8}\frac{T^2}{\sigma^2}\mathcal{C}\left(\frac{t}{T}\right) .$$

By Lemma 9,

$$\mathcal{C}\left(\frac{t}{T}\right) \geq \mathcal{C}\left(\min\left\{\frac{t}{T}, 1\right\}\right) \geq \min\left\{\frac{t^2}{3T^2}, \frac{1}{3}\right\} .$$

So finally,

$$\frac{5\sigma^2}{4M^2}\mathcal{C}\left(\frac{DMt}{10\sigma^2}\right) \geq \min\left\{\frac{D}{24}\frac{t^2}{\sigma^2}, \frac{D}{24}\frac{T^2}{\sigma^2}\right\} \geq \min\left\{\frac{D}{12}\sigma^2\mathcal{C}\left(\frac{t}{\sigma^2}\right), \frac{D}{24}\frac{T^2}{\sigma^2}\right\} \geq \min\left\{\sigma^2\mathcal{C}\left(\frac{t}{\sigma^2}\right), \frac{T^2}{2\sigma^2}\right\} ,$$

where we have used Lemma 9 and the fact that $D \geq 12$.

$\square$

With this result in hand we are ready to prove Theorem 5. We restate it here in a more technically explicit version. For a more intuitive understanding, please refer back to the original statement. Note that we only require the hash function h of the theorem to be 2-independent, whereas Theorem 5 requires the hash function to be 3-independent. The difference lies in that the statement of Theorem 5 is slightly stronger, guaranteeing query invariance. Having deferred the treatment of query invariance until later, we only need 2-independence for now.

Theorem 19. *Let $\varepsilon \in (0, 1]$ and $m \geq 2$ be given. Let $h\colon A \to [m]$ be a 2-independent hash function satisfying the following. For every $\gamma > 0$ and every value function $\tilde{v}\colon A \times [m] \to [-1, 1]$ such that $Q = \{i \in [m] \mid \exists x \in A\colon \tilde{v}(x, i) \neq 0\}$ has size $|Q| \leq m^\varepsilon$, the random variables $W = \sum_{x\in A} \tilde{v}(x, h(x))$ and $W_j = \sum_{x\in A} \tilde{v}(x, h(x) \oplus j), j \in [m]$ with mean $\mu_W = \mathbb{E}[W] = \mathbb{E}[W_j]$ and variance $\sigma_W^2 = \operatorname{Var}[W]$ satisfy the inequalities*

$$\Pr[|W - \mu_W| \geq C \cdot t] \leq 2\exp\left(-\sigma_W^2\mathcal{C}\left(\frac{t}{\sigma_W^2}\right)\right) + O(|A|\, m^{-\gamma}), \tag{22}$$

$$\Pr\left[\sum_{j\in[m]} (W_j - \mu_W)^2 \geq D \cdot \sum_{x\in A}\sum_{k\in Q} \tilde{v}(x, k)^2\right] = O(|A|\, m^{-\gamma}), \tag{23}$$

for every $t > 0$, where C and D are universal constants depending on γ and ε.

Let $v\colon A \times [m] \to [-1, 1]$ be any value function, $\tau\colon [m] \to [m]$ a uniformly random permutation independent of h, and $\gamma > 0$. The random variable $U = \sum_{x\in A} v(x, \tau(h(x)))$ with expectation $\mu = \mathbb{E}[U]$ and variance $\sigma^2 = \operatorname{Var}[U]$ satisfies

$$\Pr[|U - \mu| \geq E \cdot t] \leq 6\exp\left(-\sigma^2\mathcal{C}\left(\frac{t}{\sigma^2}\right)\right) + O(|A|\, m^{-\gamma}) \tag{24}$$

for every $t > 0$, where E is a universal constant depending on γ and ε.

Proof. Define $v' : A \times [m] \to [-1,1]$ by letting $v'(x,i) = \frac{1}{2}\left(v(x,i) - \frac{1}{m}\sum_{j\in[m]} v(x,j)\right)$ and write $V = U - \mu$. Since $\sum_{i\in[m]}\left([\tau(h(x)) = i] - \frac{1}{m}\right) = 0$, we may write

$$\begin{aligned}
V =& \sum_{x\in A}\sum_{i\in[m]} v(x,i)[\tau(h(x)) = i] - \frac{1}{m}\sum_{x\in A}\sum_{i\in[m]} v(x,i) + \sum_{x\in A}\left(\left(\sum_{i\in[m]}\left([\tau(h(x)) = i] - \frac{1}{m}\right)\right)\cdot\left(\frac{1}{m}\sum_{j\in[m]} v(x,j)\right)\right)\\
=& \sum_{x\in A}\sum_{i\in[m]}\left(v(x,i) - \frac{1}{m}\sum_{j\in[m]} v(x,j)\right)\left([\tau(h(x)) = i] - \frac{1}{m}\right)\\
=& 2\sum_{x\in A}\sum_{i\in[m]} v'(x,i)\left([\tau(h(x)) = i] - \frac{1}{m}\right).
\end{aligned}$$

We write $V' = \sum_{x\in A}\sum_{i\in[m]} v'(x,i)\left([\tau(h(x)) = i] - \frac{1}{m}\right)$ such that $V = 2V'$. We note that by 2-independence

$$\sigma^2 = \sum_{x\in A}\mathrm{Var}\left[v(x,\tau(h(x)))\right] = \sum_{x\in A}\mathbb{E}\left[\left(v(x,\tau(h(x))) - \frac{1}{m}\sum_{j\in[m]} v(x,j)\right)^2\right] = \frac{4}{m}\sum_{x\in A}\sum_{i\in[m]} v'(x,i)^2.$$

Thus, we may write $\sigma'^2 = \mathrm{Var}\left[V'\right] = \frac{1}{m}\sum_{x\in A}\sum_{i\in[m]} v'(x,i)^2$. We proceed to show that for some constant E' depending on γ and ε,

$$\Pr\left[|V'| \ge E'\cdot t\right] \le 6\exp\left(-\sigma'^2 h\left(\frac{t}{\sigma'^2}\right)\right) + O(|A|m^{-\gamma}),$$

As $\sigma' \le \sigma$ and $V = 2V'$ the theorem then follows with $E = 2E'$ by applying Lemma 11.

For $i \in [m]$ we define $\sigma_i^2 = \frac{1}{m}\sum_{x\in A} v'(x,i)^2$, so that $\sum_{i\in[m]}\sigma_i^2 = \sigma'^2$. Assume without loss of generality that $\sigma_0^2 \ge \cdots \ge \sigma_{m-1}^2$. Now define $\mathcal{V} : [m]\times[m] \to \mathbb{R}$ by

$$\mathcal{V}(i,j) = \sum_{x\in A} v'(x,j)\left([h(x) = i] - \frac{1}{m}\right).$$

Note that for any $j \in [m]$, $\sum_{i\in[m]}\mathcal{V}(i,j) = 0$, regardless of the (random) choice of h. With this definition, $V' = \sum_{i\in[m]}\mathcal{V}(i,\tau(i)) = \sum_{j\in[m]}\mathcal{V}(\tau^{-1}(j),j)$. Now let

$$V_1 = \sum_{j\in[m^\varepsilon]}\mathcal{V}(\tau^{-1}(j),j) \quad\text{and}\quad V_2 = \sum_{j\in[m]\setminus[m^\varepsilon]}\mathcal{V}(\tau^{-1}(j),j),$$

and note that $V_1 + V_2 = V'$. Defining value functions $v_1', v_2' \colon A\times[m] \to [-1,1]$ by

$$v_1'(x,i) = \begin{cases} v'(x,i), & \text{if } i \in [m^\varepsilon]\\ 0, & \text{otherwise}\end{cases} \quad\text{and}\quad v_2'(x,i) = \begin{cases} v'(x,i), & \text{if } i \in [m]\setminus[m^\varepsilon]\\ 0, & \text{otherwise}\end{cases},$$

we observe that

$$V_1 = \sum_{x\in A} v_1'(x,\tau(h(x))) - \mathbb{E}\left[\sum_{x\in A} v_1'(x,\tau(h(x)))\right] \quad\text{and}\quad V_2 = \sum_{x\in A} v_2'(x,\tau(h(x))) - \mathbb{E}\left[\sum_{x\in A} v_2'(x,\tau(h(x)))\right]$$

Let $D \ge 1$ be such that Eq. (23) holds with added error probability $O(|A|m^{-\gamma-1})$ and let $M = \max\left\{C, \frac{\sigma'}{\sqrt{2D\gamma\log m}}\right\}$ for some large constant C to be fixed later. Define the two events

$$\mathcal{A} = \bigcap_{j\in[m]\setminus[m^\varepsilon]}\left(\max_{i\in m}|\mathcal{V}(i,j)| \le M\right) \quad\text{and}\quad \mathcal{B} = \bigcap_{j\in[m]}\left(\sum_{i\in[m]}\mathcal{V}(i,j)^2 < D\sigma_j^2 m\right).$$

By a union bound,

$$\Pr[|V'| \geq E't] \leq \Pr[|V_1| \geq E't/2] + \Pr[|[\mathcal{A}] \cdot [\mathcal{B}] \cdot V_2| \geq E't/2] + \Pr[\mathcal{A}^c] + \Pr[\mathcal{B}^c],$$

and we proceed to bound each of these terms individually.

First, we bound $\Pr[|V_1| \geq E't/2]$. To do so, suppose we fix the permutation $\tau = \tau_0$. With this conditioning and by 2-independence,

$$\begin{aligned}\mathrm{Var}\left[V_1 \mid \tau = \tau_0\right] =& \mathrm{Var}\left[\sum_{x \in A} [\tau_0(h(x)) \in [m^\varepsilon]] \cdot v'(x, \tau_0(h(x)))\right] \leq \sum_{x \in A} \mathbb{E}\left[[\tau_0(h(x)) \in [m^\varepsilon]] \cdot v'(x, \tau_0(h(x)))^2\right] \\ =& \frac{1}{m} \sum_{x \in A} \sum_{j \in [m^\varepsilon]} v'(x,j)^2 \leq \sigma'^2.\end{aligned}$$

Defining $\overline{v} : A \times [m] \to [-1, 1]$ by $\overline{v}(x, i) = v'_1(x, \tau_0(i))$ it holds that

$$V_1 = \sum_{x \in A} \sum_{i \in \tau_0^{-1}([m^\varepsilon])} \overline{v}(x,i) \left([h(x) = i] - \frac{1}{m}\right).$$

As $\overline{v}$ has support of size at most m^ε we can apply Eq. (22) to conclude that

$$\Pr[|V_1| \geq E't/2 \,|\, \tau = \tau_0] \leq 2 \exp\left(-\sigma'^2 \mathcal{C}\left(\frac{t}{\sigma'^2}\right)\right) + O(|A| m^{-\gamma}),$$

if the constant E' is large enough. Since this holds for any fixed τ_0, it also holds for the unconditioned probability.

We now bound $\Pr[|[\mathcal{A}] \cdot [\mathcal{B}] \cdot V_2| \geq E't/2]$. It suffices to condition on $h = h_0$ for some h_0 satisfying that $[\mathcal{A}] = [\mathcal{B}] = 1$ and make the bound over the randomness of τ. For this we may use Lemma 18. Indeed if $h = h_0$ for some h_0 such that $[\mathcal{A}] = [\mathcal{B}] = 1$, then $\sum_{i \in [m]} \sum_{j \in [m]} \frac{1}{m} \mathcal{V}(i,j)^2 \leq D\sigma'^2$. Here we used the conditioning on $\mathcal{A}$. Define the function $\mathcal{V}_0 : [m] \times [m] \to \mathbb{R}$ by $\mathcal{V}_0(i,j) = \mathcal{V}(i,j)$ when $j \in [m] \setminus [m^\varepsilon]$ and $\mathcal{V}_0(i,j) = 0$ otherwise. Then also $\sum_{i \in [m]} \sum_{j \in [m]} \frac{1}{m} \mathcal{V}_0(i,j)^2 \leq D\sigma'^2$ and further, for each $j \in [m]$, $\sum_{i \in [m]} \mathcal{V}_0(i,j) = 0$. Finally, the conditioning on $\mathcal{B}$ gives that $\max_{i,j \in [m]} \mathcal{V}_0(i,j) \leq M$. Note that $V_2 = \sum_{j \in [m]} \mathcal{V}_0(\tau^{-1}(j), j)$. Applying Lemma 18 to $\mathcal{V}_0$, noting that the bound obtained in that lemma is increasing in σ, we obtain that

$$\Pr\left[|V_2| \geq E't/2\right] \leq 4\left(\exp\left(-D\sigma'^2 \mathcal{C}\left(\frac{t}{D\sigma'^2}\right)\right) + \exp\left(-\gamma \log m\right)\right) = 4 \exp\left(-\Omega\left(\sigma'^2 \mathcal{C}\left(\frac{t}{\sigma'^2}\right)\right)\right) + O(m^{-\gamma}),$$

if E' is sufficiently large. From this it follows that,

$$\Pr[[\mathcal{A}] \cdot [\mathcal{B}] \cdot |V_2| \geq E't/2] \leq 4 \exp\left(-\sigma'^2 \mathcal{C}\left(\frac{t}{\sigma'^2}\right)\right) + O(m^{-\gamma}).$$

We finally need to bound $\Pr[\mathcal{A}^c]$ and $\Pr[\mathcal{B}^c]$. By the choice of D and a union bound we obtain that $\Pr[\mathcal{B}^c] = O(|A| m^{-\gamma})$, so for completing the proof it suffices to bound $\Pr[\mathcal{A}^c]$ which we proceed to do now. More specifically we bound $\Pr[|\mathcal{V}(i,j)| \geq M]$ for each $(i,j) \in [m] \times ([m] \setminus [m^\varepsilon])$, finishing with a union bound over the m^2 choices. So let $(i,j) \in [m] \times ([m] \setminus [m^\varepsilon])$ be fixed and define $\tilde{v} : A \times [m] \to [-1,1]$ by $\tilde{v}(x,i) = v'_2(x,j)$ and $\tilde{v}(x,k) = 0$ for $k \neq i$. Then $\tilde{v}$ has support $A \times \{i\}$,

$$\mathcal{V}(i,j) = \sum_{x \in A} \sum_{k \in [m]} \tilde{v}(x,k) \left([\tau(h(x)) = i] - \frac{1}{m}\right),$$

and $\mathrm{Var}\left[\mathcal{V}(i,j)\right] \leq \frac{1}{m} \sum_{x \in A} v'_2(x,j)^2 = \sigma_j^2 \leq \sigma'^2/m^\varepsilon$. The last inequality follows from our assumption that $\sigma_0^2 \geq \cdots \geq \sigma_{m-1}^2$ and $j \geq m^\varepsilon$.

By the assumption of Eq. (22) with γ replaced by $\gamma + 2$ it follows that

$$\Pr[|\mathcal{V}(i,j)| \geq M] \leq 2\exp\left(-\Omega\left(\sigma_j^2 \mathcal{C}\left(\frac{M}{\sigma_j^2}\right)\right)\right) + O(|A|m^{-\gamma-2}) \leq 2\exp\left(-D'\frac{\sigma'^2}{m^\varepsilon}\mathcal{C}\left(\frac{Mm^\varepsilon}{\sigma'^2}\right)\right) + O(|A|m^{-\gamma-2}),$$

for some constant D'. We finish the proof by showing that if the constant C from the definition of M is large enough, then

$$2\exp\left(-D'\frac{\sigma'^2}{m^\varepsilon}\mathcal{C}\left(\frac{Mm^\varepsilon}{\sigma'^2}\right)\right) = O(m^{-\gamma-2}).$$

For this it suffices to show that if C is large enough and m is greater than some constant, then

$$\frac{\sigma'^2}{m^\varepsilon}\mathcal{C}\left(\frac{Mm^\varepsilon}{\sigma'^2}\right) \geq \frac{(\gamma+2)\log m}{D'}.$$

Suppose first that $\sigma'^2 \leq m^{\varepsilon/2}$. In that case we use Lemma 9 to conclude that

$$\frac{\sigma'^2}{m^\varepsilon}\mathcal{C}\left(\frac{Mm^\varepsilon}{\sigma'^2}\right) \geq \frac{M}{2}\log\left(\frac{Mm^\varepsilon}{\sigma'^2}+1\right) \geq \frac{C}{2}\log\left(Cm^{\varepsilon/2}+1\right) \geq \frac{C\varepsilon}{4}\log m,$$

so if $C \geq 4\frac{\gamma+2}{D'\varepsilon}$ this is at least $\frac{(\gamma+2)\log m}{D'}$. Now suppose $m^{\varepsilon/2} < \sigma'^2 \leq m^{2\varepsilon}/(2D\gamma\log m)$. In that case we recall that $M = \max\left\{C, \frac{\sigma'}{\sqrt{2D\gamma\log m}}\right\}$ and use the bound

$$\frac{\sigma'^2}{m^\varepsilon}\mathcal{C}\left(\frac{Mm^\varepsilon}{\sigma'^2}\right) \geq \frac{M}{2}\log\left(\frac{Mm^\varepsilon}{\sigma'^2}+1\right) \geq \frac{\sigma'}{\sqrt{8D\gamma\log m}}\log\left(\frac{m^\varepsilon}{\sigma'\sqrt{2D\gamma\log m}}+1\right) = \Omega\left(\frac{m^{\varepsilon/4}}{\sqrt{\log m}}\right).$$

If m is larger than some constant, this is certainly at least $\frac{(\gamma+2)\log m}{D'}$. Finally suppose that $\sigma'^2 > m^{2\varepsilon}/(2D\gamma\log m)$. Using the inequality $\log(1+x) \geq \frac{x}{2}$ holding for $0 \leq x \leq 1$ we find that

$$\frac{\sigma'^2}{m^\varepsilon}\mathcal{C}\left(\frac{Mm^\varepsilon}{\sigma'^2}\right) \geq \frac{\sigma'}{\sqrt{8D\gamma\log m}}\log\left(\frac{m^\varepsilon}{\sigma'\sqrt{2D\gamma\log m}}+1\right) \geq \frac{m^\varepsilon}{8D\gamma\log m}.$$

Again it holds that if m is greater than some constant, this is at least $\frac{(\gamma+2)\log m}{D'}$. It follows that if C is large enough, then $\Pr[|\mathcal{V}(i,j)| \geq M] = O(|A|m^{-\gamma-2})$. Union bounding over $(i,j) \in [m] \times ([m] \setminus [m^\varepsilon])$ we find that $\Pr[\mathcal{A}^c] = O(|A|m^{-\gamma})$. Combining the bounds we find that

$$\Pr[|V'| \geq E't] \leq 6\exp\left(-\sigma'^2 h\left(\frac{t}{\sigma'^2}\right)\right) + O(|A|m^{-\gamma}),$$

which completes the proof.

□

6 Extending the Hash Range

This section is dedicated to proving Theorem 6, which we will restate shortly. Again, we will postpone the argument that our concentration bounds are query invariant to Section 7. First, we prove the following technical lemma.

Lemma 20. *Let $\sigma^2 > 0$ and $t > 0$. Writing $s = \max\left\{\sigma^2, \sqrt{t\sigma^2}\right\}$,*

$$s \cdot \mathcal{C}\left(\frac{t}{s}\right) \geq \sigma^2 \mathcal{C}\left(\frac{t}{\sigma^2}\right)/4\,.$$

Proof. For $t \le \sigma^2$ the inequality is trivial, so suppose $t > \sigma^2$. We note that for $x \ge 0$, $1 + \sqrt{x} \ge \sqrt{1+x}$, such that $\lg(1+\sqrt{x}) \ge \lg(1+x)/2$ for every $x \ge 0$. Using this fact in between two applications of Lemma 9, we find that

$$\sqrt{t\sigma^2}\mathcal{C}\left(\frac{t}{\sqrt{t\sigma^2}}\right) \ge t\lg\left(1+\sqrt{\frac{t}{\sigma^2}}\right)/2 \ge t\lg\left(1+\frac{t}{\sigma^2}\right)/4 \ge \sigma^2\mathcal{C}\left(\frac{t}{\sigma^2}\right)/4 .$$

□

Next, we recall the law of total variance.

Lemma 21 (Law of Total Variance)**.** *For every pair of random variables X, Y,*

$$\mathrm{Var}[Y] = \mathbb{E}[\mathrm{Var}[Y \mid X]] + \mathrm{Var}[\mathbb{E}[Y \mid X]] .$$

In particular, $\mathrm{Var}[Y] \ge \mathrm{Var}[\mathbb{E}[Y \mid X]]$.

We are now ready to prove the main theorem of the section, which informally states that concatenating the output values of hash functions preserves the property of having Chernoff-style bounds. Note that the following is a much more explicit and elaborate statement of Theorem 6. The purpose of this restatement is to make a formal proof more readable. The reader is encouraged to refer back to Theorem 6 for intuition regarding the theorem statement. Again, we highlight that we have left out the part of Theorem 6 concerning query independence. How query independence is obtained will be discussed in Section 7

Theorem 22. *Let A be a finite set. Let $(X_a)_{a\in A}$ and $(Y_a)_{a\in A}$ be pairwise independent families of random variables taking values in B_X and B_Y, respectively, and satisfying that the distributions of $(X_a)_{a\in A}$ and $(Y_a)_{a\in A}$ are independent. Suppose that there exist universal constants $D_X, D_Y \ge 1$, $K_X, K_Y > 0$, and $R_X, R_Y \ge 0$ such that for every choice of value functions $v_X \colon A \times B_X \to [0,1]$ and $v_Y \colon A \times B_Y \to [0,1]$ and for every $t > 0$,*

$$\Pr\left[\left|\sum_{a\in A} v_X(a, X_a) - \mu_X\right| > t\right] < K_X \exp\left(-\sigma_X^2\mathcal{C}\left(\frac{t}{\sigma_X^2}\right)/D_X\right) + R_X , \tag{25}$$

$$\Pr\left[\left|\sum_{a\in A} v_Y(a, Y_a) - \mu_Y\right| > t\right] < K_Y \exp\left(-\sigma_Y^2\mathcal{C}\left(\frac{t}{\sigma_Y^2}\right)/D_Y\right) + R_Y . \tag{26}$$

where $\mu_X = \mathbb{E}\left[\sum_{a\in A} v_X(a, X_a)\right]$, $\mu_Y = \mathbb{E}\left[\sum_{a\in A} v_Y(a, Y_a)\right]$, $\sigma_X^2 = \mathrm{Var}\left[\sum_{a\in A} v_X(a, X_a)\right]$, and $\sigma_Y^2 = \mathrm{Var}\left[\sum_{a\in A} v_Y(a, Y_a)\right]$. Then for every value function $\overline{v} \colon A \times B_X \times B_Y \to [0,1]$ and every $t > 0$,

$$\Pr\left[\left|\sum_{a\in A} \overline{v}(a, X_a, Y_a) - \mu_{XY}\right| > t\right] < K_{KY} \exp\left(-\sigma_{XY}^2\mathcal{C}\left(\frac{t}{\sigma_{XY}^2}\right)/D_{XY}\right) + R_{XY} ,$$

where $\mu_{XY} = \mathbb{E}\left[\sum_{a\in A} \overline{v}(a, X_a, Y_a)\right]$, $\sigma_{XY}^2 = \mathrm{Var}\left[\sum_{a\in A} \overline{v}(a, X_a, Y_a)\right]$, $D_{XY} = \max\{144 D_X, 72 D_Y\}$, $K_{XY} = 3K_X + K_Y$, and $R_{XY} = 3R_X + R_Y$.

Proof. Let a value function, $\overline{v} \colon A \times B_X \times B_Y \to [0,1]$, and a positive real, $t > 0$, be given. Define $V_a = \overline{v}(a, X_a, Y_a)$, $\mu_a = \mathbb{E}[V_a]$, and $\sigma_a^2 = \mathrm{Var}[V_a]$. We shall be concerned with the variance of V_a when conditioned on X_a. Hence, we define

$$L_a = \left[\mathrm{Var}[V_a \mid X_a] > \sqrt{\frac{6\sigma_{XY}^2}{t}}\right] \quad \text{and} \quad S_a = \left[\mathrm{Var}[V_a \mid X_a] \le \sqrt{\frac{6\sigma_{XY}^2}{t}}\right]$$

to be the indicators on the conditional variance of V_a given X_a being larger or smaller, respectively, than the threshold $\sqrt{\frac{6\sigma_{XY}^2}{t}}$. Noting that $L_a + S_a = 1$, we split the sum $\sum_{a\in A}(V_a - \mu_a)$ into three parts.

$$\begin{aligned}\sum_{a\in A}(V_a - \mu_a) &= \underbrace{\sum_{a\in A}(\mathbb{E}\left[V_a \mid X_a\right] - \mu_a)}_{T_1} \\ &\quad + \underbrace{\sum_{a\in A} L_a(V_a - \mathbb{E}\left[V_a \mid X_a\right])}_{T_2} \\ &\quad + \underbrace{\sum_{a\in A} S_a(V_a - \mathbb{E}\left[V_a \mid X_a\right])}_{T_3}\end{aligned}$$

Now, the triangle inequality and a union bound yields

$$\Pr\left[\left|\sum_{a\in A}\overline{v}(a, X_a, Y_a) - \mu_{XY}\right| > t\right] = \Pr\left[\left|\sum_{a\in A}(V_a - \mu_a)\right| > t\right] \le \sum_{i=1}^{3}\Pr\left[|T_i| > t/3\right].$$

We shall bound each of the three terms T_1, T_2, and T_3 individually.

For bounding $\Pr\left[|T_1| > t/3\right]$, define the value function $v_X^{(1)} \colon A \times B_X \to [0,1]$ by $v_X^{(1)}(a,x) = \mathbb{E}\left[V_a \mid X_a = x\right]$. Note that $\mathbb{E}\left[\mathbb{E}\left[V_a \mid X_a\right]\right] = \mu_a$ and $\operatorname{Var}\left[\mathbb{E}\left[V_a \mid X_a\right]\right] \le \sigma_a^2$, by the law of total variance, such that $\operatorname{Var}\left[\sum_{a\in A} v_X^{(1)}(a, X_a)\right] \le \sigma_{XY}^2$. Thus, by Equation (25) and Lemma 10,

$$\begin{aligned}\Pr\left[|T_1| > t/3\right] &= \Pr\left[\sum_{a\in A}\left(v_X^{(1)}(a, X_a) - \mu_a\right) > t/3\right] \\ &< K_X \exp\left(-\sigma_{XY}^2 \mathcal{C}\left(\frac{t/3}{\sigma_{XY}^2}\right)/D_X\right) + R_X \\ &\le K_X \exp\left(-\sigma_{XY}^2 \mathcal{C}\left(\frac{t}{\sigma_{XY}^2}\right)/(9D_X)\right) + R_X\ .\end{aligned}$$

For bounding $\Pr\left[|T_2| > t/3\right]$, we may assume that $t > 6\sigma_{XY}^2$ since otherwise $T_2 = 0$ almost surely. Now, recall that $L_a = \left[\operatorname{Var}\left[V_a \mid X_a\right] > \sqrt{6\sigma_{XY}^2/t}\right]$ and write $Z = \sum_{a\in A} L_a$. We observe that since $V_a \in [0,1]$ almost surely, $Z \ge |T_2|$ almost surely. By the law of total variance, $\mathbb{E}\left[\operatorname{Var}\left[V_a \mid X_a\right]\right] \le \sigma_a^2$, so by Markov's inequality,

$$\mathbb{E}\left[L_a\right] = \Pr\left[\operatorname{Var}\left[V_a \mid X_a\right] > \sqrt{\frac{6\sigma_{XY}^2}{t}}\right] \le \sigma_a^2\sqrt{\frac{t}{6\sigma_{XY}^2}}.$$

Now, $\operatorname{Var}\left[L_a\right] \le \mathbb{E}\left[L_a\right] \le \sigma_a^2\sqrt{t/(6\sigma_{XY}^2)}$ as $L_a \in [0,1]$ almost surely. Thus, $\mathbb{E}\left[Z\right] \le \sqrt{t\sigma_{XY}^2/6}$ and $\operatorname{Var}\left[Z\right] \le \sqrt{t\sigma_{XY}^2/6}$. Combining this with $t > 6\sigma_{XY}^2$, we may write

$$\Pr\left[|T_2| > t/3\right] \le \Pr\left[Z - \mathbb{E}\left[Z\right] > t/3 - \sqrt{t\sigma_{XY}^2/6}\right] \le \Pr\left[|Z - \mathbb{E}\left[Z\right]| > t/6\right].$$

Applying Equation (25) with the value function $v_X^{(2)} \colon A \times B_X \to [0,1]$ given by $v_X^{(2)}(a, X_a) = L_a$ to

$\Pr\left[|Z - \mathbb{E}[Z]| > t/6\right]$ yields

$$\begin{aligned}\Pr\left[|T_2| > t/3\right] &< K_X \exp\left(-\sqrt{t\sigma_{XY}^2/6} \cdot \mathcal{C}\left(\frac{t/6}{\sqrt{t\sigma_{XY}^2/6}}\right)/D_X\right) + R_X \\ &\le K_X \exp\left(-\sigma_{XY}^2 \mathcal{C}\left(\frac{t/6}{\sigma_{XY}^2}\right)/(4D_X)\right) + R_X \\ &\le K_X \exp\left(-\sigma_{XY}^2 \mathcal{C}\left(\frac{t}{\sigma_{XY}^2}\right)/(144 \cdot D_X)\right) + R_X,\end{aligned}$$

where the second follows from Lemma 20 and the third inequality follows from Lemma 10.

Lastly, we shall bound $\Pr\left[|T_3| > t/3\right]$. By a union bound,

$$\begin{aligned}\Pr\left[|T_3| > t/3\right] \le &\underbrace{\Pr\left[(|T_3| > t/3) \wedge \left(\mathrm{Var}\left[T_3 \mid (X_a)_{a\in A}\right] \le 2\max\left\{\sigma_{XY}^2, \sqrt{t\sigma_{XY}^2}\right\}\right)\right]}_{R_1} \\ &+ \underbrace{\Pr\left[\mathrm{Var}\left[T_3 \mid (X_a)_{a\in A}\right] > 2\max\left\{\sigma_{XY}^2, \sqrt{t\sigma_{XY}^2}\right\}\right]}_{R_2}.\end{aligned}$$

First, we bound R_1. For each $a \in A$, let $x_a \in B_X$ be given such that $P(\forall a \in A\colon X_a = x_a) > 0$. We bound the probability of R_1 conditioned on $(X_a = x_a)_{a\in A}$ and since our bound will be the same for every choice of $(x_a)_{a\in A}$, the bound will hold unconditionally. Now, if $\mathrm{Var}\left[T_3 \mid (X_a = x_a)_{a\in A}\right] > 2\max\left\{\sigma_{XY}^2, \sqrt{t\sigma_{XY}^2}\right\}$, then $R_1 = 0$. So assume otherwise and define the value function $v_Y^{(1)}\colon A \times B_Y \to [0,1]$ by $v_Y^{(1)}(a, y) = S_a \cdot \overline{v}(a, x_a, y)$, where $S_a = \left[\mathrm{Var}\left[V_a \mid X_a = x_a\right] \le \sqrt{6\sigma_{XY}^2/t}\right]$. Then $T_3 = \sum_{a\in A}\left(v_Y^{(1)}(Y_a) - \mathbb{E}\left[v_Y^{(1)}(Y_a)\right]\right)$ and by assumption, $\mathrm{Var}\left[\sum_{a\in A} v_Y^{(1)}(a, Y_a)\right] \le 2\max\left\{\sigma_{XY}^2, \sqrt{t\sigma_{XY}^2}\right\}$. Thus, we may apply Equation (26) with $v_Y^{(1)}$ to obtain

$$\begin{aligned}&\Pr\left[(|T_3| > t/3) \wedge \left(\mathrm{Var}\left[T_3 \mid (X_a)_{a\in A}\right] \le 2\max\left\{\sigma_{XY}^2, \sqrt{t\sigma_{XY}^2}\right\}\right) \,\middle|\, (X_a = x_a)_{a\in A}\right] \\ &\le K_Y \exp\left(-2\max\left\{\sigma_{XY}^2, \sqrt{t\sigma_{XY}^2}\right\}\mathcal{C}\left(\frac{t/3}{2\max\left\{\sigma_{XY}^2, \sqrt{t\sigma_{XY}^2}\right\}}\right)/D_Y\right) + R_Y \\ &\le K_Y \exp\left(-\max\left\{\sigma_{XY}^2, \sqrt{t\sigma_{XY}^2}\right\}\mathcal{C}\left(\frac{t}{\max\left\{\sigma_{XY}^2, \sqrt{t\sigma_{XY}^2}\right\}}\right)/(18D_Y)\right) + R_Y \\ &\le K_Y \exp\left(-\sigma_{XY}^2 \mathcal{C}\left(\frac{t}{\sigma_{XY}^2}\right)/(72D_Y)\right) + R_Y,\end{aligned}$$

where the second follows from Lemma 10 and the third inequality follows from Lemma 20. In conclusion,

$$R_1 \le K_Y \exp\left(-\sigma_{XY}^2 \mathcal{C}\left(\frac{t}{\sigma_{XY}^2}\right)/(72D_Y)\right) + R_Y.$$

Second, we bound R_2. Define the value function $v_X^{(3)}\colon A \times B_X \to [0,1]$ by

$$v_X^{(3)}(a, x_a) = \left[\mathrm{Var}\left[V_a \mid X_a = x_a\right] \le \sqrt{\frac{6\sigma_{XY}^2}{t}}\right] \cdot \mathrm{Var}\left[V_a \mid X_a = x_a\right].$$

Then $\mathrm{Var}\left[T_3 \mid (X_a)_{a\in A}\right] = \sum_{a\in A} v_X^{(3)}(a, X_a)$. Now, by the law of total variance,

$$\mathbb{E}\left[\mathrm{Var}\left[T_3 \mid (X_a)_{a\in A}\right]\right] \le \mathrm{Var}\left[T_3\right] \le \sigma_{XY}^2,$$

and since $\sqrt{\frac{t}{6\sigma_{XY}^2}} v_X^{(3)}(X_a) \in [0,1]$ almost surely for every $a \in A$, pairwise independence yields

$$\mathrm{Var}\left[\sqrt{\frac{t}{6\sigma_{XY}^2}}\mathrm{Var}\left[T_3 \mid (X_a)_{a\in A}\right]\right] \le \mathbb{E}\left[\sqrt{\frac{t}{6\sigma_{XY}^2}}\mathrm{Var}\left[T_3 \mid (X_a)_{a\in A}\right]\right] \le \sqrt{t\sigma_{XY}^2/6}\,.$$

Applying Equation (25) with $v_X^{(3)}$, Lemma 20, and Lemma 10, we obtain

$$\begin{aligned}
R_2 &\le \Pr\left[\left|\mathrm{Var}\left[T_3 \mid (X_a)_{a\in A}\right] - \mathbb{E}\left[\mathrm{Var}\left[T_3 \mid (X_a)_{a\in A}\right]\right]\right| > \max\left\{\sigma_{XY}^2, \sqrt{t\sigma_{XY}^2}\right\}\right] \\
&= \Pr\left[\sqrt{\frac{t}{6\sigma_{XY}^2}}\left|\mathrm{Var}\left[T_3 \mid (X_a)_{a\in A}\right] - \mathbb{E}\left[\mathrm{Var}\left[T_3 \mid (X_a)_{a\in A}\right]\right]\right| > \max\left\{\sqrt{t\sigma_{XY}^2/6}, t/6\right\}\right] \\
&\le \Pr\left[\sqrt{\frac{t}{6\sigma_{XY}^2}}\left|\mathrm{Var}\left[T_3 \mid (X_a)_{a\in A}\right] - \mathbb{E}\left[\mathrm{Var}\left[T_3 \mid (X_a)_{a\in A}\right]\right]\right| > t/6\right] \\
&< K_X \exp\left(-\sqrt{t\sigma_{XY}^2/6}\,\mathcal{C}\left(\frac{t/6}{\sqrt{t\sigma_{XY}^2/6}}\right)/D_X\right) + R_X \\
&\le K_X \exp\left(-\sigma_{XY}^2\mathcal{C}\left(\frac{t/6}{\sigma_{XY}^2}\right)/(4D_X)\right) + R_X \\
&\le K_X \exp\left(-\sigma_{XY}^2\mathcal{C}\left(\frac{t}{\sigma_{XY}^2}\right)/(144D_X)\right) + R_X.
\end{aligned}$$

Combining the bounds on $\Pr\left[|T_i| > t/3\right]$ for $i \in \{1,2,3\}$ completes the proof. □

7 Query invariance

In the following, we will briefly explain for each of the main sections of the paper, why all theorems still hold when adding the condition of query invariance of Definition 2. Recall that query invariance comes into play when we have a hash function and a concentration bound in the following manner. The concentration bound is query invariant if for any hash key q, a *query key*, the concentration bound still holds whenever we condition the hash function on the hash value of q.

Simple Tabulation Hashing. In [38] it is observed that ordering the position characters $\alpha_1 \prec \cdots \prec \alpha_r$ such that $\alpha_1, \ldots, \alpha_c$ are the position characters of the query key q only worsens the bound on the groups, G_i, by a factor of 2. We consider a slightly more general case, but exactly the same argument still applies. Always imposing this ordering in our proofs lets us condition on the hash value of q and only causes some of the constants to increase by a small factor.

Tabulation-Permutation In the proof of Theorem 5 we consider some specific value function w. We proceed by considering separately the m^ε bins $S \subset [m]$ of largest contribution to the variance, σ^2, and then the remaining bins, $[m] \setminus S$. The contribution of each subset of bins is then individually bounded. In the first case, we simply use the assumption on the hash function h that we received in a black box manner and use no properties of the permutation. Now, say towards query invariance that we require that $\tau \circ h(q) = i$. To support this, we instead chose S to have have size $|S| = m^\varepsilon/2$. This does not change the proof by more than constant factors and simply adding i to S yields a set $S' = S \cup \{i\}$ of size $S' < m^\varepsilon$, such that the assumption on h directly yields the result. In conclusion, the proof goes through exactly as before.

Extending the Codomain In this section nothing in the proof requires us to take into special consideration the conditioning on a query key. We simply consider families of hash functions in a black box manner and thus, we may as well consider families that have already been condition on the hash value of the query key q.

8 Tightness of Concentration: Simple Tabulation into Few Bins

Recall the result of Theorem 1. If $h\colon [u] \to [m]$ is a simple tabulation hash function with $[u] = \Sigma^c$ and $c = O(1)$, and $S \subseteq [u]$ is a set of hash keys of size $n = |S|$ where each key $x \in S$ is given a weight $w_x \in [0,1]$. Then for arbitrary $y \in [m]$ and a constant $\gamma > 0$ the total weight of the balls landing in bin y, given by the random variable $X = \sum_{x\in S} w_x[h(x) = y]$, satisfies the concentration bound

$$\Pr\left[|X - \mu| \geq t\right] \leq 2\exp(-\Omega(\sigma^2\mathcal{C}(t/\sigma^2))) + n/m^\gamma, \tag{27}$$

where $\mu = \mathbb{E}[X]$ and $\sigma^2 = \mathrm{Var}[X]$ are the expectation and variance, respectively, of X, and the constant in the Ω-notation depends on γ. As mentioned in the introduction, the added error probability n/m^γ renders the theorem nearly useless for small m, the prime example being the tossing of an unbiased coin corresponding to $m = 2$. The purpose of this section is to show that the bound of (27) is optimal in the sense that an added error probability of at least $m^{-\gamma}$ for some constant γ is inevitable so long as we insist on strong concentration according to Definition 1. In other words, we must accept an added error probability of $m^{-\gamma}$ to have Chernoff-style bounds on the sum X. In fact, it will turn out that unless allowing an error term of the form $m^{-\gamma}$, the deviation from the case of a fully random hash function can be quite significant.

The example where simple tabulation does not concentrate well, which we shall use in the formal proof below, is the following. For some $k < |\Sigma|$, we consider the key set $S = [k]^{c-1} \times \Sigma \subset \Sigma^c$ with weights $w_x = 1$ for every $x \in S$. We shall think of k as slightly superconstant and mutually dependent on γ. Recall that h is defined by c fully random functions $h_0, \dots, h_{c-1}\colon \Sigma \to [m]$ and that $h(x) = \bigoplus_{i=0}^{c} h_i(x_i)$. With probability $m^{-(k-1)(c-1)}$, h_i is constant on $[k]$ for each $0 \leq i \leq c-2$. Under such a *collapse* it holds for every $\alpha \in \Sigma$ that every key from the set $[k]^{c-1} \times \{\alpha\}$ hashes to the same value in $[m]$ under h. Hence, each entry of h_{c-1} decides where k^{c-1} keys hash to. Thus, during such a collapse, we may view the hashing of S into $[m]$ as throwing $|\Sigma|$ balls each of weight k^{c-1} into m bins. This increases the variance by a factor of k^{c-1} affecting the Chernoff bounds accordingly.

Without further ado, let us present the formal statement of the above. Essentially, it states that there is a *delay* of the exponential decrease which depends on γ. If γ is superconstant, so is the delay, and hence, we do not have strong concentration according to Definition 1.

Theorem 23. *Let $m \leq |\Sigma|^{1-\varepsilon}$ for some constant $\varepsilon > 0$ and $h\colon [u] \to [m]$ be a simple tabulation hash function. Let $0 < \varepsilon' < \varepsilon$ be a constant and suppose that $C\colon \mathbb{R}^+ \to \mathbb{R}^+$ is a function such that for all $0 \leq \gamma \leq |\Sigma|^{\varepsilon'/c}$, all sets $S \subseteq [u]$, all choices of weights $w_x \in [0,1], x \in S$, and every $y \in [m]$, the random variable $X = \sum_{x\in S} w_x[h(x) = y]$ satisfies*

$$\Pr[|X - \mu| \geq t] \leq 2\exp\left(\frac{-\sigma^2\mathcal{C}(t/\sigma^2)}{C(\gamma)}\right) + m^{-\gamma} \tag{28}$$

for all $t > 0$. Then $C(\gamma) = \Omega(\gamma^{c-2})$.

Proof. Assume the existence of the function C. As suggested above, consider the set of keys $S = [k]^{c-1} \times \Sigma$ for some k to be determined. Denote by $\mathcal{E}$ the event that the first $c-1$ position characters of S collapse, i.e., that each $h_i, 0 \leq i \leq c-2$ is constant on $[k]$. We easily calculate $\Pr[\mathcal{E}] = m^{-(k-1)(c-1)}$. Now, conditioning on $\mathcal{E}$, we may consider the situation as follows. Let y' be the random variable satisfying $\bigoplus_{i=0}^{c-2} h_i(x_i) = y'$ for all $x_0, \dots, x_{c-2} \in [k]$. The last positional hash function h_{c-1} is a fully random hash function $\Sigma \to m$ such that the conditioned variable $(X|\mathcal{E})$ satisfies

$$(X|\mathcal{E}) = \sum_{\alpha\in\pi_{c-1}(S)} k^{c-1}[h_{c-1}(\alpha) = y \oplus y'] \overset{d}{=} \sum_{\alpha\in\Sigma} k^{c-1}[h_{c-1}(\alpha) = 0] =: X',$$

where $\overset{d}{=}$ denotes equality of distribution. We write $\sigma'^2 = \mathrm{Var}[X'] = k^{2(c-1)} |\Sigma| \frac{m-1}{m^2}$ and note that $\mathbb{E}[X'] = \mu$. Now, since h_{c-1} is a uniformly random hash function, tightness of the Bennet inequality, Eq. (3), implies that for $t = O(\sigma'^2)$,

$$\Pr[|X' - \mu| \geq t] = \Omega\left(\exp\left(-\sigma'^2 \mathcal{C}(t/\sigma'^2)\right)\right) = \Omega\left(\exp\left(-\frac{t^2}{3\sigma'^2}\right)\right) \tag{29}$$

where we have applied Lemma 9.

Towards our main conclusion, observe that $\sigma^2 = k^{c-1} |\Sigma| \frac{m-1}{m^2} = \sigma'^2 / k^{c-1}$. Letting $t = \sigma' \sqrt{\log m}$, $t \leq \sigma'^2$ since $\sigma' > \sqrt{\log m}$ by the assumption on the size of m, so we may apply (29) to get

$$\Pr[|X - \mu| \geq t] \geq \Pr[\mathcal{E}] \cdot \Pr[|X' - \mu| \geq t] \geq \Omega\left(m^{-ck}\right).$$

On the other hand, (28) demands that whenever $k \leq \gamma$,

$$\Pr[|X - \mu| \geq t] \leq 2\exp\left(-\frac{\sigma^2 \mathcal{C}\left(\sqrt{\log(m) k^{c-1}}/\sigma\right)}{C(\gamma)}\right) + m^{-\gamma} \leq 2m^{-\frac{k^{c-1}}{C(\gamma)}} + m^{-\gamma},$$

where the last inequality used Lemma 9 and $\sqrt{\log(m)k^{c-1}} < \sigma$. Let $k = \frac{\gamma}{2c}$ and combine the above inequalities to conclude that $2m^{-\frac{(\gamma/(2c))^{c-1}}{C(\gamma)}} + m^{-\gamma} = \Omega(m^{-\gamma/2})$. It follows that indeed, $C(\gamma) = \Omega(\gamma^{c-2})$.

□

Twisted Tabulation and "permutation-tabulation" Variations upon the example above can also be used to show that the analysis of twisted tabulation hashing is tight in the sense that the added error probability cannot be improved while maintaining strong concentration. In twisted tabulation we *twist* the last position character of the input before applying simple tabulation. The twist is a Feistel permutation that for the key set $S = [k]^b \times \Sigma^{c-b}$, will only permute the keys within the set. Since the set of twisted keys is the same as the original set S, this has no effect on the filling of bins. For almost the same reason, a reversal of the order of operations in our new tabulation-permutation hashing, i.e., if is first permuted each position character and the applied simple tabulation, would not improve the analysis, since the set S while not invariant under the operation, would retain the same structure.

9 Experiments

This section is dedicated to provide further details regarding the timing experiments presented in the introduction in Section 1.7.4. Furthermore, we present experiments which demonstrate concrete bad input sets for several hash functions that do not guarantee strong concentration bounds.

As explained in Section 1.7.4, we ran experiments on various basic hash functions. More precisely, we compared our new hashing schemes tabulation-permutation and tabulation-1permutation with the following hashing schemes: k-independent PolyHash [9], Multiply-Shift [18], simple tabulation [49], twisted tabulation [41], mixed tabulation [16], and double tabulation [45]. We were interested in both the speed of the hash functions involved, and the quality of the output. For our timing experiments we studied the hashing of 32-bit keys to 32-bit hash values, and 64-bit keys to 64-bit hash values. Aside from having strong theoretical guarantees, our experiments show that tabulation-permutation and tabulation-1permutation are very fast in practice.

All experiments are implemented in C++11 using a random seed from `https://www.random.org`. The seed for the tabulation based hashing methods uses a random 100-independent PolyHash function. PolyHash is implemented using the Mersenne primes $p = 2^{61} - 1$ for 32 bits and $p = 2^{89} - 1$ for 64 bits. Furthermore, it has been implemented using Horner's rule, and GCC's 128-bit integers to ensure an efficient implementation. Double tabulation is implemented as described in [45] with $\Sigma = [2^{16}], c = 2, d = 20$.

	Running time (ms)			
	Computer 1		Computer 2	
Hash function	32 bits	64 bits	32 bits	64 bits
Multiply-Shift	4.2	7.5	23.0	36.5
2-Independent PolyHash	14.8	20.0	72.2	107.3
Simple tabulation	13.7	17.8	53.1	55.9
Twisted tabulation	17.2	26.1	65.6	92.5
Mixed tabulation	28.6	68.1	120.1	236.6
Tabulation-1permutation	16.0	19.3	63.8	67.7
Tabulation-permutation	27.3	43.2	118.1	123.6
Double tabulation	1130.1	–	3704.1	–
"Random" (100-Independent PolyHash)	2436.9	3356.8	7416.8	11352.6

Table 3: The time for different hash functions to hash 10^7 keys of length 32 bits and 64 bits, respectively, to ranges of size 32 bits and 64 bits. The experiment was carried out on two computers. The hash functions written in italics are those without general Chernoff-style bounds. Hash functions written in bold are the contributions of this paper. The hash functions in regular font are known to provide Chernoff-style bounds. Note that we were unable to implement double tabulation from 64 bits to 64 bits since the hash tables were too large to fit in memory.

Timing We timed the speed of the hash functions on two different computers. The first computer (Computer 1 in Table 3) has a 2.4 GHz Quad-Core Intel Core i5 processor and 8 GB RAM, and it is running macOS Catalina. The second computer (Computer 2 in Table 3) has 1.5 GHz Intel Core i3 processor and 4 GB RAM, and it is running Windows 10. We restate the results of our experiments in Table 3 and refer the reader to Section 1.7.4 for a discussion of these results and of the choice of parameters used in the various hashing schemes.

Quality We will now present experiments with concrete bad instances for the schemes without general concentration bounds, that is, Multiply-Shift, 2-independent PolyHash, simple tabulation, and twisted tabulation. In each case, we compare with our new tabulation-permutation scheme as well as 100-independent PolyHash, which is our approximation to an ideal fully random hash function. We note that all schemes considered are 2-independent, so they all have exactly the same variance as fully-random hashing. From 2-independence, it also follows that the schemes work perfectly on sufficiently random input [35]. Our concern is therefore concrete inputs making them fail in the tail.

First, we consider simple bad instances for Multiply-Shift and 2-independent PolyHash. These are analyzed in detail in [39, Appendix B]. The specific instance we consider is that of hashing the arithmetic progression $A = \{a \cdot i \mid i \in [50000]\}$ into 16 bins, where we are interested in the number of keys from A that hashes to a specific bin. We performed this experiment 5000 times, with independently chosen hash functions. The cumulative distribution functions on the number of keys from A hashing to a specific bin is presented in Figure 2. We see that most of the time 2-independent PolyHash and Multiply-Shift distribute the keys perfectly with exactly 1/16 of the keys in our bin. Since the variance is the same as with fully random hashing, this should suggest a much heavier tail, which is indeed what our experiments show. For contrast, we see that the cumulative distribution function with our tabulation-permutation hash function is almost indistinguishable from that of 100-independent Poly-Hash. We note that our experiments with tabulation-permutation is only a sanity check: No experiment can prove good performance on all possible inputs.

Our second set of experiments shows bad instances for simple tabulation and twisted tabulation. We already know theoretically from Section 8 that these bad instances exist, but we shall now see that, in a sense, things can be even worse than described in Section 8 for certain sets of keys. The specific instance we consider is hashing the discrete cube $Q = [2]^7 \times [2^6]$ to $m = 2$ bins using simple tabulation, twisted

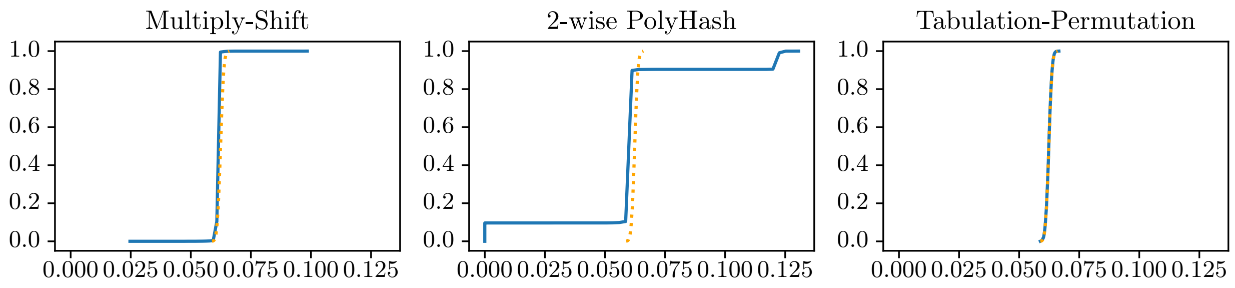

Figure 2: Hashing the arithmetic progression $\{a \cdot i \mid i \in [50000]\}$ to 16 bins for a random integer a. The dotted line is a 100-independent PolyHash.

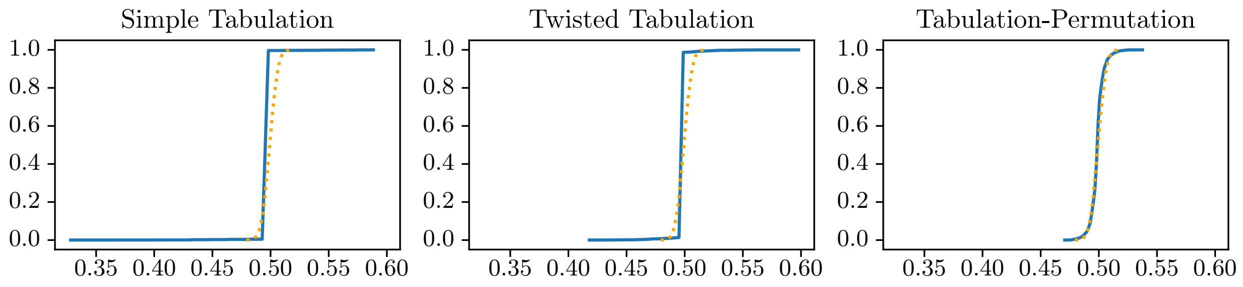

Figure 3: Hashing the discrete cube $[2]^7 \times [2^6]$ to 2 bins. The dotted line is a 100-independent PolyHash.

tabulation, and tabulation-permutation. We performed this experiment 5000 times, with independently chosen hash functions, and again we were interested in the number of keys from Q hashing to one of the bins. The cumulative distribution functions of the number of such keys is presented in Figure 3. Let us explain the appearance of the curves for simple and twisted tabulation. In general, if we hash the set of keys $[2] \times R$ to $[2]$ with simple tabulation, then if $h_1(0) \neq h_1(1)$, each bin will get exactly the same amount of keys. When we hash the set of keys $[2]^7 \times [2^6]$ this happens with probability $1 - 2^{-7}$. If on the other hand $h_i(0) = h_i(1)$ for each $i = 1, \dots, 7$, which happens with probability 2^{-7}, the distribution of the balls in the bins is the same as that when 2^6 balls, each of weight 2^7, are distributed independently and uniformly at random into the two bins. If this happens, the variance of the number of balls in a bin is a factor of 2^7 higher, so we expect a much heavier tail than in the completely independent case. These observations agree with the results in Figure 3. Most of the time, the distribution is perfect, but the tail is very heavy. We believe that this instance is also one of the worst instances for tabulation-permutation hashing. We would therefore expect to see that on this instance it performs slightly worse than 100-independent PolyHash, which is indeed what our experiments show. We note that that no amount of experimentation can prove that tabulation-permutation always works well for all inputs. We do, however, have mathematical concentration guarantees, and the experiments performed here give us some idea of the impact of the constant delay hidden in the exponential decrease in the bounds of Theorem 2. For completeness, we note that the situation with mixed tabulation is unresolved. Neither do we have strong concentration bounds, nor any bad instances showing that such bounds do not hold. Running experiments is not expected to resolve this issue since mixed tabulation, as any other 2-independent hashing scheme, performs well on almost all inputs [35].

Acknowledgement

Anders Aamand, Jakob B. T. Knudsen, Peter M. R. Rasmussen, and Mikkel Thorup are partly supported by Thorup's Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM

Foundation.

References

[1] Aamand, A., Das, D., Kipouridis, E., Knudsen, J. B. T., Rasmussen, P. M. R., and Thorup, M. No repetition: Fast streaming with highly concentrated hashing. *CoRR* (2020). `arxiv.org/abs/2004.01156`.

[2] Andersson, A., Miltersen, P. B., Riis, S., and Thorup, M. Static dictionaries on AC^0 RAMs: Query time $\Theta(\sqrt{\log n / \log\log n})$ is necessary and sufficient. In *37th Annual Symposium on Foundations of Computer Science (FOCS)* (1996), pp. 441–450.

[3] Appleby, A. MurmurHash3. `https://github.com/aappleby/smhasher/wiki/MurmurHash3`, 2016.

[4] Aumasson, J.-P., Neves, S., Wilcox-O'Hearn, Z., and Winnerlein, C. Blake2: Simpler, smaller, fast as MD5. In *Applied Cryptography and Network Security* (Berlin, Heidelberg, 2013), M. Jacobson, M. Locasto, P. Mohassel, and R. Safavi-Naini, Eds., Springer Berlin Heidelberg, pp. 119–135.

[5] Bar-Yossef, Z., Jayram, T. S., Kumar, R., Sivakumar, D., and Trevisan, L. Counting distinct elements in a data stream. In *International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)* (2002), pp. 1–10.

[6] Bennett, G. Probability inequalities for the sum of independent random variables. *Journal of the American Statistical Association 57*, 297 (1962), 33–45.

[7] Bernstein, S. N. On a modification of Chebyshev's inequality and of the error formula of Laplace. *Ann. Sci. Inst. Sav. Ukraine, Sect. Math.*, 1 (1924), 38–49.

[8] Broder, A. Z. On the resemblance and containment of documents. In *Compression and Complexity of Sequences (SEQUENCES)* (1997), pp. 21–29.

[9] Carter, L., and Wegman, M. N. Universal classes of hash functions. *Journal of Computer and System Sciences 18*, 2 (1979), 143–154. Announced at STOC'77.

[10] Celis, L. E., Reingold, O., Segev, G., and Wieder, U. Balls and bins: Smaller hash families and faster evaluation. In *52nd Annual Symposium on Foundations of Computer Science (FOCS)* (2011), pp. 599–608.

[11] Chandra, A. K., Stockmeyer, L. J., and Vishkin, U. Constant depth reducibility. *SIAM J. Comput. 13*, 2 (1984), 423–439.

[12] Chernoff, H. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics 23*, 4 (1952), 493–507.

[13] Christiani, T., and Pagh, R. Generating k-independent variables in constant time. In *55th Annual Symposium on Foundations of Computer Science (FOCS)* (2014), pp. 196–205.

[14] Christiani, T., Pagh, R., and Thorup, M. From independence to expansion and back again. In *Proceedings of the 47rd ACM Symposium on Theory of Computing (STOC)* (2015).

[15] Chung, K.-M., Mitzenmacher, M., and Vadhan, S. Why simple hash functions work: Exploiting the entropy in a data stream. *Theory of Computing 9*, 1 (2013), 897–945.

[16] Dahlgaard, S., Knudsen, M. B. T., Rotenberg, E., and Thorup, M. Hashing for statistics over k-partitions. In *56th Annual Symposium on Foundations of Computer Science (FOCS)* (2015), pp. 1292–1310.

[17] Dahlgaard, S., Knudsen, M. B. T., and Thorup, M. Practical hash functions for similarity estimation and dimensionality reduction. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS)* (2017), Curran Associates Inc., pp. 6618–6628.

[18] Dietzfelbinger, M. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In *Proceedings of the 13th Symposium on Theoretical Aspects of Computer Science (STACS)* (1996), pp. 569–580.

[19] Dietzfelbinger, M., and Meyer auf der Heide, F. Dynamic hashing in real time. In *Informatik, Festschrift zum 60. Geburtstag von Günter Hotz*. 1992, pp. 95–119.

[20] Dietzfelbinger, M., and Rink, M. Applications of a splitting trick. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP)* (2009), pp. 354–365.

[21] Dietzfelbinger, M., and Weidling, C. Balanced allocation and dictionaries with tightly packed constant size bins. *Theor. Comput. Sci. 380* (06 2007), 47–68.

[22] Dietzfelbinger, M., and Woelfel, P. Almost random graphs with simple hash functions. In *Proceedings of the 25th ACM Symposium on Theory of Computing (STOC)* (2003), pp. 629–638.

[23] Dumey, A. I. Indexing for rapid random access memory systems. *Computers and Automation 5*, 12 (1956), 6–9.

[24] Fan, X., Grama, I., and Liu, Q. Hoeffding's inequality for supermartingales. *Stochastic Processes and their Applications 122*, 10 (2012), 3545 – 3559.

[25] Fotakis, D., Pagh, R., Sanders, P., and Spirakis, P. Space efficient hash tables with worst case constant access time. *Theory of Computing Systems 38*, 2 (Feb 2005), 229–248.

[26] Gopalan, P., Kane, D. M., and Meka, R. Pseudorandomness via the discrete fourier transform. *SIAM J. Comput. 47*, 6 (2018), 2451–2487.

[27] Hagerup, T., and Tholey, T. Efficient minimal perfect hashing in nearly minimal space. In *Proceedings of the 18th Symposium on Theoretical Aspects of Computer Science (STACS)* (2001), pp. 317–326.

[28] Hennessy, J. L., and Patterson, D. A. *Computer Architecture - A Quantitative Approach, 5th Edition*. Morgan Kaufmann, 2012.

[29] Knuth, D. E. Notes on open addressing. Unpublished memorandum. See `http://citeseer.ist.psu.edu/knuth63notes.html`, 1963.

[30] Krishnamurthy, B., Sen, S., Zhang, Y., and Chen, Y. Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the 3rd Internet Measurement Conference (IMC)* (2003), pp. 234–247.

[31] Lemire, D., and Kaser, O. Faster 64-bit universal hashing using carry-less multiplications. *Journal of Cryptographic Engineering 6*, 3 (Sep 2016), 171–185.

[32] Mansour, Y., Nisan, N., and Tiwari, P. The computational complexity of universal hashing. *Theor. Comput. Sci. 107*, 1 (1993), 121–133.

[33] Meka, R., Reingold, O., Rothblum, G. N., and Rothblum, R. D. Fast pseudorandomness for independence and load balancing - (extended abstract). In *Proceedings of the 41st International Colloquium on Automata, Languages and Programming (ICALP)* (2014), pp. 859–870.

[34] Miltersen, P. B. Lower bounds for static dictionaries on rams with bit operations but no multiplication. In *Proceedings of the 23rd International Colloquium on Automata, Languages and Programming (ICALP)* (1996), pp. 442–453.

[35] Mitzenmacher, M., and Vadhan, S. P. Why simple hash functions work: exploiting the entropy in a data stream. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2008), pp. 746–755.

[36] Motwani, R., and Raghavan, P. *Randomized Algorithms.* Cambridge University Press, 1995.

[37] Pagh, A., and Pagh, R. Uniform hashing in constant time and optimal space. *SIAM Journal on Computing 38*, 1 (2008), 85–96.

[38] Pătraşcu, M., and Thorup, M. The power of simple tabulation-based hashing. *Journal of the ACM 59*, 3 (2012), Article 14. Announced at STOC'11.

[39] Pătraşcu, M., and Thorup, M. On the k-independence required by linear probing and minwise independence. *ACM Trans. Algorithms 12*, 1 (2016), 8:1–8:27.

[40] Pike, G., and Alakuijala, J. Introducing cityhash. `https://opensource.googleblog.com/2011/04/introducing-cityhash.html`, 2011.

[41] Pătraşcu, M., and Thorup, M. Twisted tabulation hashing. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2013), pp. 209–228.

[42] Schilling, R. L. *Measures, Integrals and Martingales.* Cambridge University Press, 2005.

[43] Schmidt, J. P., Siegel, A., and Srinivasan, A. Chernoff-Hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics 8*, 2 (1995), 223–250. Announced at SODA'93.

[44] Siegel, A. On universal classes of extremely random constant-time hash functions. *SIAM Journal on Computing 33*, 3 (2004), 505–543. Announced at FOCS'89.

[45] Thorup, M. Simple tabulation, fast expanders, double tabulation, and high independence. In *54th Annual Symposium on Foundations of Computer Science (FOCS)* (2013), pp. 90–99.

[46] Thorup, M. High speed hashing for integers and strings. *CoRR* (2015). `arxiv.org/abs/1504.06804`.

[47] Thorup, M., and Zhang, Y. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM Journal on Computing 41*, 2 (2012), 293–331. Announced at SODA'04 and ALENEX'10.

[48] Wegman, M. N., and Carter, L. New classes and applications of hash functions. *Journal of Computer and System Sciences 22*, 3 (1981), 265–279. Announced at FOCS'79.

[49] Zobrist, A. L. A new hashing method with application for game playing. Tech. Rep. 88, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1970.

Appendix B

No Repetition: Fast Streaming with Highly Concentrated Hashing

arXiv:2004.01156v1 [cs.DS] 2 Apr 2020

No Repetition: Fast Streaming with Highly Concentrated Hashing

Anders Aamand* Debarati Das* Evangelos Kipouridis* Jakob B. T. Knudsen*
Peter M. R. Rasmussen* Mikkel Thorup*

April 3, 2020

Abstract

To get estimators that work within a certain error bound with high probability, a common strategy is to design one that works with constant probability, and then boost the probability using independent repetitions. Important examples of this approach are small space algorithms for estimating the number of distinct elements in a stream, or estimating the set similarity between large sets. Using standard strongly universal hashing to process each element, we get a sketch based estimator where the probability of a too large error is, say, $1/4$. By performing r independent repetitions and taking the median of the estimators, the error probability falls exponentially in r. However, running r independent experiments increases the processing time by a factor r.

Here we make the point that if we have a hash function with strong concentration bounds, then we get the same high probability bounds without any need for repetitions. Instead of r independent sketches, we have a single sketch that is r times bigger, so the total space is the same. However, we only apply a single hash function, so we save a factor r in time, and the overall algorithms just get simpler.

Fast practical hash functions with strong concentration bounds were recently proposed by Aamand *et al.* (to appear in *STOC 2020*). Using their hashing schemes, the algorithms thus become very fast and practical, suitable for online processing of high volume data streams.

*Basic Algorithms Research Copenhagen (BARC), University of Copenhagen.

1 Introduction

To get estimators that work within a certain error bound with high probability, a common strategy is to design one that works with constant probability, and then boost the probability using independent repetitions. A classic example of this approach is the algorithm of Bar-Yossef *et al.* [3] to estimate the number of distinct elements in a stream. Using standard strongly universal hashing to process each element, we get an estimator where the probability of a too large error is, say, $1/4$. By performing r independent repetitions and taking the median of the estimators, the error probability falls exponentially in r. However, running r independent experiments increases the processing time by a factor r.

Here we make the point that if we have a hash function with strong concentration bounds, then we get the same high probability bounds without any need for repetitions. Instead of r independent sketches, we have a single sketch that is $\Theta(r)$ times bigger, so the total space is essentially the same. However, we only apply a single hash function, processing each element in constant time regardless of r, and the overall algorithms just get simpler.

Fast practical hash functions with strong concentration bounds were recently proposed by Aamand *et al.* [1]. Using their hashing schemes, we get a very fast implementation of the above streaming algorithm, suitable for online processing of high volume data streams.

To illustrate a streaming scenario where the constant in the processing time is critical, consider the Internet. Suppose we want to process packets passing through a high-end Internet router. Each application only gets very limited time to look at the packet before it is forwarded. If it is not done in time, the information is lost. Since processors and routers use some of the same technology, we never expect to have more than a few instructions available. Slowing down the Internet is typically not an option. The papers of Krishnamurthy *et al.* [19] and Thorup and Zhang [25] explain in more detail how high speed hashing is necessary for their Internet traffic analysis. Incidentally, the hash function we use from [1] is a bit faster than the ones from [19, 25], which do not provide Chernoff-style concentration bounds.

The idea is generic and can be applied to other algorithms. We will also apply it to Broder's original min-hash algorithm [7] to estimate set similarity, which can now be implemented efficiently, giving the desired estimates with high probability.

Concentration Let us now be more specific about the algorithmic context. We have a key universe U, e.g., 64-bit keys, and a random hash function h mapping U uniformly into $R = (0, 1]$.

For some input set S and some fraction $p \in [0, 1)$, we want to know the number X of keys from S that hash below p. Here p could be an unknown function of S, but p should be independent of the random hash function h. Then the mean μ is $\mathbb{E}[X] = |S|p$.

If the hash function h is fully random, we get the classic Chernoff bounds on X (see, e.g, [20]):

$$\Pr[X \ge (1+\varepsilon)\mu] \le \exp(-\varepsilon^2\mu/3) \text{ for } 0 \le \varepsilon \le 1, \tag{1}$$

$$\Pr[X \le (1-\varepsilon)\mu] \le \exp(-\varepsilon^2\mu/2) \text{ for } 0 \le \varepsilon \le 1. \tag{2}$$

Unfortunately, we cannot implement fully random hash functions as it requires space as big as the universe.

To get something implementable in practice, Wegman and Carter [26] proposed strongly universal hashing. The random hash function $h : U \to R$ is *strongly universal* if for any given distinct keys $x, y \in U$, $(h(x), h(y))$ is uniform in R^2. The standard implementation of a strongly universal hash function into $[0, 1)$ is to pick large prime $\wp$ and two uniformly random numbers $a, b \in \mathbb{Z}_\wp$. Then $h_{a,b}(x) = ((ax + b) \bmod \wp)/\wp$ is strongly universal from $U \subseteq \mathbb{Z}_\wp$ to $R = \{i/\wp | i \in Z_\wp\} \subset [0, 1)$. Obviously it is not uniform in $[0, 1)$, but for any $p \in [0, 1)$, we have $\Pr[h(x) < p] \approx p$ with equality if $p \in R$. Below we ignore this deviation from uniformity in $[0, 1)$.

Assuming we have a strongly universal hash function $h : U \to [0, 1)$, we again let X be the number of elements from S that hash below p. Then $\mu = \mathbb{E}[X] = |S|p$ and because the hash values are 2-independent, we have $\operatorname{Var}[X] \le \mathbb{E}[X] = \mu$. Therefore, by Chebyshev's inequality,

$$\Pr[|X - \mu| \ge \varepsilon\mu] < 1/(\varepsilon^2\mu).$$

As $\varepsilon^2\mu$ gets large, we see that the concentration we get with strongly universal hashing is much weaker than the Chernoff bounds with fully random hashing. However, Chebyshev is fine if we just aim at a constant error probability like 1/4, and then we can use the median over independent repetitions to reduce the error probability.

In this paper we discuss benefits of having hash functions with strong concentration akin to that of fully random hashing:

Definition 1. *A hash function $h : U \to [0,1)$ is* strongly concentrated with added error probability $\mathcal{E}$ *if for any set $S \subseteq U$ and $p \in [0,1)$, if X is the number of elements from S hashing below p, $\mu = p|S|$ and $\varepsilon \leq 1$, then*

$$\Pr\left[|X-\mu| \geq \varepsilon\mu\right] = 2\exp(-\Omega(\varepsilon^2\mu)) + \mathcal{E}.$$

If $\mathcal{E} = 0$, we simply say that h is strongly concentrated.

Another way of viewing the added error probability $\mathcal{E}$ is as follows. We have strong concentration as long as we do not aim for error probabilities below $\mathcal{E}$, so if $\mathcal{E}$ is sufficiently low, we can simply ignore it.

What makes this definition interesting in practice is that Aamand *et al.* [1] recently presented a fast practical small constant time hash function that for $U = [u] = \{0, \ldots, u-1\}$ is strongly concentrated with added error probability $u^{-\gamma}$ for any constant γ. This term is so small that we can ignore it in all our applications. The speed is obtained using certain character tables in cache that we will discuss later.

Next we consider our two streaming applications, distinct elements and set-similarity, showing how strongly concentrated hashing eliminates the need for time consuming independent repetitions. We stress that in streaming algorithms on high volume data streams, speed is of critical importance. If the data is not processed quickly, the information is lost.

Distinct elements is the simplest case, and here we will also discuss the ramifications of employing the strongly concentrated hashing of Aamand *et al.* [1] as well as possible alternatives.

2 Counting distinct elements in a data stream

We consider a sequence of keys $x_1, \ldots, x_s \in [u]$ where each element may appear multiple times. Using only little space, we wish to estimate the number n of distinct keys. We are given parameters ε and δ, and the goal is to create an estimator, $\hat{n}$, such that $(1-\varepsilon)n \leq \hat{n} \leq (1+\varepsilon)n$ with probability at least $1-\delta$.

Following the classic approach of Bar-Yossef *et al.* [3], we use a strongly universal hash function $h : U \to (0,1]$. For simplicity, we assume h to be collision free over U.

For some $k > 1$, we maintain the k smallest distinct hash values of the stream. We assume for simplicity that $k \leq n$. The space required is thus $O(k)$, so we want k to be small. Let $x_{(k)}$ be the key having the k'th smallest hash value under h and let $h_{(k)} = h(x_{(k)})$. As in [3], we use $\hat{n} = k/h_{(k)}$ as an estimator for n (we note that [3] suggests several other estimators, but the points we will make below apply to all of them).

The point in using a hash function h is that all occurrences of a given key x in the stream get the same hash value, so if S is the set of distinct keys, $h_{(k)}$ is just the k smallest hash value from S. In particular, $\hat{n}$ depends only on S, not on the frequencies of the elements of the stream. Assuming no collisions, we will often identify the elements with the hash values, so x_i is smaller than x_j if $h(x_i) \leq h(x_j)$.

We would like $1/h_{(k)}$ to be concentrated around n/k. For any probability $p \in [0,1]$, let $X^{<p}$ denote the number of elements from S that hash below p. Let $p_- = k/((1+\varepsilon)n)$ and $p_+ = k/((1-\varepsilon)n)$. Note that both p_- and p_+ are independent of the random hash function h. Now

$$\begin{aligned} 1/h_{(k)} \leq (1-\varepsilon)n/k &\iff X^{<p_+} < k = (1-\varepsilon)\mathbb{E}\left[X^{<p_+}\right] \\ 1/h_{(k)} > (1+\varepsilon)n/k &\iff X^{<p_-} \geq k = (1+\varepsilon)\mathbb{E}\left[X^{<p_-}\right], \end{aligned}$$

and these observations form a good starting point for applying probabilistic tail bounds as we now describe.

2.1 Strong universality and independent repetitions

Since h is strongly universal, the hash values of any two keys are independent, so for any p, we have $\mathrm{Var}\left[X^{<p}\right] \leq \mathbb{E}\left[X^{<p}\right]$, and so by Chebyshev's inequality,

$$\begin{aligned}\Pr\left[1/h_{(k)} \leq (1-\varepsilon)n/k\right] &< (1-\varepsilon)/(k\varepsilon^2)\\ \Pr\left[1/h_{(k)} > (1+\varepsilon)n/k\right] &\leq (1+\varepsilon)/(k\varepsilon^2).\end{aligned}$$

Assuming $\varepsilon \leq 1$, we thus get that

$$\Pr\left[|\hat{n}-n| > \varepsilon n\right] = \Pr\left[\left|1/h_{(k)} - n/k\right| > \varepsilon n/k\right] \leq 2/(k\varepsilon^2).$$

To get the desired error probability δ, we could now set $k = 2/(\delta\varepsilon^2)$, but if δ is small, e.g. $\delta = 1/u$, k becomes way too large. As in [3] we instead start by aiming for a constant error probability, δ_0, say $\delta_0 = 1/4$. For this value of δ_0, it suffices to set $k_0 = 8/\varepsilon^2$. We now run r (to be determined) independent experiments with this value of k_0, obtaining independent estimators for n, $\hat{n}_1, \ldots, \hat{n}_r$. Finally, as our final estimator, $\hat{n}$, we return the median of $\hat{n}_1, \ldots, \hat{n}_r$. Now for each $1 \leq i \leq r$, $\Pr[|\hat{n}_i - n| > \varepsilon n] \leq 1/4$ and these events are independent. If $|\hat{n} - n| \geq \varepsilon n$, then $|\hat{n}_i - n| \geq \varepsilon n$ for at least half of the $1 \leq i \leq r$. By the standard Chernoff bound (1), this probability can be bounded by

$$\Pr\left[|\hat{n}-n| > \varepsilon n\right] \leq \exp(-(r/4)/3) = \exp(-r/12).$$

Setting $r = 12\ln(1/\delta)$, we get the desired error probability $1/\delta$. The total number of hash values stored is $k_0 r = (8/\varepsilon^2)(12\ln(\delta)) = 96\ln(1/\delta)/\varepsilon^2$.

2.2 A better world with fully random hashing

Suppose instead that $h : [u] \to (0,1]$ is a fully random hash function. In this case, the standard Chernoff bounds (1) and (2) with $\varepsilon \leq 1$ yield

$$\begin{aligned}\Pr\left[1/h_{(k)} \leq (1-\varepsilon)n/k\right] &< \exp(-(k/(1-\varepsilon))\varepsilon^2/2)\\ \Pr\left[1/h_{(k)} > (1+\varepsilon)n/k\right] &\leq \exp(-(k/(1+\varepsilon))\varepsilon^2/3).\end{aligned}$$

Hence

$$\Pr\left[|\hat{n}-n| > \varepsilon n\right] = \Pr\left[|1/h_{(k)} - n/k| \geq \varepsilon n/k\right] \leq 2\exp(-k\varepsilon^2/6). \tag{3}$$

Thus, to get error probability δ, we just use $k = 6\ln(2/\delta)/\varepsilon^2$. There are several reasons why this is much better than the above approach using 2-independence and independent repetitions.

- It avoids the independent repetitions, so instead of applying $r = \Theta(\log(1/\delta))$ hash functions to each key we just need one. We thus save a factor of $\Theta(\log(1/\delta))$ in speed.
- Overall we store fewer hash values: $k = 6\ln(2/\delta)/\varepsilon^2$ instead of $96\ln(1/\delta)/\varepsilon^2$.
- With independent repetitions, we are tuning the algorithm depending on ε and δ, whereas with a fully-random hash function, we get the concentration from (3) for every $\varepsilon \leq 1$.

The only caveat is that fully-random hash functions cannot be implemented.

2.3 Using hashing with strong concentration bounds

We now discuss the effect of relaxing the abstract full-random hashing to hashing with strong concentration bounds and added error probability $\mathcal{E}$. Then for $\varepsilon \leq 1$,

$$\begin{aligned}\Pr\left[1/h_{(k)} \leq (1-\varepsilon)n/k\right] &= 2\exp(-\Omega(k/(1-\varepsilon))\varepsilon^2) + \mathcal{E}\\ \Pr\left[1/h_{(k)} > (1+\varepsilon)n/k\right] &= 2\exp(-\Omega(k/(1+\varepsilon))\varepsilon^2) + \mathcal{E}.\end{aligned}$$

so

$$\Pr[|\hat{n} - n| \geq \varepsilon n] = \Pr\left[|1/h_{(k)} - n/k| \geq \varepsilon n/k\right] \leq 2\exp(-\Omega(k\varepsilon^2)) + O(\mathcal{E}). \tag{4}$$

To obtain the error probability $\delta = \omega(\mathcal{E})$, we again need to store $k = O(\log(1/\delta)/\varepsilon^2)$ hash values. Within a constant factor this means that we use the same total number using 2-independence and independent repetitions, and we still retain the following advantages from the fully random case.

- With no independent repetitions we avoid applying $r = \Theta(\log(1/\delta))$ hash functions to each key, so we basically save a factor $\Theta(\log(1/\delta))$ in speed.
- With independent repetitions, we only address a given $\varepsilon \leq 1$ and δ, while with a fully-random hash function we get the concentration from (3) for every $\varepsilon \leq 1$.

2.4 Implementation and alternatives

We briefly discuss how to maintain the k smallest elements/hash values. The most obvious method is using a priority queue, but this takes $O(\log k)$ time per element, dominating the cost of evaluating the hash function. However, we can get down to constant time per element if we have a buffer for k. When the buffer gets full, we find the median in linear time with (randomized) selection and discard the bigger elements. This is standard to de-amortize if needed.

A different, and more efficient, sketch from [3] identifies the smallest b such that the number $X^{<1/2^b}$ of keys hashing below $1/2^b$ is at most k. For the online processing of the stream, this means that we increment b whenever $X^{<1/2^b} > k$. At the end, we return $2^b X^{<1/2^b}$. The analysis of this alternative sketch is similar to the one above, and we get the same advantage of avoiding independent repetitions using strongly concentrated hashing, that is, for error probability δ, in [3], they run $O(\log(1/\delta))$ independent experiments with independent hash functions, each storing up to $k = O(1/\varepsilon^2)$ hash values, whereas we run only a single experiment with a single strongly concentrated hash function storing $k = O(\log(1/\delta)/\varepsilon^2)$ hash values. The total number of hash values stored is the same, but asymptotically, we save a factor $\log(1/\delta)$ in time.

Other alternatives Estimating the number of distinct elements in a stream began with the work of Flajolet and Martin [13] and has continued with a long line of research [2, 3, 4, 5, 8, 9, 11, 12, 13, 14, 15, 16, 17, 27]. In particular, there has been a lot of focus on minimizing the sketch size. Theoretically speaking, the problem finally found an asymptotically optimal, both in time and in space, solution by Kane, Nelson and Woodruff [18], assuming we only need $\frac{2}{3}$ probability of success. The optimal space, including that of the hash function, is $O(\varepsilon^{-2} + \log n)$ bits, improving the $O(\varepsilon^{-2} \cdot \log n)$ bits needed by Bar-Yossef *et al.* [3] to store $O(\varepsilon^{-2})$ hash values. Both [3] and [18], suggest using $O(\log(1/\delta))$ independent repetitions to reduce the error probability to $1/\delta$, but then both time and space blow up by a factor $O(\log(1/\delta))$.

Recently Blasiok [6] found a space optimal algorithm for the case of small error probability $1/\delta$. In this case, the bound from [18] with independent repetitions was $O(\log(1/\delta)(\varepsilon^{-2} + \log n))$ which he reduces to $O(\log(1/\delta)\varepsilon^{-2} + \log n)$, again including the space for hash functions. He no longer has $O(\log(1/\delta))$ hash functions, but this only helps his space, not his processing time, which he states as polynomial in $\log(1/\delta)$ and $\log n$.

The above space optimal algorithms [6, 18] are very interesting, but fairly complicated, seemingly involving some quite large constants. However, here our focus is to get a fast practical algorithm to handle a high volume data stream online, not worrying as much about space. Assuming fast strongly concentrated hashing, it is then much better to use our implementation of the simple algorithm of Bar-Yossef *et al.* [3] using $k = O(\varepsilon^{-2}\log(1/\delta))$.

2.5 Implementing Hashing with Strong Concentration

As mentioned earlier, Aamand *et al.* [1] recently presented a fast practical small constant time hash function, Tabulation-1Permutation, that for $U = [u] = \{0, \ldots, u-1\}$ is strongly concentrated with additive error $u^{-\gamma}$ for any constant γ. The scheme obtains its power and speed using certain character tables in cache.

More specifically, we view keys as consisting of a small number c of characters from some alphabet Σ, that is, $U = \Sigma^c$. For 64-bit keys, this could be $c = 8$ characters of 8 bits each. Let's say that hash values are also from U, but viewed as bit strings representing fractions in $[0, 1)$.

Tabulation-1Permutation needs $c + 1$ character tables mapping characters to hash values. To compute the hash value of a key, we need to look up $c + 1$ characters in these tables. In addition we need $O(c)$ fast AC^0 operations to extract the characters and xor the hash values. The character tables can be populated with an $O(\log n)$ independent pseudo-random number generator, needing a random seed of $O((\log n)(\log u))$ bits.

Computer dependent versus problem dependent view of resources for hashing We view the resources used for Tabulation-1Permutation as computer dependent rather than problem dependent. When you buy a new computer you can decide how much cache you want to allocate for your hash functions. In the experiments performed in [1], using 8-bit characters and $c = 8$ for 64-bit keys was very efficient. On two computers, it was found that tabulation-1permutation was less than 3 times slower than the fastest known strongly universal hashing scheme; namely Dietzfelbinger's [10] which does just one multiplication and one shift. Also, Tabulation-1Permutation was more than 50 times faster than the fastest known highly independent hashing scheme; namely Thorup's [24] double tabulation scheme which, in theory also works in constant time.

In total, the space used by all the character tables is $9 \times 2^8 \times 64$ bits which is less than 20 KB, which indeed fits in very fast cache. We note that when we have first populated the tables with hash values, they are not overwritten. This means that the cache does not get dirty, that is different computer cores can access the tables and not worry about consistency.

This is different than the work space used to maintain the sketch of the number of distinct keys represented via $k = O(\varepsilon^{-2} \log(1/\delta))$ hash values, but let's compare anyway with real numbers. Even with a fully random hash function with perfect Chernoff bounds, we needed $k = 6 \ln(2/\delta)/\varepsilon^2$, so with, say, $\delta = 1/2^{30}$ and $\varepsilon = 1\%$, we get $k > 2^{20}$, which is much more than the 9×2^8 hash values stored in the character tables for the hash functions. Of course, we would be happy with a much smaller k so that everything is small and fits in fast cache.

We note that any $k > |\Sigma| = 2^8$ rules out the concentration of previous tabulation schemes such a simple tabulation [21] and twisted tabulation [22]. The reader is referred to [1] for a thorough discussion of the alternatives.

Finally, we relate our strong concentration from Definition 1 to the exact concentration result from [1]:

Theorem 1. *Let $h\colon [u] \to [r]$ be a tabulation-1permutation hash function with $[u] = \Sigma^c$ and $[r] = \Sigma^d$, $c, d = O(1)$. Consider a key/ball set $S \subseteq [u]$ of size $n = |S|$ where each ball $x \in S$ is assigned a weight $w_x \in [0, 1]$. Choose arbitrary hash values $y_1, y_2 \in [r]$ with $y_1 \le y_2$. Define $X = \sum_{x \in S} w_x \cdot [y_1 \le h(x) < y_2]$ to be the total weight of balls hashing to the interval $[y_1, y_2)$. Write $\mu = \mathbb{E}[X]$ and $\sigma^2 = \mathrm{Var}[X]$. Then for any constant γ and every $t > 0$,*

$$\Pr[|X - \mu| \ge t] \le 2 \exp(-\Omega(\sigma^2\, \mathcal{C}(t/\sigma^2))) + 1/u^\gamma. \tag{5}$$

Here $\mathcal{C} : (-1, \infty) \to [0, \infty)$ is given by $\mathcal{C}(x) = (x + 1) \ln(x + 1) - x$, so $\exp(-\mathcal{C}(x)) = \frac{e^x}{(1+x)^{(1+x)}}$. The above also holds if we condition the random hash function h on a distinguished query key q having a specific hash value.

The above statement is far more general than what we need. All our weights are unit weights. We fix $r = u$ and $y_1 = 0$. Viewing hash values as fractions in $[0, 1)$, the random variable X is the number of items hashing below $p = y_2/u$. Also, since $\mathrm{Var}[X] \le \mathbb{E}[X]$, (5) implies the same statement with μ instead of σ^2. Moreover, our $\varepsilon \le 1$ corresponds to $t = \varepsilon\mu \le \mu$, and then we get

$$\Pr[|X - \mu| \ge \varepsilon\mu] \le 2 \exp(-\Omega(\mu\, \mathcal{C}(\varepsilon))) + 1/u^\gamma \le 2 \exp(-\Omega(\mu\varepsilon^2)) + 1/u^\gamma.$$

which is exactly as in our Definition 1. Only remaining difference is that Definition 1 should work for *any* $p \in [0, 1)$ while the bound we get only works for p that are multiples of $1/u$. However, this suffices by the following general lemma:

Lemma 2. *Suppose we have a hash function $h : [u] \to [0,1)$ such that for any set $S \subseteq U$ and for any $p \in [0,1)$ that is a multiple of $1/u$, for the number $X^{<p}$ of elements from S that hash below p, with $\mu_p = p|S|$ and $\varepsilon \le 1$, it holds that*

$$\Pr\left[|X^{<p} - \mu_p| \ge \varepsilon\mu_p\right] \le 2\exp(-\Omega(\varepsilon^2\mu_p)) + O(\mathcal{E}).$$

Then the same statement holds for all $p \in [0,1)$

Proof. First we note that the statement is trivially true if $\varepsilon^2\mu_p = O(1)$, so we can assume $\varepsilon^2\mu_p = \omega(1)$. Since $\varepsilon \le 1$, we also have $\mu_p = \omega(1)$.

We are given an arbitrary $p \in [0,1)$. Let $p_+ = i/u$ be the nearest higher multiple of $1/u$. Since $|S| \le u$ and $\mu_p = p|S|$ we have $i \ge \mu_p$, implying $i = \omega(1)$. We also let $p_- = (i-1)/u$.

It is now clear that since $p_- < p \le p_+$, it holds that $X^{<p_-} \le X^{<p} \le X^{<p_+}$. We first show that

$$X^{<p} \le (1-\varepsilon)\mu_p \implies X^{<p_-} \le (1-\varepsilon/2)\mu_{p_-}.$$

Indeed, $X^{<p} \le (1-\varepsilon)\mu_p$ implies $X^{<p_-} \le (1-\varepsilon)p|S| \le (1-\varepsilon)(p_- + 1/u)|S| = \mu_{p_-} - \varepsilon\mu_{p_-} + (1-\varepsilon)|S|/u$.

But $|S| \le u$ and $(1-\varepsilon) < 1$, so $X^{<p_-} \le \mu_{p_-} - \varepsilon\mu_{p_-} + 1 \le (1-\varepsilon/2)\mu_{p_-}$. The last follows from the fact that $(\varepsilon/2)\mu_{p_-} \ge (\varepsilon/2)\mu_p - (\varepsilon/2)|S|/u \ge (\varepsilon^2/2)\mu_p - 1$, but $\varepsilon^2\mu_p = \omega(1)$ and so $(\varepsilon/2)\mu_{p_-} = \omega(1)$.

The exact same reasoning gives

$$X^{<p} \ge (1+\varepsilon)\mu_p \implies X^{<p_+} \ge (1+\varepsilon/2)\mu_{p_+}.$$

But then

$$\Pr\left[|X^{<p} - \mu_p| \ge \varepsilon\mu_p\right] = \Pr\left[X^{<p} \le (1-\varepsilon)\mu_p\right] + \Pr\left[X^{<p} \ge (1+\varepsilon)\mu_p\right] \le$$

$$\Pr\left[X^{<p_-} \le (1-\varepsilon/2)\mu_{p_-}\right] + \Pr\left[X^{<p_+} \ge (1+\varepsilon/2)\mu_{p_+}\right] \le$$

$$\Pr\left[|X^{<p_-} - \mu_{p_-}| \ge (\varepsilon/2)\mu_{p_-}\right] + \Pr\left[|X^{<p_+} - \mu_{p_+}| \ge (\varepsilon/2)\mu_{p_+}\right] \le$$

Notice that $\mu_p - 1 \le \mu_{p_-} \le \mu_{p_+}$, and p_- and p_+ are multiples of $1/u$, so we can use the bounds of the statement. Thus $\Pr\left[|X^{<p} - \mu_p| \ge \varepsilon\mu_p\right]$ is upper bounded by

$$4\exp(-\Omega((\varepsilon/2)^2(\mu_p - 1))) + O(\mathcal{E}) = 2\exp(-\Omega(\varepsilon^2\mu_p)) + O(\mathcal{E})$$

□

We note that [1] also presents a slightly slower scheme, Tabulation-Permutation, which offers far more general concentration bounds than those for Tabulation-1Permutation in Theorem 1. However, Tabulation-1Permutation is faster and sufficient for the strong concentration needed for our streaming applications.

3 Set similarity

We now consider Broder's [7] original algorithm for set similarity. As above, it uses a hash function $h : [u] \to [0,1]$ which we assume to be collision free. The bottom-k sample $\mathrm{MIN}_k(S)$ of a set $S \subseteq [u]$ consists of the k elements with the smallest hash values. If h is fully random then $\mathrm{MIN}_k(S)$ is a uniformly random subset of k distinct elements from $\mathrm{MIN}_k(S)$. We assume here that $k \le n = |S|$. With $\mathrm{MIN}_k(S)$, we can estimate the frequency $f = |T|/|S|$ of any subset $T \subseteq S$ as $|\mathrm{MIN}_k(S) \cap T|/k$.

Broder's main application is the estimation of the Jaccard similarity $f = |A \cap B|/|A \cup B|$ between sets A and B. Given the bottom-k samples from A and B, we may construct the bottom-k sample of their union as $\mathrm{MIN}_k(A \cup B) = \mathrm{MIN}_k(\mathrm{MIN}_k(A) \cup \mathrm{MIN}_k(B))$, and then the similarity is estimated as $|\mathrm{MIN}_k(A \cup B) \cap \mathrm{MIN}_k(A) \cap \mathrm{MIN}_k(B)|/k$.

We note again the crucial importance of having a common hash function h. In a distributed setting, samples $\mathrm{MIN}_k(A)$ and $\mathrm{MIN}_k(B)$ can be generated by different entities. As long as they agree on h, they only need to communicate the samples to estimate the Jaccard similarity of A and B. As noted before, for Tabulation-1Permutation h can be shared by exchanging a random seed of $O((\log n)(\log u))$ bits.

For the hash function h, Broder [7] first considers fully random hashing. Then $\mathrm{MIN}_k(S)$ is a fully random sample of k distinct elements from S, which is very well understood.

Broder also sketches some alternatives with realistic hash functions, but Thorup [23] showed that even if we just use 2-independence, we get the same expected error as with fully random hashing, but here we want strong concentration. Our analysis follows the simple union-bound approach from [23].

For the analysis, it is simpler to study the case where we are sampling from a set S and want to estimate the frequency $f = |T|/|S|$ of a subset $T \subseteq S$. Let $h_{(k)}$ be the kth smallest hash value from S as in the above algorithm for estimating distinct elements. For any p let $Y^{\le p}$ be the number of elements from T with hash value at most p. Then $|T \cap \mathrm{MIN}_k(S)| = Y^{\le h_{(k)}}$ which is our estimator for fk.

Theorem 3. *For $\varepsilon \le 1$, if h is strongly concentrated with added error probability $\mathcal{E}$, then*

$$\Pr\left[|Y^{\le h_{(k)}} - fk| > \varepsilon fk\right] = 2\exp(-\Omega(fk\varepsilon^2)) + O(\mathcal{E}). \tag{6}$$

Proof. Let $n = |S|$. We already saw in (4) that for any $\varepsilon_S \le 1$, $P_S = \Pr\left[|1/h_{(k)} - n/k| \ge \varepsilon_S n/k\right] \le 2\exp(-\Omega(k\varepsilon_S^2)) + O(\mathcal{E})$. Thus, with $p_- = k/((1+\varepsilon_S)n)$ and $p_+ = k/((1-\varepsilon_S)n)$, we have $h_{(k)} \in [p_-, p_+]$ with probability $1 - P_S$, and in that case, $Y^{\le p_-} \le Y^{\le h_{(k)}} \le Y^{\le p_+}$.

Let $\mu^- = \mathbb{E}\left[Y^{\le p_-}\right] = fk/(1+\varepsilon_S) \ge fk/2$. By strong concentration, for any $\varepsilon_T \le 1$, we get that

$$P_T^- = \Pr\left[Y^{\le p_-} \le (1-\varepsilon_T)\mu_-\right] \le 2\exp(-\Omega(\mu_-\varepsilon_T^2)) + \mathcal{E} = 2\exp(-\Omega(fk\varepsilon_T^2)) + \mathcal{E}.$$

Thus

$$\Pr\left[Y^{\le h_{(k)}} \le \frac{1-\varepsilon_T}{1+\varepsilon_S} fk\right] \le P_T^- + P_S.$$

Likewise, with $\mu^+ = \mathbb{E}\left[Y^{\le p_+}\right] = fk/(1-\varepsilon_S)$, for any ε_T, we get that

$$P_T^+ = \Pr\left[Y^{\le p_+} \ge (1+\varepsilon_T)\mu_+\right] \le 2\exp(-\Omega(\mu_+\varepsilon_T^2)) + \mathcal{E} = 2\exp(-\Omega(fk\varepsilon_T^2)) + \mathcal{E},$$

and

$$\Pr\left[Y^{\le h_{(k)}} \ge \frac{1+\varepsilon_T}{1-\varepsilon_S} fk\right] \le P_T^+ + P_S.$$

To prove the theorem for $\varepsilon \le 1$, we set $\varepsilon_S = \varepsilon_T = \varepsilon/3$. Then $\frac{1+\varepsilon_T}{1-\varepsilon_S} \le 1+\varepsilon$ and $\frac{1-\varepsilon_T}{1+\varepsilon_S} \ge 1-\varepsilon$. Therefore

$$\Pr\left[|Y^{\le h_{(k)}} - fk| \ge \varepsilon fk\right] \le P_T^- + P_T^+ + 2P_S \le 8\exp(-\Omega(fk\varepsilon_T^2)) + O(\mathcal{E}) = 2\exp(-\Omega(fk\varepsilon_T^2)) + O(\mathcal{E}).$$

This completes the proof of (6). □

As for the problem of counting distinct elements in a stream, in the online setting we may again modify the algorithm above to obtain a more efficient sketch. Assuming that the elements from S appear in a stream, we again identify the smallest b such that the number of keys from S hashing below $1/2^b$, $X^{\le 1/2^b}$, is at most k. We increment b by one whenever $X^{\le 1/2^b} > k$ and in the end we return $Y^{\le 1/2^b}/X^{\le 1/2^b}$ as an estimator for f. The analysis of this modified algorithm is similar to the analysis provided above.

Acknowledgements

Research of all authors partly supported by Thorup's Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation. Evangelos Kipouridis has also received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 801199.

References

[1] Aamand, A., Knudsen, J. B. T., Knudsen, M. B. T., Rasmussen, P. M. R., and Thorup, M. Fast hashing with strong concentration bounds. *CoRR abs/1905.00369* (2019). Accepted for STOC'20.

[2] Alon, N., Matias, Y., and Szegedy, M. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences 58*, 1 (1999), 209–223. Announced at STOC'96.

[3] Bar-Yossef, Z., Jayram, T. S., Kumar, R., Sivakumar, D., and Trevisan, L. Counting distinct elements in a data stream. In *International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)* (2002), pp. 1–10.

[4] Bar-Yossef, Z., Kumar, R., and Sivakumar, D. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proc. 13th ACM/SIAM Symposium on Discrete Algorithms (SODA)* (2002), pp. 623–632.

[5] Beyer, K. S., Haas, P. J., Reinwald, B., Sismanis, Y., and Gemulla, R. On synopses for distinct-value estimation under multiset operations. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007* (2007), pp. 199–210.

[6] Blasiok, J. Optimal streaming and tracking distinct elements with high probability. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018* (2018), pp. 2432–2448.

[7] Broder, A. Z. On the resemblance and containment of documents. In Proc. Compression and Complexity of Sequences (SEQUENCES) (1997), pp. 21–29.

[8] Brody, J., and Chakrabarti, A. A multi-round communication lower bound for gap hamming and some consequences. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity, CCC 2009, Paris, France, 15-18 July 2009* (2009), pp. 358–368.

[9] Cohen, E. Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and System Sciences 55*, 3 (1997), 441–453. Announced at STOC'94.

[10] Dietzfelbinger, M. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In *Proc. 13th Symposium on Theoretical Aspects of Computer Science (STACS)* (1996), pp. 569–580.

[11] Durand, M., and Flajolet, P. Loglog counting of large cardinalities (extended abstract). In *Proc. 11th European Symposium on Algorithms (ESA)* (2003), pp. 605–617.

[12] Estan, C., Varghese, G., and Fisk, M. E. Bitmap algorithms for counting active flows on high-speed links. *IEEE/ACM Trans. Netw. 14*, 5 (2006), 925–937.

[13] Flajolet, P., and Martin, G. N. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences 31*, 2 (1985), 182–209.

[14] Flajolet, P., Éric Fusy, Gandouet, O., and Meunier, F. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *In AOFA '07: Proceedings of the 2007 International Conference on Analysis of Algorithms* (2007), pp. 127–146.

[15] Gibbons, P. B. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy* (2001), pp. 541–550.

[16] Gibbons, P. B., and Tirthapura, S. Estimating simple functions on the union of data streams. In *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA 2001, Heraklion, Crete Island, Greece, July 4-6, 2001* (2001), pp. 281–291.

[17] Indyk, P., and Woodruff, D. P. Tight lower bounds for the distinct elements problem. In *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings* (2003), pp. 283–288.

[18] Kane, D. M., Nelson, J., and Woodruff, D. P. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2010, June 6-11, 2010, Indianapolis, Indiana, USA* (2010), pp. 41–52.

[19] Krishnamurthy, B., Sen, S., Zhang, Y., and Chen, Y. Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the 3rd ACM SIGCOMM Internet Measurement Conference, IMC 2003, Miami Beach, FL, USA, October 27-29, 2003* (2003), pp. 234–247.

[20] Motwani, R., and Raghavan, P. *Randomized Algorithms.* Cambridge University Press, 1995.

[21] Pǎtraşcu, M., and Thorup, M. The power of simple tabulation-based hashing. *Journal of the ACM 59*, 3 (2012), Article 14. Announced at STOC'11.

[22] Pǎtraşcu, M., and Thorup, M. Twisted tabulation hashing. In *Proc. 24th ACM/SIAM Symposium on Discrete Algorithms (SODA)* (2013), pp. 209–228.

[23] Thorup, M. Bottom-k and priority sampling, set similarity and subset sums with minimal independence. In *Proc. 45th ACM Symposium on Theory of Computing (STOC)* (2013).

[24] Thorup, M. Simple tabulation, fast expanders, double tabulation, and high independence. In *Proc. 54th IEEE Symposium on Foundations of Computer Science (FOCS)* (2013), pp. 90–99.

[25] Thorup, M., and Zhang, Y. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM Journal on Computing 41*, 2 (2012), 293–331. Announced at SODA'04 and ALENEX'10.

[26] Wegman, M. N., and Carter, L. New classes and applications of hash functions. *Journal of Computer and System Sciences 22*, 3 (1981), 265–279. Announced at FOCS'79.

[27] Woodruff, D. P. Optimal space lower bounds for all frequency moments. In *Proc. 15th ACM/SIAM Symposium on Discrete Algorithms (SODA)* (2004), pp. 167–175.

Appendix C

Non-Empty Bins with Simple Tabulation Hashing

Non-Empty Bins with Simple Tabulation Hashing

Anders Aamand and Mikkel Thorup

August 31, 2020

Abstract

We consider the hashing of a set $X \subseteq U$ with $|X| = n$ using a simple tabulation hash function $h : U \to [m] = \{0, \dots, m-1\}$ and analyse the number of non-empty bins, that is, the size of $h(X)$. We show that the expected size of $h(X)$ matches that with fully random hashing to within low-order terms. We also provide concentration bounds. The number of non-empty bins is a fundamental measure in the balls and bins paradigm, and it is critical in applications such as Bloom filters and Filter hashing. For example, normally Bloom filters are proportioned for a desired low false-positive probability assuming fully random hashing (see `en.wikipedia.org/wiki/Bloom_filter`). Our results imply that if we implement the hashing with simple tabulation, we obtain the same low false-positive probability for any possible input.

1 Introduction

We consider the balls and bins paradigm where a set $X \subseteq U$ of $|X| = n$ balls are distributed into a set of m bins according to a hash function $h : U \to [m]$. We are interested in questions relating to the distribution of $|h(X)|$, for example: What is the expected number of non-empty bins? How well is $|h(X)|$ concentrated around its mean? And what is the probability that a query ball lands in an empty bin? These questions are critical in applications such as Bloom filters [3] and Filter hashing [7].

In the setting where h is a fully random hash function, meaning that the random variables $(h(x))_{x \in U}$ are mutually independent and uniformly distributed in $[m]$, the situation is well understood. The random distribution process is equivalent to throwing n balls sequentially into m bins by for each ball choosing a bin uniformly at random and independently of the placements of the previous balls. The probability that a bin becomes empty is thus $(1 - 1/m)^n$; so the expected number of non-empty bins is exactly $\mu_0 := m\ (1 - (1 - 1/m)^n)$ and, unsurprisingly, the number of non-empty bins turns out to be sharply concentrated around μ_0 (see for example Kamath et al. [8] for several such concentration results).

In practical applications fully random hashing is unrealistic and so it is desirable to replace the fully random hash functions with realistic and implementable hash functions that still provide at least some of the probabilistic guarantees that were available in the fully random setting. However, as the mutual independence of the keys is often a key ingredient in proving results in the fully random setting most of these proofs do not carry over. Often the results are simply no longer true and if they are one has to come up with alternative techniques for proving them.

In this paper, we study the number of non-empty bins when the hash function h is chosen to be a simple tabulation hash function [14, 21]; which is very fast and easy to implement (see description below in Section 1.1). We provide estimates on the expected size of $|h(X)|$ which asymptotically match[1] those with fully random hashing on any possible input. To get a similar match within the classic k-independence paradigm [20], we would generally need $k = \Omega((\log n)/(\log \log n))$. For comparison, simple tabulation is the fastest known 3-independent hash function [14]. We will also study how $|h(X)|$ is concentrated around its mean.

Our results complements those from [14], which show that with simple tabulation hashing, we get Chernoff-type concentration on the number of balls in a given bin when $n \gg m$. For example, the results from [14] imply that all bins are non-empty with high probability (whp) when $n = \omega(m \log m)$. More precisely, for any constant $\gamma > 0$, there exists a $C > 0$ such that if $n \geq Cm \log m$, all bins are non-empty with probability $1 - O(m^{-\gamma})$. As a consequence, we only have to study $|h(X)|$ for $n = O(m \log m)$ below. On the other hand, [14] does not provide any good bounds on the probability that a bin is non-empty when, say, $n = m$. In this case, our results imply that a bin is non-empty with probability $1 - 1/e \pm o(1)$, as in the fully random case. The understanding we provide here is critical to applications such as Bloom filters [3] and Filter hashing [7], which we describe in section 2.1 and 2.2.

We want to emphasize the advantage of having many complementary results for simple tabulation hashing. An obvious advantage is that simple tabulation can be reused in many contexts, but there may also be applications that need several strong properties to work in tandem. If, for example, an application has to hash a mix of a few heavy balls and many light balls, and the hash function do not know which is which, then the results from [14] give us the Chernoff-style concentration of the number of light balls in a bin while the results of this paper give us the right probability that a bin contains a heavy ball. For another example where an interplay of properties becomes important see section 2.2 on Filter hashing. The reader is

[1]Here we use "asymptotically" in the classic mathematical sense to mean equal to within low order terms, not just within a constant factor.

referred to [18] for a survey of results known for simple tabulation hashing, as well as examples where simple tabulation does not suffice and where slower more sophisticated hash functions are needed.

1.1 Simple tabulation hashing

Recall that a hash function h is a map from a universe U to a range R chosen with respect to some probability distribution on the set of all such functions. If the distribution is uniform (equivalently the random variables $(h(x))_{x\in U}$ are mutually independent and uniformly distributed in R) we will say that h is fully random.

Simple tabulation was introduced by Zobrist [21]. For simple tabulation $U = [u] = \{0,\dots,u-1\}$ and $R = [2^r]$ for some $r \in \mathbb{N}$. The keys $x \in U$ are viewed as vectors $x = (x[0],\dots,x[c-1])$ of $c = O(1)$ characters with each $x[i] \in \Sigma := [u^{1/c}]$. The simple tabulation hash function h is defined by

$$h(x) = \bigoplus_{i\in[c]} h_i(x[i]),$$

where $h_0,\dots,h_{c-1} : \Sigma \to R$ are independent fully random hash functions and where $\oplus$ denotes the bitwise XOR. What makes it fast is that the character domains of $h_0,\dots,h_{c-1}$ are so small that they can be stored as tables in fast cache. Experiments in [14] found that the hashing of 32-bit keys divided into 4 8-bit characters was as fast as two 64-bit multiplications. Note that on machines with larger cache, it may be faster to use 16-bit characters. As useful computations normally involve data and hence cache, there is no commercial drive for developing processors that do multiplications much faster than cache look-ups. Therefore, on real-world processors, we always expect cache based simple tabulation to be at least comparable in speed to multiplication. The converse is not true, since many useful computations do not involve multiplications. Thus there is a drive to make cache faster even if it is too hard/expensive to speed up multiplication circuits.

Other important properties include that the c character table lookups can be done in parallel and that when initialised the character tables are not changed. For applications such as Bloom filters where more than one hash function is needed another nice property of simple tabulation is that the output bits are mutually independent. Using (kr)-bit hash values is thus equivalent to using k independent simple tabulation hash functions each with values in $[2^r]$. This means that we can get k independent r-bit hash values using only c lookups of (kr)-bit strings.

1.2 Main Results

We will now present our results on the number of non-empty bins with simple tabulation hashing.

The expected number of non-empty bins. Our first theorem compares the expected number of non-empty bins when using simple tabulation to that in the fully random setting. We denote by $p_0 = 1 - (1 - 1/m)^n < n/m$ the probability that a bin becomes non-empty and by $\mu_0 = mp_0$ the expected number of non-empty bins when n balls are distributed into m bins using fully random hashing.

Theorem 1.1. *Let $X \subseteq U$ be a fixed set of $|X| = n$ balls. Let $y \in [m]$ be any bin and suppose that $h : U \to [m]$ is a simple tabulation hash function. If p denotes the probability that $y \in h(X)$ then*

$$|p - p_0| \le \frac{n^{2-1/c}}{m^2} \quad \textit{and hence} \quad |\mathbb{E}[|h(X)|] - \mu_0| \le \frac{n^{2-1/c}}{m}.$$

If we let y depend on the hash of a distinguished query ball $q \in U \backslash X$, e.g., $y = h(q)$, then the bound on p above is replaced by the weaker $|p - p_0| \leq \frac{2n^{2-1/c}}{m^2}$.

The last statement of the theorem is important in the application to Bloom filters where we wish to upper bound the probability that $h(q) \in h(X)$ for a query ball $q \notin X$.

To show that the expected relative error $|\mathbb{E}[|h(X)|] - \mu_0| / \mu_0$ is always small, we have to complement Theorem 1.1 with the result from [14] that all bins are full, whp, when $n \geq Cm \log m$ for some large enough constant C. In particular, this implies $|\mathbb{E}[|h(X)|] - \mu_0| / \mu_0 \leq 1/m$ when $n \geq Cm \log m$. The relative error from Theorem 1.1 is maximized when n is maximized, and with $n = Cm \log m$, it is bounded by $\frac{n^{2-1/c}}{m^2} = O((\log^2 m)/m^{1/c}) = \tilde{O}(m^{-1/c})$. Thus we conclude:

Corollary 1.2. *Let $X \subseteq U$ be a fixed sets of $|X| = n$ balls and let $h : U \to [m]$ be a simple tabulation hash function. Then $|\mathbb{E}[|h(X)|] - \mu_0| / \mu_0 = \tilde{O}(m^{-1/c})$.*

As discussed above, the high probability bound from [14] takes over when the bounds from Theorem 1.1 get weaker. This is because the analysis in this paper is of a very different nature than that in [14].

Concentration of the number of non-empty bins: We now consider the concentration of $|h(X)|$ around its mean. In the fully random setting it was shown by Kamath et al. [8] that the concentration of $|h(X)|$ around μ_0 is sharp: For any $\lambda \geq 0$ it holds that

$$\Pr(||h(X)| - \mu_0| \geq \lambda) \leq 2 \exp\left(-\frac{\lambda^2(m - 1/2)}{\mu_0(2n - \mu_0)}\right) \leq 2 \exp\left(-\frac{\lambda^2}{2\mu_0}\right),$$

which for example yields that $|h(X)| = \mu_0 \pm O(\sqrt{\mu_0 \log m})$ whp, that is, with probability $1 - O(m^{-\gamma})$ for any choice of $\gamma = O(1)$. Unfortunately we cannot hope to obtain such a good concentration using simple tabulation hashing. To see this, consider the set of keys $[2]^\ell \times [n/2^\ell]$ for any constant ℓ, e.g. $\ell = 1$, and let $\mathcal{E}$ be the event that $h_i(0) = h_i(1)$ for $i = 0, \ldots, \ell - 1$. This event occurs with probability $1/m^\ell$. Now if $\mathcal{E}$ occurs then the keys of $X_i = [2]^\ell \times \{i\}$ all hash to the same value namely $h_0(0) \oplus \cdots \oplus h_{\ell-1}(0) \oplus h_\ell(i)$. Furthermore, these values are independently and uniformly distributed in $[m]$ for $i \in [n/2^\ell]$ so the distribution of $|h(X)|$ becomes identical to the distribution of non-empty bins when $n/2^\ell$ balls are thrown into m bins using truly random hashing. This observation ruins the hope of obtaining a sharp concentration around μ_0 and shows that the lower bound in the theorem below is best possible being the expected number of non-empty bins when $\Omega(n)$ balls are distributed into m bins.

Theorem 1.3. *Let $X \subseteq U$ be a fixed sets of $|X| = n$ keys. Let $h : U \to [m]$ be a simple tabulation hash function. Then whp*

$$|h(X)| \geq m\left(1 - \left(1 - \frac{1}{m}\right)^{\Omega(n)}\right)$$

As argued above, the lower bound in Theorem 1.3 is optimal. Settling with a laxer requirement than high probability, it turns out however that $|h(X)|$ is somewhat concentrated around μ_0. This is the content of the following theorem which also provides a high probability upper bound on $|h(X)|$.

Theorem 1.4. *Let $X \subseteq U$ be a fixed sets of $|X| = n$ keys. Let $h : U \to [m]$ be a random simple tabulation hash function. For $t \geq 0$ it holds that*

$$\Pr[|h(X)| \geq \mu_0 + 2t] = O\left(\exp\left(\frac{-t^2}{2m^{2-1/c}}\right)\right), \qquad \textit{and} \tag{1.1}$$

$$\Pr[|h(X)| \leq \mu_0 - 2t] = O\left(\exp\left(\frac{-t^2}{2n^{2-1/c}}\right) + \frac{n^2}{mt^2}\right). \tag{1.2}$$

The term $n^2/(mt^2)$ in the second bound in the theorem may be unexpected but it has to be there (at least when $n = O(m)$) as we will argue after proving the theorem.

Theorem 1.4 is proved using Azuma's inequality (which we will state and describe later). It turns out that when $n \ll m$ one can obtain stronger concentration using a stronger martingale inequality. For intuition, the reader is encouraged to think of the fully random setting where n balls are thrown sequentially into m bins independently and uniformly at random: In this setting the allocation of a single ball can change the conditionally expected number of non-empty bins by at most 1 and this is the type of observation that normally suggests applying Azuma's inequality. However, when $n \ll m$, it is unlikely that the allocation of a ball will change the conditional expectation of the number of non-empty bins by much — for that to happen the ball has to hit a bin that is already non-empty, and the probability that this occurs is at most $n/m \ll 1$. Using a martingale inequality by Mcdiarmid [9], that takes the variance of our martingale into consideration, one can obtain the following result which is an improvement over Theorem 1.4 when $n \ll m$, and matches within O-notation when $n = \Theta(m)$.

Theorem 1.5. *Let $X \subseteq U$ be a fixed sets of $|X| = n$ keys. Let $h : U \to [m]$ be a random simple tabulation hash function. Assume $n \leq m$. For $t \geq 0$ it holds that*

$$\Pr\left[|h(X)| \geq \mu_0 + t\right] = \exp\left(-\Omega\left(\min\left\{\frac{t^2}{\frac{n^{3-1/c}}{m}}, \frac{t}{n^{1-1/c}}\right\}\right)\right), \qquad \textit{and} \tag{1.3}$$

$$\Pr\left[|h(X)| \leq \mu_0 - t\right] = \exp\left(-\Omega\left(\min\left\{\frac{t^2}{\frac{n^{3-1/c}}{m}}, \frac{t}{n^{1-1/c}}\right\}\right)\right) + O\left(\frac{n^2}{mt^2}\right). \tag{1.4}$$

The above bounds are unwieldy so let us disentangle them. First, one can show using simple calculus that when $2 \leq n \leq m$ then $\mu_0 = n - \Theta(n^2/m)$. If $n^{1+1/c} = o(m)$ we thus have that $\mu_0 = n - o(n^{1-1/c})$. To get a non-trivial bound from (1.3) we have to let $t = \Omega(n^{1-1/c})$ and then $\mu_0 + t = m + \omega(n^{1-1/c})$. This means that (1.3) is trivial when $n^{1+1/c} = o(m)$ as we can never have more than n non-empty bins. For comparison, (1.1) already becomes trivial when $n^{1+1/(2c)} = o(m)$.

Suppose now that $n^{1+1/c} = \Omega(m)$. For a given δ put

$$t_0 = \eta \max\left\{\sqrt{\frac{n^{3-1/c}}{m}\log\frac{1}{\delta}}, n^{1-1/c}\log\frac{1}{\delta}\right\},$$

for some sufficiently large $\eta = O(1)$. Then (1.3) gives that $\Pr\left[|h(X)| \geq \mu_0 + t_0\right] \leq \delta$. It remains to understand t_0: Assuming that $n^{1+1/c} \geq m \log\frac{1}{\delta}$, we have that $t_0 = O\left(\sqrt{\frac{n^{3-1/c}}{m}\log\frac{1}{\delta}}\right)$. For comparison, to get the same guarantee on the probability using (1.1) we would have to put $t_0 = \Omega\left(\sqrt{n^{2-1/c}\log\frac{1}{\delta}}\right)$, which is a factor of $\sqrt{m/n}$ larger.

Turning to (1.4), it will typically in applications be the term $O\left(\frac{n^2}{mt^2}\right)$ that dominates the bound. For a given δ we would choose $t = \max\{t_0, n/\sqrt{m\delta}\}$ to get $\Pr\left[|h(X)| \leq \mu_0 - t\right] = O(\delta)$.

1.3 Projecting into Arbitrary Ranges

Simple tabulation is an efficient hashing scheme for hashing into r-bit hash values. But what do we do if we want hash values in $[m]$ where $2^{r-1} < m < 2^r$, say $m = 3 \times 2^{r-2}$? Besides being of theoretical interest this is an important question in several practical applications. For example, when designing Bloom filters (which we will describe shortly), to minimize the false positive probability, we have to choose the size m of the filters such that $m \approx n/\ln(2)$. When

m has to be a power of two, we may be up to a factor of $\sqrt{2}$ off, and this significantly affects the false positive probability. Another example is cuckoo hashing [13], which was shown in [14] to succeed with simple tabulation with probability $1 - O(m^{-1/3})$ when $2n(1+\varepsilon) \leq m$. If $n = 2^r$ we have to choose m as large as $2^{r+2} = 4n$ to apply this result, making it much less useful.

The way we remedy this is a standard trick, see e.g. [17]. We choose r such that $2^r \gg m$, and hash in the first step to r-bit strings with a simple tabulation hash function $h : U \to [2^r]$. Usually $2^r \geq m^2$ suffices and then the entries of the character tables only becomes twice as long. Defining $s : [2^r] \to [m]$ by $s(y) = \lfloor ym/2^r \rfloor$ our combined hash function $U \to [m]$ is simply defined as $s \circ h$. Note that s is very easy to compute since we do just one multiplication and since the division by 2^r is just an r-bit right shift. The only property we will use about s is that it is *most uniform* meaning that for $z \in [m]$ either, $|s^{-1}(\{z\})| = \lfloor \frac{2^r}{m} \rfloor$ or $|s^{-1}(\{z\})| = \lceil \frac{2^r}{m} \rceil$. For example, we could also use $s' : [2^r] \to [m]$ defined by $s'(y) = y \pmod{m}$, but s is much faster to compute. Note that if $2^r \geq m^2$, then $\left| \frac{|s^{-1}(\{z\})|}{2^r} - \frac{1}{m} \right| \leq 2^{-r} \leq m^{-2}$.

A priori it is not obvious that $s \circ h$ has the same good properties as "normal" simple tabulation. The set of bins can now be viewed as $\{s^{-1}(\{z\}) : z \in [m]\}$, so each bin consists of many "sub-bins", and a result on the number of non-empty sub-bins does not translate directly to any useful result on the number of non-empty bins. Nonetheless, many proofs of results for simple tabulation do not need to be modified much in this new setting. For example, the simplified proof given by Aamand et al. [1] of the result on cuckoo hashing from [14] can be checked to carry over to the case where the hash functions are implemented as described above if r is sufficiently large. We provide no details here.

For the present paper the relevant analogue to Theorem 1.1 is the following:

Theorem 1.6. *Let $X \subseteq U$ be a fixed set of $|X| = n$ balls, and let $S \subseteq [2^r]$ with $|S|/2^r = \rho$. Suppose $h : U \to [2^r]$ is a simple tabulation hash function. Define $p'_0 = 1 - (1-\rho)^n$. If p denotes the probability that $h(X) \cap S \neq \emptyset$, then*

$$|p - p'_0| \leq n^{2-1/c}\rho^2$$

If we let S (and hence ρ) depend on the hash of a distinguished query ball $q \in U \backslash X$, then the bound on p above is replaced by the weaker $|p - p_0| \leq 2n^{2-1/c}\rho^2$.

If we assume $2^r \geq m^2$, say, and let $S = s^{-1}(\{z\})$ be a bin of $S \subset [2^r]$ we obtain the following estimate on p:

$$\begin{aligned} |p - p_0| &\leq |p - p'_0| + |p'_0 - p_0| \\ &\leq n^{2-1/c}\left(\frac{1}{n} + \frac{1}{2^r}\right)^2 + \frac{n}{2^r} = \frac{n^{2-1/c}}{m^2}(1 + o(1)) \end{aligned}$$

This is very close to what is obtained from Theorem 1.1 and to make the difference smaller we can increase r further.

There are also analogues of Theorem 1.3, 1.4 and 1.5 in which the bins are partitioned into groups of almost equal size and where the interest is in the number of groups that are hit by a ball. To avoid making this paper unnecessarily technical, we refrain from stating and proving these theorems, but in Section 5 we will show how to modify the proof of Theorem 1.1 to obtain Theorem 1.6.

1.4 Alternatives

One natural alternative to simple tabulation is to use k-independent hashing [20]. Using an easy variation[2] of an inclusion-exclusion based argument by Mitzenmacher and Vadhan [11], one can show that if k is odd and if $n \leq m$ the probability p that a given bin is non-empty satisfies

$$p_0 - O\left(\left(\frac{n}{m}\right)^k \frac{1}{k!}\right) \leq p \leq p_0 + O\left(\left(\frac{n}{m}\right)^{k+1} \frac{1}{(k+1)!}\right), \tag{1.5}$$

and this is optimal, at least when k is not too large, say $k = o(\sqrt{n})$ — there exist two (different) k-independent families making respectively the upper and the lower bound tight for a certain set of n keys. A similar result holds when k is even. Although p approaches p_0 when k increases, for $k = O(1)$ and $n = \Omega(m)$, we have a deviation by an additive constant term. In contrast, the probability that a bin is non-empty when using simple tabulation is asymptotically the same as in the fully random setting.

Another alternative when studying the number of non-empty bins is to assume that the input comes with a certain amount of randomness. This was studied in [11] too and a slight variation[2] of their argument shows that if the input $X \subseteq U$ has enough entropy the probability that a bin is empty is asymptotically the same as in the fully random setting even if we only use 2-independent hashing. This is essentially what we get with simple tabulation. However, our results have the advantage of holding for any input with no assumptions on its entropy. Now (1.5) also suggests the third alternative of looking for highly independent hash functions. For the expectation (1.5) shows that if $n \leq m$ we would need $k = \Omega(\log m / \log\log m)$ to get guarantees comparable to those obtained for simple tabulation. Such highly independent hash functions were first studied by Siegel [15], the most efficient known construction today being the double tabulation by Thorup [16] which gives independence $u^{\Omega(1/c^2)} \gg \log m$ using space $O(cu^{1/c})$ and time $O(c)$. While this space and time matches that of simple tabulation within constant factors, it is slower by at least an order of magnitude. As mentioned in [16], double tabulation with 32-bit keys divided into 16-bit characters requires 11 times as many character table lookups as with simple tabulation and we lose the same factor in space. The larger space of double tabulation means that tables may expand into much slower memory, possibly costing us another order of magnitude in speed.

There are several other types of hash functions that one could consider, e.g., those from [6, 12], but simple tabulation is unique in its speed (like two multiplications in the experiments from [14]) and ease of implementation, making it a great choice in practice. For a more thorough comparison of simple tabulation with other hashing schemes, the reader is refered to [14].

2 Applications

Before proving our main results we describe two almost immediate applications.

2.1 Bloom Filters

Bloom filters were introduced by Bloom [3]. We will only discuss them briefly here and argue which guarantees are provided when implementing them using simple tabulation. For a thorough introduction including many applications see the survey by Broder and Mitzenmacher [4]. A Bloom filter is a simple data structure which space efficiently represents a set $X \subseteq U$ and supports membership queries of the form "is q in X". It uses k independent hash functions $h_0, \dots, h_{k-1} : U \to [m]$ and k arrays $A_0, \dots, A_{k-1}$ each of m bits which are initially all 0. For each $x \in X$ we calculate $(h_i(x))_{i \in [k]}$ and set the $h_i(x)$'th bit of A_i to 1 noting that a bit may

[2]Mitzenmacher and Vadhan actually estimate the probability of getting a false positive when using k-independent hashing for Bloom filters, but this error probability is strongly related to the expected number of non-empty bins $\mathbb{E}[|h(X)|]$ (in the fully random setting it *is* $\mathbb{E}[|h(X)|]/m$). Thus only a slight modification of their proof is needed.

be set to 1 several times. To answer the query "is q in X" we check if the bits corresponding to $(h_i(q))_{i\in[k]}$ are all 1, outputting "yes" if so and "no" otherwise. If $q \in X$ we will certainly output the correct answer but if $q \notin X$ we potentially get a false positive in the case that all the bits corresponding to $(h_i(q))_{i\in[k]}$ are set to 1 by other keys in X. In the case that $q \notin X$ the probability of getting a false positive is

$$\prod_{i=0}^{k-1} \Pr[h_i(q) \in h_i(X)],$$

which with fully random hashing is $p_0^k = (1-(1-1/m)^n)^k \approx (1-e^{-n/m})^k$.

It should be noted that Bloom filters are most commonly described in a related though not identical way. In this related setting we use a single (km)-bit array A and let $h_1,\dots,h_{k-1}: U \to [km]$, setting the bits of A corresponding to $(h_i(x))_{i\in[k]}$ to 1 for each $x \in X$. With fully random hashing the probability that a bit is set to 1 is then $q_0 := 1-\left(1-\frac{1}{km}\right)^{nk}$ and the probability of a false positive is thus at most $q_0^k = \left(1-\left(1-\frac{1}{km}\right)^{nk}\right)^k \le p_0^k$. Despite the difference, simple calculus shows that $p_0 - q_0 = O(1/m)$ and so

$$p_0^k - q_0^k = (p_0-q_0)\sum_{i=0}^{k-1} p_0^i q_0^{k-i-1} = O\left(\frac{kp_0^{k-1}}{m}\right).$$

In particular if $p_0 = 1-\Omega(1)$ or if the number of filters k is not too large (both being the case in practice) the failure probability in the two models are almost identical. We use the model with k different tables each of size m as this makes it very easy to estimate the error probability using Theorem 1.1 and the independence of the hash functions. We can in fact view h_i as a map from U to $[km]$ but having image in $[(i+1)n]\backslash[im]$ getting us to the model with just one array.

From Theorem 1.1 we immediately obtain the following corollary.

Corollary 2.1. *Let $X \subseteq U$ with $|X| = n$ and $y \in U\backslash X$. Suppose we represent X with a Bloom filter using k independent simple tabulation hash functions $h_0,\dots,h_{k-1}: U \to [m]$. The probability of getting a false positive when querying q is at most*

$$\left(p_0 + \frac{2n^{2-1/c}}{m^2}\right)^k.$$

At this point one can play with the parameters. In the fully random setting one can show that if the number of balls n and the the total number of bins km are fixed one needs to choose k and m such that $p_0 \approx 1/2$ in order to minimise the error probability (see [4]). For this, one needs $n \approx m\ln(2)$ and if m is chosen so, the probability above is at most $(p_0 + O(m^{-1/c}))^k$. In applications, k is normally a small number like 10 for a 0.1% false positive probability. In particular, $k = m^{o(1)}$, and then $(p_0 + O(m^{-1/c}))^k = p_0^k(1+o(1))$, asymptotically matching the fully random setting.

To resolve the issue that the range of a simple tabulation function has size 2^r but that we wish to choose $m \approx n/\ln(2)$, we choose r such that $2^r \ge m^2$ and use the combined hash function $s \circ h : U \to [m]$ described in Section 1.3. Now appealing to Theorem 1.6 instead of Theorem 1.1 we can again drive the false positive probability down to $p_0^k(1+o(1))$ when $k = m^{o(1)}$.

Alternatives: The argument by Mitzenmacher and Vadhan [11] discussed in relation to (1.5) actually yields a tight bound on the probability of a false positive when using ℓ-independent hashing for Bloom filters. We do not state their result here but mention that when ℓ is constant the error probability may again deviate by an additive constant from that of the fully random setting. It is also shown in [11] that if the input has enough entropy we can get the probability of a false positive to match that from the fully random setting asymptotically even using 2-independent hashing, yet it cannot be trusted for certain types of input.

Now, imagine you are a software engineer that wants to implement a Bloom filter, proportioning it for a desired low false-positive probability. You can go to a wikipedia page (`en.wikipedia.org/wiki/Bloom_filter`) or a texbook like [10] and read how to do it assuming full randomness. If you read [11], what do you do? Do you set $\ell = 2$ and cross your fingers, or do you pay the cost of a slower hash function with a larger ℓ, adjusting the false-positive probabilities accordingly? Which ℓ do you pick?

With our result, there are now hard choices. The answer is simple. We just have to add that everything works as stated for any possible input if the hashing is implemented with simple tabulation hashing (`en.wikipedia.org/wiki/Tabulation_hashing`) which is both very fast and very easy to implement.

2.2 Filter Hashing

In Filter hashing, as introduced by Fotakis et al. [7], we wish to store as many elements as possible of a set $X \subseteq U$ of size $|X| = n = m$ in d hash tables $(T_i)_{i\in[d]}$. The total number of entries in the tables is at most m and each entry can store just a single key. For $i \in [d]$ we pick independent hash functions $h_i : U \to [m_i]$ where m_i is the number of entries in T_i. The keys are allocated as follows: We first greedily store a key from $h_0^{-1}(\{y\})$ in $T_0[y]$ for each $y \in h_0(X)$. This lets us store exactly $|h_0(X)|$ keys. Letting S_0 be the so stored keys and $X_1 = X \backslash S_0$ the remaining keys, we repeat the process, storing $|h(X_1)|$ keys in T_1 using h_1 etc.

An alternative and in practice more relevant way to see this is to imagine that the keys arrive sequentially. When a new key x arrives we let i be the smallest index such that $T_i[h_i(x)]$ is unmatched and store x in that entry. If no such i exists the key is not stored. The name Filter hashing comes from this view which prompts the picture of particles (the keys) passing through filters (the tables) being caught by a filter only if there is a vacant spot.

The question is for a given $\varepsilon > 0$ how few filters that are needed in order to store all but at most εm keys with high probability. Note that the remaining εm keys can be stored using any hashing scheme which uses linear space, for example Cuckoo hashing with simple tabulation [13, 14], to get a total space usage of $(1 + O(\varepsilon))m$.

One can argue that with fully random hashing one needs $\Omega(\log^2(1/\varepsilon))$ filters to achieve that whp at least $(1 - \varepsilon)m$ keys are stored. To see that we can achieve this bound with simple tabulation we essentially proceed as in [7]. Let $\gamma > 0$ be any constant and choose $\delta > 0$ according to Theorem 1.3 so that if $X \subseteq U$ with $|X| = n$ and $h : U \to [m]$ is a simple tabulation hash function, then $|h(X)| \geq m(1 - (1 - 1/m)^{\delta n})$ with probability at least $1 - m^{-\gamma}$.

Let $m_0 = n$. For $i = 0, 1, \ldots$, we pick n_i to be the largest power of two below $\delta m_i / \log(1/\varepsilon)$. We then set $m_{i+1} = n - \sum_{j=0}^{i} n_j$, terminating when $m_{i+1} \leq \varepsilon n$. Then T_i is indexed by $(\log_2 n_i)$-bit strings — the range of a simple tabulation hash function h_i. Letting d be minimal such that $n_d \leq \varepsilon m$ we have that $(1 - \varepsilon)m \leq \sum_{i\in[d]} m_i \leq m$ and as n_i decreases by at least a factor of $\left(1 - \frac{\delta}{2\log(1/\varepsilon)}\right)$ in each step, $d \leq \lceil 2\log(1/\varepsilon)^2/\delta \rceil$.

How many bins of T_i get filled? Even if all bins from filters $(T_j)_{j<i}$ are non-empty we have at least n_i balls left and so with probability $1 - O(m_i^{-\gamma})$ the number of bins we hit is at least

$$m_i(1 - (1 - 1/m_i)^{\delta n_i}) \geq m_i(1 - e^{-\delta n_i/m_i}) \geq m_i(1 - \varepsilon).$$

Thus, with probability at least $1 - O(dm_d^{-\gamma})$, for each $i \in [d]$, filter i gets at least $(1 - \varepsilon)m_i$ balls. Since $\sum_{i\in[d]} m_i \geq (1 - \varepsilon)m$, the number of overflowing balls is at most $2\varepsilon m$ in this case.

Assuming for example that $\varepsilon = \Omega(m^{-1/2})$, as would be the case in most applications, we get that the fraction of balls not stored is $O(\varepsilon)$ with probability at least $1 - \tilde{O}(m^{-\gamma/2})$.

Alternatives The hashing scheme for Filter hashing described in [7] uses $(12\lceil \ln(4/\varepsilon) + 1 \rceil)$-independent polynomial hashing to achieve an overflow of at most εm balls. In particular the choice of hash functions depends on ε and becomes more unrealistic the smaller ε is. In contrast when using simple tabulation (which is only 3-independent) for Filter hashing we only need to change the number of filters, not the hashing, when ε varies. It should be mentioned that only $\lceil \ln(4/\varepsilon)^2 \rceil$ filters are needed for the result in [7] whereas we need a constant factor more. It can however be shown (we provide no details) that we can get down to $d = \lceil 2\log(1/\varepsilon)^2 \rceil$ filters by applying (1.2) of Theorem 1.4 if we settle for an error probability of $O(m^{-1+\eta})$ for a given constant $\eta > 0$.

Taking a step back we see the merits of a hashing scheme giving many complementary probabilistic guarantees. As shown by Pătraşcu and Thorup [14], Cuckoo hashing [13] implemented with simple tabulation succeeds with probability $1 - O(m^{-1/3})$ (for a recent simpler proof of this result, see Aamand et al. [1]). More precisely, for a set X' of n' balls, let m' be the least power of two bigger than $(1 + \Omega(1))n'$. Allocating tables T'_0, T'_1 of size m', and using simple tabulation hash functions $h'_0, h'_1 : U \to [m']$, with probability $1 - O(m^{-1/3})$ Cuckoo hashing succeeds in placing the keys such that every key $x \in X'$ is found in either $T'_0[h'_0(x)]$ or $T'_1[h'_1(x)]$. In case it fails, we just try again with new random h'_0, h'_1. We now use Cuckoo hashing to store the $m' = O(\varepsilon m)$ keys remaining after the filer hashing, appending the Cuckoo tables to the filter tables so that $T_{d+i} = T'_i$ and $h_{d+i} = h'_i$ for $i = 0, 1$. Then $x \in X$ if and only if for some $i \in [d + 2]$, we have $x = T_i[h_i(x)]$. We note that all these $d + 2$ lookups could be done in parallel. Moreover, as the output bits of simple tabulation are mutually independent, the $d + 2$ hash functions $h_i : U \to [2^{r_i}]$, $2^{r_i} = n_i$, can be implemented as a single simple tabulation hash function $h : U \to [2^{r_1 + \cdots + r_{d+2}}]$ and therefore all be calculated using just $c = O(1)$ look-ups in simple tabulation character tables.

3 Preliminaries

As in [16] we define a **position character** to be an element $(j, a) \in [c] \times \Sigma$. Simple tabulation hash functions are initially defined only on keys in U but we can extend the definition to sets of position characters $S = \{(i_j, a_j) : j \in [k]\}$ by letting $h(S) = \bigoplus_{j \in [k]} h_{i_j}(a_j)$. This coincides with $h(x)$ when the key $x \in U = [\Sigma]^c$ is viewed as the set of position characters $\{(i, x[i]) : i \in [c]\}$.

We start by describing an ordering of the position characters, introduced by Pătraşcu and Thorup [14] in order to prove that the number of balls hashing to a specific bin is Chernoff concentrated when using simple tabulation. If $X \subseteq U$ is a set of keys and $\prec$ is any ordering of the position characters $[c] \times \Sigma$ we for $\alpha \in [c] \times \Sigma$ define $X_\alpha = \{x \in X \mid \forall \beta \in [c] \times \Sigma : \beta \in x \Rightarrow \beta \preceq \alpha\}$. Here we view the keys as sets of position characters. Further define $G_\alpha = X_\alpha \backslash (\bigcup_{\beta \prec \alpha} X_\beta)$ to be the set of keys in X_α containing α as a position character. Pătraşcu and Thorup argued that the ordering may be chosen such that the groups G_α are not too large.

Lemma 3.1 (Pătraşcu and Thorup [14])**.** *Let $X \subseteq U$ with $|X| = n$. There exists an ordering $\prec$ of the position characters such that $|G_\alpha| \leq n^{1-1/c}$ for all position characters α. If q is any (query) key in X or outside X, we may choose the ordering such that the position characters of q are first in the order and such that $|G_\alpha| \leq 2n^{1-1/c}$ for all position characters α.*

Let us throughout this section assume that $\prec$ is chosen as to satisfy the properties of Lemma 3.1. A set $Y \subseteq U$ is said to be ***d*-bounded** if $|h^{-1}(\{z\}) \cap Y| \leq d$ for all $z \in R$. In other words no bin gets more than d balls from Y.

Lemma 3.2 (Pătraşcu and Thorup [14]). *Assume that the number of bins m is at least $n^{1-1/(2c)}$. For any constant γ, and $d = \min\left\{2c(3+\gamma)^c, 2^{2c(3+\gamma)}\right\}$ all groups G_α are d-bounded with probability at least $1 - m^{-\gamma}$.*

Lemma 3.2 follows from another lemma from [14] which we restate here as we will use it in one of our proofs.

Lemma 3.3 (Pătraşcu and Thorup [14]). *Let $\varepsilon > 0$ be a fixed constant and assume that $n \le m^{1-\varepsilon}$. For any constant γ no bin gets more than $\min\left(((1+\gamma)/\varepsilon)^c, 2^{(1+\gamma)/\varepsilon}\right) = O(1)$ balls with probability at least $1 - n^{-\gamma}$.*

Let us describe heuristically why we are interested in the order $\prec$ and its properties. We will think of h as being uncovered stepwise by fixing $h(\alpha)$ only when $(h(\beta))_{\beta \prec \alpha}$ has been fixed. At the point where $h(\alpha)$ is to be fixed the internal clustering of the keys in G_α has been settled and $h(\alpha)$ acts merely as a translation, that is, as a shift by an XOR with $h(\alpha)$. This viewpoint opens up for sequential analyses where for example it may be possible to calculate the probability of a bin becoming empty or to apply martingale concentration inequalities. The hurdle is that the internal clustering of the keys in the groups are not independent as the hash value of earlier position characters dictate how later groups cluster so we still have to come up with ways of dealing with these dependencies.

4 Proofs of main results

In order to pave the way for the proofs of our main results we start by stating two technical lemmas, namely Lemma 4.1 and 4.2 below. We provide proofs at the end of this section. Lemma 4.1 is hardly more than an observation. We include it as we will be using it repeatedly in the proofs of our main theorems.

Lemma 4.1. *Assume $\alpha \ge 1$ and $m, m_0 \ge 0$ are real numbers. Further assume that $0 \le g_1, \dots, g_k \le n_0$ and $\sum_{i=1}^k g_i = m$. Then*

$$\sum_{i=1}^{k} g_i^\alpha \le n_0^{\alpha-1} n. \tag{4.1}$$

If further $n_0 \le m$ for some real m then

$$\prod_{i=1}^{k}\left(1 - \frac{g_i}{m}\right) \ge \left(1 - \frac{n_0}{m}\right)^{n/n_0}. \tag{4.2}$$

In our applications of Lemma 4.1, $g_1, \dots, g_k$ will be the sizes of the groups G_α described in Lemma 3.1, and n_0 will be the upper bound on the group sizes provided by the same lemma.

For the second lemma we assume that the set of keys X has been partitioned into k groups $(X_i)_{i\in[k]}$. Let C_i denote the number of sets $\{x, y\} \subseteq X_i$ such that $x \ne y$ but $h(x) = h(y)$, that is, the number of pairs of colliding keys internal to X_i. Denote by $C = \sum_{i=1}^k C_i$ the total number of collisions internal in the groups. The second lemma bounds the expected value of C as well as its variance in the case where the groups are not too large.

Lemma 4.2. *Let $X \subseteq U$ with $|X| = n$ be partitioned as above. Suppose that there is an $n_0 \ge 1$ such that for all $i \in [k]$, $|X_i| \le n_0$. Then*

$$\mathbb{E}[C] \le \frac{n \cdot n_0}{2m}, \qquad \textit{and} \tag{4.3}$$

$$\mathrm{Var}[C] \le \frac{(3^c+1)n^2}{m} + \frac{n \cdot n_0^2}{m^2}. \tag{4.4}$$

For a given query ball $q \in U \backslash X$ and a bin $z \in [m]$, the upper bound on $\mathbb{E}[C]$ is also an upper bound on $\mathbb{E}[C \mid h(q) = z]$. For the variance estimate note that if in particular $n_0^2 = O(nm)$, then $\mathrm{Var}[C] = O(n^2/m)$.

We will apply this lemma when the X_i are the groups arising from the order $\prec$ of Lemma 3.1. With these results in hand we are ready to prove Theorem 1.1.

Proof of Theorem 1.1. Let us first prove the theorem in the case where y is a fixed bin not chosen dependently on the hash value of a query ball. If $n^{1-1/c} \geq m$ the result is trivial as then the stated upper bound is at least 1. Assume then that $n^{1-1/c} \leq m$. Consider the ordering $\alpha_1 \prec \cdots \prec \alpha_k$ of the position characters obtained from Lemma 3.1 such that all groups $G_i := G_{\alpha_i}$ have size at most $n^{1-1/c}$. We will denote by $n_0 := n^{1-1/c}$ the maximal possible group size.

We randomly fix the $h(\alpha_i)$ in the order obtained from $\prec$ not fixing $h(\alpha_i)$ before having fixed $h(\alpha_j)$ for all $j < i$. If $x \in G_i$ then $h(x) = h(\alpha_i) \oplus h(x \backslash \{\alpha_i\})$ and since $\beta \prec \alpha_i$ for all $\beta \in x \backslash \{\alpha_i\}$ only $h(\alpha_i)$ has to be fixed in order to settle $h(x)$. The number of different bins hit by the keys of G_i when fixing $h(\alpha_i)$ is thus exactly the size of the set $\{h(x \backslash \{\alpha_i\}) : x \in G_i\}$ which is simply translated by an XOR with $h(\alpha_i)$ and for $x \in G_i$ we have that $h(x)$ is uniform in its range when conditioned on the values $(h(\alpha_j))_{j<i}$.

To make it easier to calculate the probability that $y \in h(X)$ we introduce some *dummy balls*. At the point where we are to fix $h(\alpha_i)$ we dependently on $(h(\alpha_j))_{j<i}$ in any deterministic way choose a set $D_i \subseteq R = [m]$ of dummy balls, disjoint from $\{h(x \backslash \{\alpha_i\}) : x \in G_i\}$, such that $\{h(x \backslash \{\alpha_i\}) : x \in G_i\} \cup D_i$ has size exactly $|G_i|$. We will say that a bin z is *hit* if either $z \in h(X)$ or there exists an i such that $z = d \oplus h(\alpha_i)$ for some $d \in D_i$. In the latter case we will say that z is hit by a dummy ball. This modified random process can be seen as ensuring that when we are to finally fix the hash values of the elements of G_i by the last translation with $h(\alpha_i)$, we modify the group by adding dummy balls to ensure that exactly $|G_i|$ bins are hit by either a ball in G_i or a dummy ball in D_i. We let $D = \sum_{i=1}^k |D_i|$ denote the total number of dummy balls.

Let $\mathcal{H}$ denote the event that y is hit and $\mathcal{D}$ denote the event that y is hit by a dummy ball. With the presence of the dummy balls, $\Pr[\mathcal{H}]$ is easy to calculate:

$$\Pr[\mathcal{H}] = 1 - \prod_{i=1}^{k} \left(1 - \frac{|G_i|}{m}\right) \geq 1 - \prod_{i=1}^{k} \left(1 - \frac{1}{m}\right)^{|G_i|} = p_0.$$

Clearly $\Pr[y \in h(X)] \geq \Pr[\mathcal{H}] - \Pr[\mathcal{D}]$ so for a lower bound on $\Pr[y \in h(X)]$ it suffices to upper bound $\Pr[\mathcal{D}]$. Let $\mathcal{D}_i$ denote the event that y is hit by a dummy ball from D_i. We can calculate $\Pr[\mathcal{D}_i] = \sum_{\ell=0}^{\infty} \Pr[\mathcal{D}_i \mid |D_i| = \ell] \times \Pr[|D_i| = \ell]$. The conditional probability $\Pr[\mathcal{D}_i \mid |D_i| = \ell]$ is exactly ℓ/m as the choice of D_i only depends on the hash values $(h(\alpha_j))_{j<i}$ and when translated by an XOR with $h(\alpha_i)$ the bin y is hit with probability $|D_i|/m$. It follows that $\Pr[\mathcal{D}_i] = \mathbb{E}[|D_i|]/m$ and thus that $\Pr[\mathcal{D}] \leq \sum_{i=1}^k \Pr[\mathcal{D}_i] = \mathbb{E}[D]/m$. Finally the total number of dummy balls is upper bounded by the number C of internal collisions in the groups, so Lemma 4.2 gives that $\Pr[\mathcal{D}] \leq \mathbb{E}[C]/m \leq \frac{n^{2-1/c}}{2m^2}$. This gives the desired lower bound on p (throwing away the factor of 1/2, in order to simplify the statement in the theorem).

For the upper bound note that $\Pr[y \in h(X)] \leq \Pr[\mathcal{H}]$ so by Lemma 4.1

$$p \leq \Pr[\mathcal{H}] \leq 1 - \left(1 - \frac{n_0}{m}\right)^{n/n_0}.$$

Using the inequality $\left(1 + \frac{x}{\ell}\right)^{\ell} \geq e^x \left(1 - \frac{x^2}{\ell}\right)$ holding for $\ell \geq 1$ and $|x| \leq \ell$ with $x = -n/m$ and $\ell = n/n_0$ (note that $|x| \leq \ell$ as we assumed that $n^{1-1/c} \leq m$) we obtain that

$$p \leq 1 - e^{-n/m}\left(1 - \frac{n \cdot n_0}{m^2}\right) \leq 1 - e^{-n/m} + \frac{n \cdot n_0}{m^2} \leq p_0 + \frac{n^{2-1/c}}{m^2},$$

as desired. The bound on $\mathbb{E}[|h(X)|]$ follows immediately as $\mathbb{E}[|h(X)|] = \sum_{y\in[m]} \Pr[y \in h(X)]$.

Finally consider the case where y is chosen conditioned on $h(q) = z$ for a query ball $q \notin X$ and a bin z. Here we may assume that $2n^{1-1/c} \le m$ as otherwise the claimed upper bound is at least 1. We choose the ordering $\prec$ such that the position characters of q are first in the order and such that all groups have size at most $2n^{1-1/c}$ which is possible by Lemma 3.1. Let $n_0 = \min(n, 2n^{1-1/c})$ denote the maximal possible group size. Introducing dummy balls the same way as before and repeating the arguments, the probability of the event $\mathcal{H}$ that y is hit satisfies

$$p_0 \le \Pr[\mathcal{H} \mid h(q) = z] \le 1 - \left(1 - \frac{n_0}{m}\right)^{n/n_0} \le 1 - e^{-n/m}\left(1 - \frac{n \cdot n_0}{m^2}\right) \le p_0 + \frac{2n^{2-1/c}}{m^2}.$$

The desired upper bound follows immediately as $\Pr[y \in h(X) \mid h(q) = z] \le \Pr[\mathcal{H} \mid h(q) = z]$. For the lower bound we again let $\mathcal{D}$ denote the event that y is hit by a dummy ball and $\mathcal{D}_i$ denote the event that y is hit by a dummy ball from D_i. Then

$$\Pr[\mathcal{D}_i \mid h(q) = z] = \sum_{\ell=0}^{\infty} \Pr[\mathcal{D}_i \mid h(q) = z \wedge |D_i| = \ell] \times \Pr[|D_i| = \ell \mid h(q) = z].$$

As before we have that $\Pr[\mathcal{D}_i \mid h(q) = z \wedge |D_i| = \ell] = \ell/m$ since the hash values of the position characters of q are fixed before $h(\alpha_i)$. Thus,

$$\Pr[\mathcal{D}_i \mid h(q) = z] = \sum_{\ell=0}^{\infty} \frac{\ell}{m} \Pr[|D_i| = \ell \mid h(q) = z] = \frac{\mathbb{E}[|D_i| \mid h(q) = z]}{m},$$

and another union bound gives that

$$\Pr[\mathcal{D} \mid h(q) = z] \le \frac{\mathbb{E}[D \mid h(q) = z]}{m} \le \frac{\mathbb{E}[C \mid h(q) = z]}{m} \le \frac{n^{2-1/c}}{m^2},$$

where we in the last step used Lemma 4.2. □

We are now going to prove Theorem 1.3. We start out by recalling Azuma's inequality.

Theorem 4.3 (Azuma's inequality [2]). *Suppose $(X_i)_{i=0}^k$ is a martingale satisfying that $|X_i - X_{i-1}| \le s_i$ almost surely for all $i = 1, \ldots, k$. Let $s = \sum_{i=1}^k s_i^2$. Then for any $t \ge 0$ it holds that*

$$\Pr(X_k \ge X_0 + t) \le \exp\left(\frac{-t^2}{2s}\right), \quad \textit{and} \quad \Pr(X_k \le X_0 - t) \le \exp\left(\frac{-t^2}{2s}\right).$$

To apply Azuma's inequality we need to recall a little measure theory. Suppose $(\Omega, \mathcal{F}, \Pr)$ is a finite measure space (that is Ω is finite), and that $Y : \Omega \to \mathbb{R}$ is an $\mathcal{F}$-measurable random variable. A sequence of σ-algebras $(\mathcal{F}_i)_{i=1}^k$ on Ω is called a *filter* of the σ-algebra $\mathcal{F}$ if $\{\emptyset, \Omega\} = \mathcal{F}_0 \subseteq \cdots \subseteq \mathcal{F}_k = \mathcal{F}$. Defining $Y_i = \mathbb{E}[Y|\mathcal{F}_i]$, the sequence $(Y_i)_{i=0}^k$ becomes a martingale with $Y_0 = \mathbb{E}[Y]$ and $Y_k = Y$. It is for such martingales that we will apply Azuma's inequality.

Proof of Theorem 1.3. By the result by Pătraşcu and Thorup [14] we may assume that $n \le Cm \log m$ for some constant C as otherwise all bins are full whp from which the results of the theorem immediately follow.

Let $G_1, \ldots, G_k$ be the groups described in Lemma 3.1 and $\alpha_1, \ldots, \alpha_k$ be the corresponding position characters. Again we think of the $h(\alpha_i)$ as being fixed sequentially. We let $(\Omega, \mathcal{F}, \Pr)$ be the underlying probability space when choosing h, that is, Ω is the set of all simple tabulation hash functions, $\mathcal{F} = \mathcal{P}(\Omega)$, and Pr is the uniform probability measure on Ω. For $i = 0, \ldots, k$ we define $\mathcal{F}_i = \sigma(h(\alpha_1), \dots, h(\alpha_i))$ to be the σ-algebra generated by the hash values of the first i position characters. Then $\{\emptyset, \Omega\} = \mathcal{F}_0 \subseteq \cdots \subseteq \mathcal{F}_k = \mathcal{F}$ is a filter of $\mathcal{F}$.

Ideally we would hope that for the martingale $(X_i)_{i=0}^k = (\mathbb{E}[|h(X)| \mid \mathcal{F}_i])_{i=0}^k$ we could effectively bound $|X_i - X_{i-1}|$ and thus apply Azuma's inequality. This is however too much to hope for — the example with keys $[2] \times [n/2]$ shows that the hash value of a single position character can have a drastic effect on the conditionally expected number of non-empty bins. To remedy this we will again be using dummy balls but this time in a different way.

First of all, we let $\gamma > 0$ be any constant. Since $n \le Cm \log m$, Lemma 3.2 gives that there exists a $d = d(\gamma) = O(1)$ such that all groups are d-bounded with probability at least $1 - m^{-\gamma}$. Here is how we use the dummy balls: After having fixed $(h(\alpha_j))_{j \prec i}$ we again look at the set $G'_i := \{h(x \backslash \{\alpha_i\}) : x \in G_i\}$ letting $I^- = \{i \in [k] : |G'_i| \ge \lceil |G_i|/d\ \rceil\}$ and $I^+ = \{i \in [k] : |G'_i| < \lceil |G_i|/d\ \rceil\}$. For $i \in I^-$ we dependently on $(h(\alpha_j))_{j \prec i}$ choose a set $D_i^- \subseteq G'_i$ such that $|G'_i \backslash D_i^-| = \lceil |G_i|/d\ \rceil$. Similarly we for $i \in I^+$ choose a set $D_i^+ \subseteq R$ disjoint from G'_i such that $|G'_i \cup D_i^+| = \lceil |G_i|/d\ \rceil$. We say that bin z is *hit* if there exists an i such that either

1. $i \in I^-$ and $z = y \oplus h(\alpha_i)$ for some $y \in G'_i \backslash D_i^-$, or

2. $i \in I^+$ and $z = y \oplus h(\alpha_i)$ for some $y \in G'_i \cup D_i^+$.

This modified random process obtained by adding balls if $|G'_i|$ is too large and removing balls if it is too small can be seen as ensuring that when we are to finally fix the hash values of the elements of G_i by the last translation by $h(\alpha_i)$ we first modify the group to ensure that we hit *exactly* $\lceil |G_i|/d\ \rceil$ bins.

Importantly, we observe that if G_i is d-bounded then $|G'_i| \ge |G_i|/d$ and since $|G'_i|$ is integral $|G'_i| \ge \lceil |G_i|/d\ \rceil$. Thus if all groups are d-bounded $I^+ = \emptyset$, and no dummy balls are added.

Letting H denote the number of bins hit, we have that

$$\mathbb{E}[H] = m\left(1 - \prod_{i=1}^{k}\left(1 - \frac{\lceil |G_i|/d\ \rceil}{m}\right)\right) \ge m\left(1 - \prod_{i=1}^{k}\left(1 - \frac{1}{m}\right)^{\lceil |G_i|/d\ \rceil}\right) \ge m\left(1 - \left(1 - \frac{1}{m}\right)^{n/d}\right).$$

We now wish to apply Azuma's inequality to the martingale $(H_i)_{i=0}^k = (\mathbb{E}[H \mid \mathcal{F}_i])_{i=0}^k$. To do this we require a good upper bound on $|H_i - H_{i-1}|$ and we claim that in fact $|H_i - H_{i-1}| \le |G_i|$. To see this, let the random variable N_i denote the number of bins not hit when the hash values of the first i position characters has been settled. Then $H_i = m - N_i \prod_{j>i}\left(1 - \frac{\lceil |G_j|/d \rceil}{m}\right)$ and so

$$|H_i - H_{i-1}| = \prod_{j>i}\left(1 - \frac{\lceil |G_j|/d \rceil}{m}\right)\left|N_i - \left(1 - \frac{\lceil |G_i|/d \rceil}{m}\right)N_{i-1}\right| \le \left|N_i - N_{i-1} + \frac{\lceil |G_i|/d \rceil \cdot N_{i-1}}{m}\right|.$$

Now $N_{i-1} - \lceil |G_i|/d \rceil \le N_i \le N_{i-1}$ as at least 0 and most $\lceil |G_i|/d \rceil$ bins are hit after fixing $h(\alpha_i)$ and from this it follows that $|H_i - H_{i-1}| \le \lceil |G_i|/d \rceil \le |G_i|$.

Letting $s_i = |G_i|$ we have that $\sum_{i=1}^k s_i^2 \le n^{2-1/c}$ by Lemma 4.1 and thus we can apply Azuma's inequality to obtain that

$$\Pr(H \le \mathbb{E}[H] - t) \le \exp\left(\frac{-t^2}{2n^{2-1/c}}\right).$$

Putting $t = \sqrt{\gamma \cdot 2n^{2-1/c} \log m}$ we obtain that with probability at least $1 - m^{-\gamma}$

$$H \ge \ell(n, m) := m\left(1 - \left(1 - \frac{1}{m}\right)^{n/d}\right) - \sqrt{\gamma \cdot 2n^{2-1/c} \log m}.$$

As $I^+ = \emptyset$ with probability at least $1 - m^{-\gamma}$ and as we in this case have that $|h(X)| \geq H$ we have that $|h(X)| \geq \ell(n,m)$ with probability at least $1 - 2n^{-\gamma}$.

The remaining part of proof is just combining what we have together with a little calculus! We first consider the case $n \leq m$. In this case the lower bound simply states that $|h(X)| = \Omega(n)$. To see that this bound holds observe that if (for example) $n \leq m^{1/2}$ then by Lemma 3.3 no bin gets more than a constant number of balls with probability at least $1 - m^{-\gamma}$. In particular $|h(X)| = \Omega(n)$ with probability at least $1 - m^{-\gamma}$. If on the other hand $n \geq m^{1/2}$ then $\sqrt{\gamma \cdot 2n^{2-1/c} \log m} = o(n)$ and $\ell(n,m) = \Omega(n) - o(n) = \Omega(n)$ which again gives the desired result.

Finally suppose $n \geq m$. Let $\alpha := (1 - 1/m)^{m/(2d)} \leq e^{-1/(2d)}$ and let β be a constant so large that $\beta \geq 2d$ and $m\,(1 - 1/m)^{n/\beta} \geq \frac{1}{1-\alpha}\sqrt{\gamma \cdot 2n^{2-1/c}\log m}$, the last requirement being possible as we assumed $n \leq Cm \log m$. Then

$$\begin{aligned}\frac{\ell(n,m)}{m} &\geq 1 - \left(1 - \frac{1}{m}\right)^{n/d} - (1-\alpha)\left(1 - \frac{1}{m}\right)^{n/\beta} \\ &\geq 1 - \left(1 - \frac{1}{m}\right)^{n/\beta}\left(\left(1 - \frac{1}{m}\right)^{n/(2d)} + (1-\alpha)\right) \geq 1 - \left(1 - \frac{1}{m}\right)^{n/\beta}.\end{aligned}$$

Since $|h(X)| \geq \ell(n,m)$ with probability at least $1 - 2n^{-\gamma}$ this gives the desired result. □

We now prove Theorem 1.4.

Proof of Theorem 1.4. When $n^{1-1/(2c)} \geq m$ the probability bounds of the theorem are trivial since they are $\Omega(1)$ when $t \leq n$. We therefore assume henceforth that $n^{1-1/(2c)} \leq m$.

Again consider the order $\prec$ obtained from Lemma 3.1 such that for all i we have $|G_i| \leq n^{1-1/c}$. We again think of the hash values of the position characters as being fixed in the order obtained from $\prec$. We also introduce dummy balls in exactly the same way as we did in the proof of Theorem 1.1 using the same definition of a bin being hit.

Letting H denote the number of bins hit (by an $x \in X$ or a dummy ball) we have that $\mathbb{E}[H] = m\left(1 - \prod_{i=1}^{k}\left(1 - \frac{|G_i|}{m}\right)\right)$, like in the proof of Theorem 1.1, and

$$\mu_0 \leq \mathbb{E}[H] \leq \mu_0 + \frac{n^{2-1/c}}{m}.$$

Furthermore letting $\mathcal{F}_i = \sigma(h(\alpha_1), \ldots, h(\alpha_i))$ be the σ-algebra generated by $(h(\alpha_j))_{j \leq i}$, the same argument as in the proof of Theorem 1.3 gives that $H_i = \mathbb{E}[H|\mathcal{F}_i]$ is a martingale satisfying that $|H_i - H_{i-1}| \leq |G_i|$ for all i. We can thus apply Azuma's inequality with $s_i = |G_i|$ and $s = \sum_{i=1}^{k} s_i^2 \leq n^{2-1/c}$ (here we used Lemma 4.1) to obtain that

$$\Pr[H \geq \mathbb{E}[H] + t] \leq \exp\left(\frac{-t^2}{2n^{2-1/c}}\right), \quad \text{and} \tag{4.5}$$

$$\Pr[H \leq \mathbb{E}[H] - t] \leq \exp\left(\frac{-t^2}{2n^{2-1/c}}\right). \tag{4.6}$$

We now wish to translate this concentration result on the number of bins hit when the dummy balls are included to a concentration result on $|h(X)|$. We begin with the bound in (1.1). As $|h(X)| \leq H$ it suffices to bound the probability $\Pr[H \geq \mu_0 + 2t]$. Since $\mathbb{E}[H] \leq \mu_0 + \frac{n^{2-1/c}}{m}$,

$$\Pr[H \geq \mu_0 + 2t] \leq \Pr\left[H - \mathbb{E}[H] \geq 2t - \frac{n^{2-1/c}}{m}\right],$$

so when $t \geq \frac{n^{2-1/c}}{m}$ the result follows immediately from (4.5). If on the other hand $t < \frac{n^{2-1/c}}{m}$ then $\frac{t^2}{n^{2-1/c}} < \frac{n^{2-1/c}}{m^2} \leq 1$ and the result is trivial as the right hans size in (1.1) can be as large as $\Omega(1)$ which is a valid upper bound on any probability.

We now turn to the proof of (1.2). Letting $\mathcal{E}$ denote the event that $|h(X)| \leq \mu_0 - 2t$ and $\mathcal{A}$ the event that $H \leq \mu_0 - t$ we have that

$$\Pr[|h(X)| \leq \mu_0 - 2t] = \Pr[\mathcal{E}] \leq \Pr[\mathcal{A}] + \Pr[\mathcal{E} \wedge \neg\mathcal{A}].$$

By (4.6) and since $\mu_0 \leq \mathbb{E}[H]$ we can upper bound $\Pr[\mathcal{A}] \leq \exp\left(\frac{-t^2}{2n^{2-1/c}}\right)$. For the other term we note that $\mathcal{E} \wedge \neg\mathcal{A}$ entails that at least t bins are hit by a dummy ball. In particular the number of dummy balls is at least t. As the number C of internal collisions of the groups is an upper bound on the number of dummy balls this in turn implies, $t \leq C$. We may assume that $t \geq n^{1-1/(2c)}$ as otherwise (1.1) is trivial. As we assumed, $n^{1-1/(2c)} \leq m$ it follows from Lemma 4.2 that $\mathbb{E}[C] \leq \frac{n^{2-1/c}}{2m} \leq \frac{tn^{1-1/(2c)}}{2m} \leq t/2$ and so $t - \mathbb{E}[C] \geq t/2$. Applying Chebychev's inequality as well as (4.4) of Lemma 4.2 we thus obtain,

$$\Pr[\mathcal{E} \wedge \neg\mathcal{A}] \leq \Pr[C \geq t] \leq \Pr[C - \mathbb{E}[C] \geq t/2] \leq \frac{4\operatorname{Var}[C]}{t^2} = O\left(\frac{n^2}{t^2 m}\right).$$

Combining the two bounds completes the proof. □

We promised to argue why we cannot dispose with the term $n^2/(mt^2)$ in general. Suppose that $n = O(m)$ and let $t = n^{1/2+\alpha}$ for an $\alpha \in [1/2, 1)$ such that $n^{1/2+\alpha} \in [\sqrt{n}, n/2]$, and consider the set of keys $[n/t] \times [t]$. With probability $\Omega((n/t)^2/m) = \Omega(n^2/(t^2m))$ we have that $h_0(a_0) = h_0(a_1)$ for two distinct $a_0, a_1 \in [n/t]$. Conditioned on this event the expected number of non-empty bins is at most $m\left(1 - \left(1 - \frac{n/t-1}{m}\right)^t\right)$ which can be shown to be $\mu_0 - \Omega(t)$ by standard calculus. The additive term $\Omega(t)$ comes from the fact that the t pairs of colliding keys $\{(a_0, b), (a_0, b)\}_{b\in[t]}$ causes the expected number of non-empty bins to decrease by $\Omega(t)$ when $n = O(m)$. Thus the deviation by $\Omega(t)$ from μ_0 occurs with probability $\Omega(n^2/(mt^2))$.

We will now set the stage for the proof of Theorem 1.5. As mentioned in the introduction we require a stronger martingale inequality than that by Azuma. The one we use is due to Mcdiarmid [9]. Again assume that $(\Omega, \mathcal{F}, \Pr)$ is a finite probability space, that $X : \Omega \to \mathbb{R}$ is an $\mathcal{F}$-measurable random variable, that $(\mathcal{F}_i)_{i=0}^k$ is a filter of $\mathcal{F}$, and that $X_i = \mathbb{E}[X \mid \mathcal{F}_i]$. Also recall the definition of conditional variance: If $\mathcal{G} \subseteq \mathcal{F}$ is a σ-algebra, then $\operatorname{Var}[X \mid \mathcal{G}] = \mathbb{E}[(X - \mathbb{E}[X \mid \mathcal{G}])^2 \mid \mathcal{G}] = \mathbb{E}[X^2 \mid \mathcal{G}] - \mathbb{E}[X \mid \mathcal{G}]^2$.

Theorem 4.4 (Mcdiarmid [9]). *Assume that* $\operatorname{Var}[X_i \mid \mathcal{F}_{i-1}] \leq \sigma_i^2$ *for* $i = 1, \ldots, k$ *and further that* $X_i - X_{i-1} \leq M$ *for* $i = 1, \ldots, k$. *Then for* $t \geq 0$,

$$\Pr[X - \mathbb{E}[X] \geq t] \leq \exp\left(\frac{-t^2}{2\left(\sum_{i=1}^k \sigma_i^2 + Mt/3\right)}\right).$$

With this tool in hand we are ready to prove Theorem 1.5, the main technical challenge being to argue why we can apply Theorem 4.4.

Proof of Theorem 1.5. We introduce dummy balls exactly in the proof of Theorem 1.4 and Theorem 1.1 and consider the same martingale $(H_i)_{i=0}^k = (\mathbb{E}[H|\mathcal{F}_i])_{i=0}^k$, where H is the number of bins hit (by either a dummy ball or a ball from X). We already saw that $|H_i - H_{i-1}| \le |G_i| \le n^{1-1/c}$, so we let $M := n^{1-1/c}$. What remains is to upper bound $\mathrm{Var}[H_i \mid \mathcal{F}_{i-1}]$. First note that

$$\begin{aligned}\mathrm{Var}[H_i \mid \mathcal{F}_{i-1}] =& \mathbb{E}[(H_i - \mathbb{E}[H_i \mid \mathcal{F}_{i-1}])^2 \mid \mathcal{F}_{i-1}] \\ =& \mathbb{E}[(H_i - H_{i-1})^2 \mid \mathcal{F}_{i-1}].\end{aligned}$$

We denote by N_i the number of bins that are empty after the hash values of the first i position characters has been settled. Then, by the same reasoning as in the proof of Theorem 1.3, we have that

$$|H_i - H_{i-1}| \le \left| N_i - N_{i-1} + \frac{|G_i| \cdot N_{i-1}}{m} \right|.$$

Now let $T_i = |G_i| - N_{i-1} + N_i$ denote the number of bins hit in the i'th step that were already hit in the $(i-1)$'st step. As $\mathbb{E}[T_i \mid \mathcal{F}_{i-1}] = (m - N_{i-1})|G_i|/m$, the above inequality reads

$$|H_i - H_{i-1}| \le |T_i - \mathbb{E}[T_i \mid \mathcal{F}_{i-1}]|,$$

and so,

$$\mathrm{Var}[H_i \mid \mathcal{F}_{i-1}] \le \mathrm{Var}[T_i \mid \mathcal{F}_{i-1}] \le \mathbb{E}[T_i^2 \mid \mathcal{F}_{i-1}].$$

Now, T_i^2 counts the number of 2-tuples (y, z) with $y, z \in \{h(x \backslash \{\alpha_i\}) : x \in G_i\} \cup D_i$ such that $h(y)$ and $h(z)$ are already hit after the $(i-1)$'st step. Conditioned on $\mathcal{F}_{i-1}$ the probability that this occurs for a given such pair is at most $\frac{m - N_{i-1}}{m} \le \frac{n}{m}$, and there are exactly $|G_i|^2$ such pairs. Hence

$$\mathrm{Var}[H_i \mid \mathcal{F}_{i-1}] \le \frac{n}{m}|G_i|^2 := \sigma_i^2.$$

By Lemma 4.2, $\sum_{i=1}^k \sigma_i^2 \le \frac{n^{3-1/c}}{m}$ so Theorem 4.4 gives that for $t \ge 0$

$$\begin{aligned}\Pr[H - \mathbb{E}[H] \ge t] &\le \exp\left(\frac{-t^2}{2\left(\frac{n^{3-1/c}}{m} + n^{1-1/c}t/3\right)}\right) \qquad (4.7) \\ &\le \exp\left(-\min\left\{\frac{t^2}{4\frac{n^{3-1/c}}{m}}, \frac{3t}{2n^{1-1/c}}\right\}\right).\end{aligned}$$

As $\mathbb{E}[H] \le \mu_0 + \frac{n^{2-1/c}}{m}$ this is also an upper bound on $\Pr\left[|h(X)| \ge \mu_0 + t + \frac{n^{2-1/c}}{m}\right]$. Now the same argument as in the proof of Theorem 1.4 leads to the upper bound (1.3).

Finally, to prove (1.4) we use the same strategy as above but this time we define $H' = -H$ and the martingale $(H'_i)_{i=0}^k = (\mathbb{E}[H' \mid \mathcal{F}_i])_{i=0}^k$. Then $|H'_i - H'_{i-1}| = |H_i - H_{i-1}| \le M$ and $\mathrm{Var}[H'_i \mid \mathcal{F}_{i-1}] = \mathrm{Var}[H_i \mid \mathcal{F}_{i-1}]$ for $i = 1, \dots, k$, so we get a bound as in (4.7), but this time on $\Pr[H' - \mathbb{E}[H'] \ge t] = \Pr[H - \mathbb{E}[H] \le -t]$.

As in the proof of Theorem 1.4, the event $|h(X)| \le \mu_0 - t$ implies that either $\mathcal{A}$: $H - \mathbb{E}[H] \le -t/2$ or $\mathcal{B}$: the number of internal collisions C is at least $t/2$. $\Pr[\mathcal{A}]$ is bounded using (4.7), giving us the first term of the bound in (1.4). For $\Pr[\mathcal{B}]$, note that we may assume that $t \ge 4n^{1-1/c}$ as otherwise (4.7) is trivial. In that case $\mathbb{E}[C] \le n^{2-1/c}/m \le \frac{t}{4}\frac{n}{m} \le \frac{t}{4}$, so $t/2 - \mathbb{E}[C] \ge t/4$. Lemma 4.2 thus gives that $\Pr[\mathcal{B}] = O(\frac{n^2}{mt^2})$ — the second term in the bound (1.4). The proof is complete. $\square$

4.1 Proofs of technical lemmas

For proving Lemma 4.2 and Lemma 4.1 we need to briefly discuss the independence of simple tabulation. In the notion of k-independence introduced by Wegman and Carter [20] simple tabulation is only 3-independent as shown by the set of keys $S = \{(a_0, b_0), (a_0, b_1), (a_1, b_0)(a_1, b_1)\}$. Indeed $\bigoplus_{x \in S} h(x) = 0$ showing that the keys do not hash independently. The issue is that since each position character appears an even number of times in S the addition over $\mathbb{Z}_2$ causes the terms to cancel out. This property in a sense characterises dependencies of keys as shown by Thorup and Zhang [19]

Lemma 4.5 (Thorup and Zhang [19])**.** *The keys $x_1, \ldots, x_k \in U$ are dependent if and only if there exists a non-empty subset $I \subseteq \{1, \ldots, k\}$ such that each position character in $(x_i)_{i \in I}$ appears an even number of times. In this case we have that $\bigoplus_{i \in I} h(x_i) = 0$.*

For keys $x, y \in U$ we write $x \oplus y$ for the symmetric difference of x and y when viewed as sets of position characters. Then the property that each position character appearing an even number of times in $(x_i)_{i \in I}$ can be written as $\bigoplus_{i \in I} x_i = \emptyset$. As shown by Dahlgaard et al. [5] we can efficiently bound the number of such tuples $(x_i)_{i \in I}$.

Lemma 4.6 (Dahlgaard et al. [5])**.** *Let $A_1, \ldots, A_{2t} \subseteq U$. The number of $2t$-tuples $(x_1, \ldots, x_{2t}) \in A_1 \times \cdots \times A_{2t}$ such that $x_1 \oplus \cdots \oplus x_{2t} = \emptyset$ is at most $((2t-1)!!)^c \prod_{i=1}^{2t} \sqrt{|A_i|}$. Here $a!!$ denotes the product of all the positive integers in $\{1, \ldots, a\}$ having the same parity as a.*

We now provide the proofs of Lemma 4.2 and Lemma 4.1. Since we need Lemma 4.1 in the proof of Lemma 4.2 we prove that first.

Proof of Lemma 4.1. We prove the following more general statement: Let $f : [0, n_0] \to \mathbb{R}$ be convex with $f(0) = 0$. Let $0 \le g_1, \ldots, g_k \le n_0$ be such that $m = \sum_{i=1}^k g_i$. Define $S := \sum_{i=1}^k f(g_i)$. Then $S \le (n/n_0) f(n_0)$.

To see why the statement holds note that by convexity, $f(x) + f(y) \le f(x-t) + f(y+t)$ if $0 \le t \le x \le y \le n_0 - t$. To maximize S we thus have to set $k = \lceil n/\ n_0 \rceil$, $g_1 = \cdots = g_{k-1} = n_0$ and $g_k = n - \sum_{i=1}^{k-1} g_i = \varepsilon n_0$, where $\varepsilon \in [0, 1)$. It follows that

$$S \le \left(\frac{n}{n_0} - \varepsilon \right) f(n_0) + f(\varepsilon n_0).$$

Finally $f(\varepsilon n_0) \le \varepsilon f(n_0)$ using convexity and that $f(0) = 0$, so $S \le (n/n_0) f(n_0)$ as desired.

The first inequality (4.1) of the lemma follows immediately from the above statement with $f(x) = x^\alpha$ which is convex since $\alpha \ge 1$. For inequality (4.2) we may assume that $m > n_0$ as the result is trivial when $m = n_0$. We then define $f(x) = -\log(1 - x/n)$ which is convex with $f(0) = 0$. Then

$$S = -\sum_{i=1}^k \log\left(1 - \frac{g_i}{n}\right) \le -\frac{n}{n_0} \log\left(1 - \frac{n}{m}\right),$$

which upon exponentiation leads to inequality (4.2). □

Proof of Lemma 4.2. We define $g_i = |X_i|$ for $i \in [k]$. Now (4.3) is easily checked. Indeed, since simple tabulation is 2-independent,

$$\mathbb{E}[C] = \sum_{i=1}^k \binom{g_i}{2} \frac{1}{m} \le \frac{1}{2m} \sum_{i=1}^t g_i^2 \le \frac{n \cdot n_0}{2m},$$

where in the last step we used Lemma 4.1. The last statement of the lemma concerning $\mathbb{E}[C \mid h(q) = z]$ follows from the same argument this time however using that simple tabulation is 3-independent.

We now turn to (4.4). Writing $\mathrm{Var}[C] = \mathbb{E}[C^2] - (\mathbb{E}[C])^2$ our aim is to bound $\mathbb{E}[C^2]$. Note that C^2 counts the number of tuples $(\{x,y\},\{z,w\})$ such that $x \neq y$ and $z \neq w$ but $h(x) = h(y)$ and $h(z) = h(w)$ and furthermore $x, y \in G_i$ and $z, w \in G_j$ for some $i, j \in [k]$. We denote the set of such tuples T and for $\tau = (\{x,y\},\{z,w\}) \in T$ we let X_τ be the indicator for the event that both $h(x) = h(y)$ and $h(z) = h(w)$. Then

$$\mathbb{E}[C^2] = \sum_{\tau \in T} \Pr(X_\tau = 1). \tag{4.8}$$

We now partition T by letting

- T_1 be the elements of T for which $\{x,y\} = \{z,w\}$.
- T_2 be the elements of T for which $|\{x,y,z,w\}| = 3$.
- T_3 be the elements of T for which x,y,z,w are distinct and independent.
- T_4 be the elements of T for which x,y,z,w are distinct and dependent and there is an $i \in [k]$ such that $x,y,z,w \in G_i$.
- T_5 be the remaining elements of T, that is, those element $(\{x,y\},\{z,w\})$ such that x,y,z,w are distinct and dependent and such that $\{x,y\} \subseteq G_i$ and $\{z,w\} \subseteq G_j$ for some distinct $i,j \in [k]$.

Putting $S_j = \sum_{\tau \in T_j} \Pr(X_\tau = 1)$ the sum in (4.8) can be written as $\sum_{j=1}^{5} S_j$ and we can efficiently upper bound each of the inner sums as we now show. Clearly,

$$S_1 = \sum_{i=1}^{k} \binom{g_i}{2} \frac{1}{m} = \mathbb{E}[C].$$

For the second sum we use that simple tabulation is 3-independent and that $|\{x,y,z,w\}| = 3$ implies that x,y,z,w belongs to the same group G_i for some $i \in [k]$. Hence

$$S_2 = \sum_{i=1}^{k} \binom{g_i}{2} \cdot 2 \cdot \binom{g_i - 2}{1} \frac{1}{m^2} \leq \frac{1}{m^2} \sum_{i=1}^{k} g_i^3 \leq \frac{n \cdot n_0^2}{m^2},$$

again using Lemma 4.1 to bound the sum of cubes. Finally we upper bound S_3 as

$$S_3 \leq \frac{1}{m^2} \left(\sum_{i=1}^{k} \binom{g_i}{2} \binom{g_i - 2}{2} + \sum_{i,j \in [k]: i \neq j} \binom{g_i}{2} \binom{g_j}{2} \right) \leq \frac{1}{m^2} \left(\sum_{i=1}^{k} \binom{g_i}{2} \right)^2 = \mathbb{E}[C]^2.$$

Note that in the first three steps we have not been using anything about simple tabulation except it being 3-independent. However, if $(\{x,y\},\{z,w\}) \in T_4 \cup T_5$ then by Lemma 4.5 we have that $x \oplus y \oplus z \oplus w = \emptyset$ and thus that $h(x) = h(y)$ exactly if $h(z) = h(w)$ which happens with probability m^{-1}. Thus in this case we have to efficiently bound the sizes of T_4 and T_5. Luckily Lemma 4.6 comes to our rescue and we can bound

$$S_4 + S_5 \leq \frac{3^c}{m} \left(\sum_{i=1}^{k} g_i^2 + \sum_{i,j \in [k]: i \neq j} g_i g_j \right) = \frac{3^c}{m} \left(\sum_{i=1}^{k} g_i \right)^2 = \frac{3^c n^2}{m}.$$

Combining all this we find that

$$\mathrm{Var}[C] = \mathbb{E}[C^2] - \mathbb{E}[C]^2 \le \mathbb{E}[C] + \frac{n \cdot n_0^2}{m^2} + \frac{3^c n^2}{m} \le \frac{n \cdot n_0^2}{m^2} + \frac{(3^c+1)n^2}{m},$$

as desired. $\square$

5 Handling bins consisting of many subbins

In this section we show how to modify the proof of Theorem 1.1 to obtain Theorem 1.6.

Proof of Theorem 1.6. We may assume that $\rho n^{1-1/c} \le 1$ as otherwise the result is trivial.

As usual we consider the ordering on the position characters, $\alpha_1 \prec \cdots \prec \alpha_k$, obtained from Lemma 3.1, and we fix the values $h(\alpha_i)$ in this order. Suppose that $(h(\alpha_j))_{j<i}$ are fixed and let $V_i = \{y \in [2^r] \mid \exists x \in G_i : h(x \backslash \{\alpha_i\}) + y \in S\}$ denote those hash values $h(\alpha_i)$ that would cause $S \cap h(G_i) \neq \emptyset$. Note that V_i is a random variabel depending only on $(h(\alpha_j))_{j<i}$. Let $D_i \subseteq [2^r] \backslash V_i$ be a set of *dummy hash values* chosen dependently on $(h(\alpha_j))_{j<i}$ such that $(|D_i| + |V_i|)/2^r = \rho|G_i|$. As $|G_i| \le n^{1-1/c}$ and so $\rho|G_i| \le \rho n^{1-1/c} \le 1$ this is in fact possible. We say that S is *hit* if there exists and $i \in \{1, \dots, k\}$ such that $h(\alpha_i) \in V_i \cup D_i$, and we denote this event $\mathcal{H}$. Defining $n_0 = n^{1-1/c}$ we then have

$$\Pr[\mathcal{H}] = 1 - \prod_{i=1}^{k}(1 - |G_i|\rho) \le 1 - (1 - n_0\rho)^{n/n_0} \le 1 - e^{-n\rho}(1 - nn_0\rho^2) \le p_0' + n^{2-1/c}\rho^2,$$

using the same inequality as in the proof of Theorem 1.1. This is clearly also an upper bound on $p = \Pr[h(X) \cap S \neq \emptyset] = \Pr[\bigcup_{i=1}^{k}(h(\alpha_i) \in V_i)]$.

Now for the lower bound: For $i \in \{1, \dots, k\}$ we let $\mathcal{D}_i$ and $\mathcal{R}_i$ denote events that $h(\alpha_i) \in D_i$ and that $h(\alpha_i) \in V_i$ respectively. Then

$$\Pr\left[\bigcup_{i=1}^{k} \mathcal{D}_i\right] \le \sum_{i=1}^{k} \Pr[\mathcal{D}_i] = \sum_{i=1}^{k}(\Pr[\mathcal{R}_i \cup \mathcal{D}_i] - \Pr[\mathcal{R}_i]) = \rho n - \sum_{i=1}^{k} \Pr[\mathcal{R}_i].$$

By the Bonferroni inequality, and 2-independence

$$\Pr[\mathcal{R}_i] = \Pr[h(G_i) \cap S \neq \emptyset] \ge |G_i|\rho - \binom{|G_i|}{2}\rho^2,$$

so it follows that $\Pr[\bigcup_{i=1}^{k} \mathcal{D}_i] \le \sum_{i=1}^{k} \binom{|G_i|}{2}\rho^2 \le n^{2-1/c}\rho^2$. Finally

$$p \ge \Pr[\mathcal{H}] - \Pr\left[\bigcup_{i=1}^{k} \mathcal{D}_i\right] \ge p_0' - n^{2-1/c}\rho^2,$$

which completes the proof of the lower bound.

The case where we condition on the event $\mathcal{E}$ that $h(q) = z$ for a $z \in [2^r]$ is handled analogously but this time choosing the order $\prec$ as described in the second part of Lemma 3.1. The upper bound on p then follows as before and for the lower bound we use 3-independence of simple tabulation when applying the Bonferroni inequality to lower bound $\Pr[\mathcal{V}_i \mid \mathcal{E}]$. $\square$

References

[1] Anders Aamand, Mathias Bæk Tejs Knudsen, and Mikkel Thorup. Power of d choices with simple tabulation. In *Proc. 45st International Colloquium on Automata, Languages and Programming*, ICALP, pages 5:1–5:14, 2018.

[2] Kazuoki Azuma. Weighted sums of certain dependent random variables. *Tohoku Mathematical Journal*, 19(3):357–367, 1967.

[3] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.

[4] Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: A survey. In *Internet Mathematics*, pages 636–646, 2002.

[5] Søren Dahlgaard, Mathias B. T. Knudsen, Eva Rotenberg, and Mikkel Thorup. Hashing for statistics over k-partitions. In *Proc. 56th Symposium on Foundations of Computer Science*, FOCS, pages 1292–1310, 2015.

[6] Martin Dietzfelbinger and Philipp Woelfel. Almost random graphs with simple hash functions. In *Proc. 35th ACM Symposium on Theory of Computing*, STOC, pages 629–638, 2003.

[7] Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul G. Spirakis. Space efficient hash tables with worst case constant access time. In *Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '03, pages 271–282, 2003.

[8] Anil Kamath, Rajeev Motwani, Krishna V. Palem, and Paul G. Spirakis. Tail bounds for occupancy and the satisfiability threshold conjecture. *Random Struct. Algorithms*, 7(1):59–80, 1995.

[9] Colin Mcdiarmid. Concentration. In *Probabilistic methods for algorithmic discrete mathematics*, Algorithms and combinatorics, pages 195–248. Springer, Berlin, 1998.

[10] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, New York, NY, USA, 2005.

[11] Michael Mitzenmacher and Salil Vadhan. Why simple hash functions work: Exploiting the entropy in a data stream. In *Proc. 19. ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 746–755, 2008.

[12] Anna Pagh and Rasmus Pagh. Uniform hashing in constant time and optimal space. *SIAM Journal of Computing*, 38(1):85–96, March 2008.

[13] Rasmus Pagh and Flemming F. Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, May 2004. Announced at ESA'01.

[14] Mihai Pătraşcu and Mikkel Thorup. The power of simple tabulation hashing. *Journal of the ACM*, 59(3):14:1–14:50, June 2012. Announced at STOC'11.

[15] Alan Siegel. On universal classes of extremely random constant-time hash functions. *SIAM Journal of Computing*, 33(3):505–543, March 2004. Announced at FOCS'89.

[16] Mikkel Thorup. Simple tabulation, fast expanders, double tabulation, and high independence. In *Proc. 54th Symposium on Foundations of Computer Science*, FOCS, pages 90–99, 2013.

[17] Mikkel Thorup. High speed hashing for integers and strings. *CoRR*, abs/1504.06804, 2014.

[18] Mikkel Thorup. Fast and powerful hashing using tabulation. *Commun. ACM*, 60(7):94–101, 2017.

[19] Mikkel Thorup and Yin Zhang. Tabulation-based 5-independent hashing with applications to linear probing and second moment estimation. *SIAM Journal of Computing*, 41(2):293–331, April 2012. Announced at SODA'04 and ALENEX'10.

[20] Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.*, 22(3):265–279, 1981. Announced at FOCS'79.

[21] Albert L. Zobrist. A new hashing method with application for game playing. *Tech. Report 88*, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1970.

Appendix D

(Learned) Frequency Estimation Algorithms under Zipfian Distribution

(Learned) Frequency Estimation Algorithms under Zipfian Distribution

Anders Aamand* Piotr Indyk† Ali Vakilian‡

Abstract

The frequencies of the elements in a data stream are an important statistical measure and the task of estimating them arises in many applications within data analysis and machine learning. Two of the most popular algorithms for this problem, Count-Min and Count-Sketch, are widely used in practice.

In a recent work [Hsu et al., ICLR'19], it was shown empirically that augmenting Count-Min and Count-Sketch with a machine learning algorithm leads to a significant reduction of the estimation error. The experiments were complemented with an analysis of the expected error incurred by Count-Min (both the standard and the augmented version) when the input frequencies follow a Zipfian distribution. Although the authors established that the learned version of Count-Min has lower estimation error than its standard counterpart, their analysis of the standard Count-Min algorithm was not tight. Moreover, they provided no similar analysis for Count-Sketch.

In this paper we resolve these problems. First, we provide a simple tight analysis of the expected error incurred by Count-Min. Second, we provide the first error bounds for both the standard and the augmented version of Count-Sketch. These bounds are nearly tight and again demonstrate an improved performance of the learned version of Count-Sketch.

In addition to demonstrating tight gaps between the aforementioned algorithms, we believe that our bounds for the standard versions of Count-Min and Count-Sketch are of independent interest. In particular, it is a typical practice to set the number of hash functions in those algorithms to $\Theta(\log n)$. In contrast, our results show that to minimize the *expected* error, the number of hash functions should be a constant, strictly greater than 1.

*BARC, University of Copenhagen, `aa@di.ku.dk`
†CSAIL, MIT, `indyk@mit.edu`
‡University of Wisconsin-Madison, `vakilian@wisc.edu`

1 Introduction

The last few years have witnessed a rapid growth in using machine learning methods to solve "classical" algorithmic problems. For example, they have been used to improve the performance of data structures [KBC$^+$18, Mit18], online algorithms [LV18, PSK18, GP19, Kod19, CGT$^+$19, ADJ$^+$20, LLMV20, Roh20, ACE$^+$20], combinatorial optimization [KDZ$^+$17, BDSV18, Mit20], similarity search [WLKC16, DIRW19], compressive sensing [MPB15, BJPD17] and streaming algorithms [HIKV19, IVY19, JLL$^+$20, CGP20]. Multiple frameworks for designing and analyzing such algorithms have been proposed [ACC$^+$11, GR17, BDV18, AKL$^+$19]. The rationale behind this line of research is that machine learning makes it possible to adapt the behavior of the algorithms to inputs from a specific data distribution, making them more efficient or more accurate in specific applications.

In this paper we focus on learning-augmented streaming algorithms for frequency estimation. The latter problem is formalized as follows: given a sequence S of elements from some universe U, construct a data structure that for any element $i \in U$ computes an estimation $\tilde{f}_i$ of f_i, the number of times i occurs in S. Since counting data elements is a very common subroutine, frequency estimation algorithms have found applications in many areas, such as machine learning, network measurements and computer security. Many of the most popular algorithms for this problem, such as Count-Min (CM) [CM05a] or Count-Sketch (CS) [CCFC02] are based on hashing. Specifically, these algorithms hash stream elements into B buckets, count the number of items hashed into each bucket, and use the bucket value as an estimate of item frequency. To improve the accuracy, the algorithms use $k > 1$ such hash functions and aggregate the answers. These algorithms have several useful properties: they can handle item deletions (implemented by decrementing the respective counters), and some of them (Count-Min) never underestimate the true frequencies, i.e., $\tilde{f}_i \geq f_i$.

In a recent work [HIKV19], the authors showed that the aforementioned algorithm can be improved by augmenting them with machine learning. Their approach is as follows. During the training phase, they construct a classifier (neural network) to detect whether an element is "heavy" (e.g., whether f_i is among top k frequent items). After such a classifier is trained, they scan the input stream, and apply the classifier to each element i. If the element is predicted to be heavy, it is allocated a unique bucket, so that an exact value of f_i is computed. Otherwise, the element is forwarded to a "standard" hashing data structure $\mathcal{C}$, e.g., CM or CS. To estimate $\tilde{f}_i$, the algorithm either returns the exact count f_i (if i is allocated a unique bucket) or an estimate provided by the data structure $\mathcal{C}$.[1] An empirical evaluation, on networking and query log data sets, shows that this approach can reduce the overall estimation error.

The paper also presents a preliminary analysis of the algorithm. Under the common assumption that the frequencies follow the Zipfian law, i.e.,[2] $f_i \propto 1/i$, for $i = 1, \dots, n$ for some n, and further that item i is queried with probability proportional to its frequency, the expected error incurred by the learning-augmented version of CM is shown to be asymptotically lower than that of the "standard" CM.[3] However, the exact magnitude of the gap between the error incurred by the learned and standard CM algorithms was left as an open problem. Specifically, [HIKV19] only shows that the expected error of standard CM with k hash functions and a total of B buckets is

[1]See Figure 1 for a generic implementation of the learning-based algorithms of [HIKV19].

[2]In fact we will assume that $f_i = 1/i$. This is just a matter of scaling and is convenient as it removes the dependence of the length of the stream in our bounds

[3]This assumes that the error rate for the "heaviness" predictor is sufficiently low.

between $\frac{k}{B\log(k)}$ and $\frac{k\log^{(k+2)/(k-1)}(kn/B)}{B}$. Furthermore, no such analysis was presented for CS.

1.1 Our results

In this paper we resolve the aforementioned questions left open in [HIKV19]. Assuming that the frequencies follow a Zipfian law, we show:

- An asymptotically tight bound of $\Theta(\frac{k\log(kn/B)}{B})$ for the expected error incurred by the CM algorithm with k hash functions and a total of B buckets. Together with a prior bound for Learned CM (Table 1), this shows that learning-augmentation improves the error of CM by a factor of $\Theta(\log(n)/\log(n/B))$ if the heavy hitter oracle is perfect.
- The first error bounds for CS and Learned CS (see Table 1). In particular, we show that for Learned CS, a single hash function as in [HIKV19] leads to an asymptotically optimal error bound, improving over standard CS by a factor of $\Theta(\log(n)/\log(n/B))$ (same as CM).

We highlight that our results are presented assuming that we use a *total* of B buckets. With k hash functions, the range of each hash functions is therefore $[B/k]$. We make this assumption since we wish to compare the expected error incurred by the different sketches when the total sketch size is fixed.

	$k = 1$	$k > 1$
Count-Min (CM)	$\Theta\left(\frac{\log n}{B}\right)$ [HIKV19]	$\Theta\left(\frac{k\cdot\log(\frac{kn}{B})}{B}\right)$
Learned Count-Min (L-CM)	$\Theta\left(\frac{\log^2(\frac{n}{B})}{B\log n}\right)$ [HIKV19]	$\Omega\left(\frac{\log^2(\frac{n}{B})}{B\log n}\right)$ [HIKV19]
Count-Sketch (CS)	$\Theta\left(\frac{\log B}{B}\right)$	$\Omega\left(\frac{k^{1/2}}{B\log k}\right)$ and $O\left(\frac{k^{1/2}}{B}\right)$
Learned Count-Sketch (L-CS)	$\Theta\left(\frac{\log\frac{n}{B}}{B\log n}\right)$	$\Omega\left(\frac{\log\frac{n}{B}}{B\log n}\right)$

Table 1: This table summarizes our and previously known results on the expected frequency estimation error of Count-Min (CM), Count-Sketch (CS) and their learned variants (i.e., L-CM and L-CS) that use k functions and overall space $k \times \frac{B}{k}$ under Zipfian distribution. For CS, we assume that k is odd (so that the median of k values is well defined).

For our results on L-CS in Table 1 we initially assume that the heavy hitter oracle is *perfect*, i.e., that it makes no mistakes when classifying the heavy items. This is unlikely to be the case in practice, so we complement the results with an analysis of L-CS when the heavy hitter oracle may err with probability at most δ on each item. As δ varies in $[0, 1]$, we obtain a smooth trade-off between the performance of L-CS and its classic counterpart. Specifically, as long as $\delta = O(1/\log B)$, the bounds are as good as with a perfect heavy hitter oracle.

In addition to clarifying the gap between the learned and standard variants of popular frequency estimation algorithms, our results provide interesting insights about the algorithms themselves. For example, for both CM and CS, the number of hash functions k is often selected to be $\Theta(\log n)$, in order to guarantee that *every* frequency is estimated up to a certain error bound. In contrast, we show that if instead the goal is to bound the *expected* error, then setting k to a constant (strictly greater than 1) leads to the asymptotic optimal performance. We remark that the same phenomenon holds not only for a Zipfian query distribution but in fact for an arbitrary distribution on the queries (see Remark 2.2).

Let us make the above comparison with previous known bounds for CM and CS a bit more precise. With frequency vector $\mathbf{f}$ and for an element x in the stream, we denote by $\mathbf{f}_{-x}^{(B)}$, the vector obtained by setting the entry corresponding to x as well as the B largest entries of $\mathbf{f}$ to 0. The classic technique for analysing CM and CS (see, e.g., [CCFC02]) shows that using a single hash function and B buckets, with probability $\Omega(1)$, the error when querying the frequency of an element x is $O(\|\mathbf{f}_{-x}^{(B)}\|_1/B)$ for CM and $O(\|\mathbf{f}_{-x}^{(B)}\|_2/\sqrt{B})$ for CS. By creating $O(\log(1/\delta))$ sketches and using the median trick, the error probability can then be reduced to δ. For the Zipfian distribution, these two bounds become $O(\log(n/B)/B)$ and $O(1/B)$ respectively, and to obtain them with high probability for all elements we require a sketch of size $\Omega(B \log n)$. Our results imply that to obtain similar bounds on the expected error, we only require a sketch of size $O(B)$ and a constant number of hash functions. The classic approach described above does not yield tight bounds on the expected errors of CM and CS when $k > 1$ and to obtain our bounds we have to introduce new and quite different techniques as to be described in Section 1.3.

Our techniques are quite flexible. To illustrate this, we study the performance of the classic Count-Min algorithm with one and more hash functions, as well as its learned counterparts, in the case where the input follows the following more general Zipfian distribution with exponent $\alpha > 0$. This distribution is defined by $f_i \propto 1/i^\alpha$ for $i \in [n]$. We present the precise results in Table 3 in Appendix A.

In Section 6, we complement our theoretical bounds with empirical evaluation of standard and learned variants of Count-Min and Count-Sketch on a synthetic dataset, thus providing a sense of the constant factors of our asymptotic bounds.

1.2 Related work

The frequency estimation problem and the closely related heavy hitters problem are two of the most fundamental problems in the field of streaming algorithms [CM05a, CM05b, CCFC02, M$^+$05, CH08, CH10, BICS10, MP14, BCIW16, LNNT16, ABL$^+$17, BCI$^+$17, BDW18]. In addition to the aforementioned hashing-based algorithms (e.g., [CM05a, CCFC02]), multiple non-hashing algorithms were also proposed, e.g., [MG82, MM02, MAEA05]. These algorithms often exhibit better accuracy/space tradeoffs, but do not posses many of the properties of hashing-based methods, such as the ability to handle deletions as well as insertions.

Zipf law is a common modeling tool used to evaluate the performance of frequency estimation algorithms, and has been used in many papers in this area, including [MM02, MAEA05, CCFC02]. In its general form it postulates that f_i is proportional to $1/i^\alpha$ for some exponent parameter $\alpha > 0$. In this paper we focus mostly on the "original" Zipf law where $\alpha = 1$. We do, however, study Count-Min for more general values of α and the techniques introduced in this paper can be applied to other values of the exponent α for Count-Sketch as well.

1.3 Our techniques

Our main contribution is our analysis of the standard Count-Min and Count-Sketch algorithms for Zipfians with $k > 1$ hash functions. Showing the improvement for the learned counterparts is relatively simple (for Count-Min it was already done in [HIKV19]). In both of these analyses we consider a fixed item i and bound $\mathbb{E}[|f_i - \tilde{f}_i|]$ whereupon linearity of expectation leads to the desired results. In the following we assume that $f_j = 1/j$ for each $j \in [n]$ and describe our techniques for bounding $\mathbb{E}[|f_i - \tilde{f}_i|]$ for each of the two algorithms.

Count-Min. With a single hash function and B buckets it is easy to see that the head of the Zipfian distribution, namely the items of frequencies $(f_j)_{j \in [B]}$, contribute with $\log B/B$ to the expected error $\mathbb{E}[|f_i - \tilde{f}_i|]$, whereas the light items contribute with $\log(n/B)/B$. Our main observation is that with more hash functions the expected contribution from the heavy items drops to $1/B$ and so, the main contribution comes from the light items. To bound the expected contribution of the heavy items to the error $|f_i - \tilde{f}_i|$ we bound the probability that the contribution from these items is at least t, then integrate over t. The main observation is that if the error is at least t then for each of the hash functions, either there exist t/s items in $[B]$ hashing to the same bucket as i or there is an item $j \neq i$ in $[B]$ of weight at most s hashing to the same bucket as i. By a union bound, optimization over s, and some calculations, this gives the desired bound. The lower bound follows from simple concentration inequalities on the contribution of the tail. In contrast to the analysis from [HIKV19] which is technical and leads to suboptimal bounds, our analysis is short, simple, and yields completely tight bounds in terms of all of the parameters k, n and B.

Count-Sketch. Simply put, our main contribution is an improved understanding of the distribution of random variables of the form $S = \sum_{i=1}^{n} f_i \eta_i \sigma_i$. Here the $\eta_i \in \{0,1\}$ are i.i.d Bernouilli random variables and the $\sigma_i \in \{-1,1\}$ are independent Rademachers, that is, $\Pr[\eta_i = 1] = \Pr[\eta_i = -1] = 1/2$. Note that the counters used in CS are random variables having precisely this form. Usually such random variables are studied for the purpose of obtaining large deviation results. In contrast, in order to analyze CS, we are interested in a fine-grained picture of the distribution within a "small" interval I around zero, say with $\Pr[S \in I] = 1/2$. For example, when proving a lower bound on $\mathbb{E}[|f_i - \tilde{f}_i|]$, we must establish a certain *anti-concentration* of S around 0. More precisely we find an interval $J \subset I$ centered at zero such that $\Pr[S \in J] = O(1/\sqrt{k})$. Combined with the fact that we use k independent hash functions as well as properties of the median and the binomial distribution, this gives that $\mathbb{E}[|f_i - \tilde{f}_i|] = \Omega(|J|)$. Anti-concentration inequalities of this type are in general notoriously hard to obtain but it turns out that we can leverage the properties of the Zipfian distribution, specifically its heavy head. For our upper bounds on $\mathbb{E}[|f_i - \tilde{f}_i|]$ we need strong lower bounds on $\Pr[S \in J]$ for intervals $J \subset I$ centered at zero. Then using concentration inequalities we can bound the probability that half of the k relevant counters are smaller (larger) than the lower (highter) endpoint of J, i.e., that the median does not lie in J. Again this requires a precise understanding of the distribution of S within I.

1.4 Structure of the paper

In Section 2 we describe the algorithms Count-Min and Count-Sketch. We also formally define the estimation error that we will study as well as the Zipfian distribution. In Sections 3 and 4 we provide our analyses of the expected error of Count-Min and Count-Sketch. In Section 5 we analyze the performance of learned Count-Sketch both when the heavy hitter oracle is perfect and when it may misclassify each item with probability at most δ. In Section 6 we present our experiments. Finally, in Appendix A, we analyse Count-Min for the generalized Zipfian distribution with exponent $\alpha > 0$ both in the classic and learned case and prove matching lower bounds for the learned algorithms.

2 Preliminaries

We start out by describing the sketching algorithms Count-Min and Count-Sketch. Common to both of these algorithms is that we sketch a stream S of elements coming from some universe U of size n. For notational convenience we will assume that $U = [n] := \{1, \dots, n\}$. If item i occurs f_i times then either algorithm outputs an estimate $\tilde{f}_i$ of f_i.

Count-Min. We use k independent and uniformly random hash functions $h_1, \dots, h_k : [n] \to [B]$. Letting C be an array of size $[k] \times [B]$ we let $C[\ell, b] = \sum_{j \in [n]} [h_\ell(j) = b] f_j$. When querying $i \in [n]$ the algorithm returns $\tilde{f}_i = \min_{\ell \in [k]} C[\ell, h_\ell(i)]$. Note that we always have that $\tilde{f}_i \geq f_i$.

Count-Sketch. We pick independent and uniformly random hash functions $h_1, \dots, h_k : [n] \to [B]$ and $s_1, \dots, s_k : [n] \to \{-1, 1\}$. Again we initialize an array C of size $[k] \times [B]$ but now we let $C[\ell, b] = \sum_{j \in [n]} [h_\ell(j) = b] s_\ell(j) f_j$. When querying $i \in [n]$ the algorithm returns the estimate $\tilde{f}_i = \mathsf{median}_{\ell \in [k]}\, s_\ell(i) \cdot C[\ell, h_\ell(i)]$.

Remark 2.1. The bounds presented in Table 1 assumes that the hash functions have codomain $[B/k]$ and not $[B]$, i.e., that the *total* number of buckets is B. In the proofs to follows we assume for notational ease that the hash functions take value in $[B]$ and the claimed bounds follows immediately by replacing B by B/k.

Estimation Error. To measure and compare the overall accuracy of different frequency estimation algorithms, we will use the *expected* estimation error which is defined as follows: let $\mathcal{F} = \{f_1, \cdots, f_n\}$ and $\tilde{\mathcal{F}}_\mathcal{A} = \{\tilde{f}_1, \cdots, \tilde{f}_n\}$ respectively denote the actual frequencies and the estimated frequencies obtained from algorithm $\mathcal{A}$ of items in the input stream. We remark that when $\mathcal{A}$ is clear from the context we denote $\tilde{\mathcal{F}}_\mathcal{A}$ as $\tilde{\mathcal{F}}$. Then we define

$$\mathrm{Err}(\mathcal{F}, \tilde{\mathcal{F}}_\mathcal{A}) := \mathbb{E}_{i \sim \mathcal{D}} |f_i - \tilde{f}_i|, \tag{1}$$

where $\mathcal{D}$ denotes the *query distribution* of the items. Here, similar to previous work (e.g., [RKA16, HIKV19]), we assume that the query distribution $\mathcal{D}$ is the same as the frequency distribution of items in the stream, i.e., for any $i^* \in [n]$, $\Pr_{i \sim \mathcal{D}}[i = i^*] \propto f_{i^*}$ (more precisely, for any $i^* \in [n]$, $\Pr_{i \sim \mathcal{D}}[i = i^*] = f_{i^*}/N$ where $N = \sum_{i \in [n]} f_i$ denotes the total sum of all frequencies in the stream).

Remark 2.2. As all upper/lower bounds in this paper are proved by bounding the expected error when estimating the frequency of a single item, $\mathbb{E}[|\tilde{f}_i - f_i|]$, then using linearity of expectation, in fact we obtain bounds for *any* query distribution $(p_i)_{i \in [n]}$.

Zipfian Distribution. In our analysis we assume that the frequency distribution of items follows Zipf's law. That is, if we sort the items according to their frequencies with no loss of generality assuming that $f_1 \geq f_2 \geq \cdots \geq f_n$, then for any $i \in [n]$, $f_i \propto 1/i$. In fact, we shall assume that $f_i = 1/i$, which is just a matter of scaling, and which conveniently removes the dependence on the length of the stream in our bounds. Assuming that the query distribution is the same as the distribution of the frequencies of items in the input stream (i.e., $\Pr_{i \sim \mathcal{D}}[i^*] = f_{i^*}/N = 1/(i^* \cdot H_n)$ where H_n denotes the n-th harmonic number), we can write the expected error in eq. (1) as follows:

$$\mathrm{Err}(\mathcal{F}, \tilde{\mathcal{F}}_\mathcal{A}) = \mathbb{E}_{i \sim \mathcal{D}}[|f_i - \tilde{f}_i|] = \frac{1}{N} \cdot \sum_{i \in [n]} |\tilde{f}_i - f_i| \cdot f_i = \frac{1}{H_n} \cdot \sum_{i \in [n]} |\tilde{f}_i - f_i| \cdot \frac{1}{i} \tag{2}$$

Throughout this paper, we present our results with respect to the objective function at the right hand side of eq. (2), i.e., $(1/H_n) \cdot \sum_{i=1}^{n} |\tilde{f}_i - f_i| \cdot f_i$. However, it is easy to use our results to obtain bounds for any query distribution as stated in Remark 2.2.

Later we shall study the generalized Zipfian distribution with exponent $\alpha > 0$. Sorting the items according to their frequencies, $f_1, \geq f_2 \geq \cdots \geq f_n$, it holds for any $i \in [n]$ that $f_i \propto 1/i^{\alpha}$. Again we present our result with respect to the objective function $\sum_{i=1}^{n} |\tilde{f}_i - f_i| \cdot f_i$.

Algorithm 1 Learning-Based Frequency Estimation

```
1: procedure LearnedSketch($B$, $B_h$, HH-Oracle, SketchAlg)
2:     for each stream element $i$ do
3:         if HH-Oracle($i$) $= 1$ then          ▷ predicts whether $i$ is heavy (in top $B_h$- frequent items)
4:             if a unique bucket is already assigned to item $i$ then
5:                 $\mathsf{counter}_i \leftarrow \mathsf{counter}_i + 1$
6:             else
7:                 allocate a new unique bucket to item $i$ and $\mathsf{counter}_i \leftarrow 1$
8:             end if
9:         else
10:            feed $i$ to SketchAlg($B - B_h$)          ▷ an instance of SketchAlg with $B - B_h$ buckets
11:        end if
12:    end for
13: end procedure
```

Figure 1: A generic learning augmented algorithm for the frequency estimation problem. HH-Oracle denotes a given learned oracle for detecting whether the item is among the top B_h frequent items of the stream and SketchAlg is a given (sketching) algorithm (e.g., CM or CS) for the frequency estimation problem.

Learning Augmented Sketching Algorithms for Frequency Estimation. In this paper, following the approach of [HIKV19], the *learned* variants of CM and CS are algorithms augmented with a machine learning based *heavy hitters* oracle. More precisely, we assume that the algorithm has access to an oracle HH-Oracle that predicts whether an item is "heavy" (i.e., is one of the B_h most frequent items) or not. Then, the algorithm treats heavy and non-heavy items differently: (a) a unique bucket is allocated to each heavy item and their frequencies are computed with no error, (b) the rest of items are fed to the given (sketching) algorithm SketchAlg using the remaining $B - B_h$ buckets and their frequency estimates are computed via SketchAlg (see Figure 1). We shall assume that $B_h = \Theta(B - B_h) = \Theta(B)$, that is, we use asymptotically the same number of buckets for the heavy items as for the sketching of the light items. One justification for this assumption is that in any case we can increase both the number of buckets for heavy and light items to B without affecting the overall asymptotic space usage.

Note that, in general the oracle HH-Oracle can make errors. In our analysis we first obtain a theoretical understanding, by assuming that the oracle is perfect, i.e., the error rate is zero. We later complement this analysis, by studying the incurred error when the oracle misclassifies each item with probability at most δ.

Concentration bounds. We finally collect some concentration inequalities for reference in the the proofs of our results. The inequality we will use the most is Bennett's inequality. However, we remark that for our applications, several other variance based concentration result would suffice, e.g., Bernstein's inequality.

Theorem 2.3 (Bennett's inequality [Ben62])**.** *Let $X_1, \dots, X_n$ be independent, mean zero random variables. Let $S = \sum_{i=1}^n X_i$, and $\sigma^2, M > 0$ be such that* $\mathrm{Var}[S] \le \sigma^2$ *and $|X_i| \le M$ for all $i \in [n]$. For any $t \ge 0$,*

$$\Pr[S \ge t] \le \exp\left(-\frac{\sigma^2}{M^2} h\left(\frac{tM}{\sigma^2}\right)\right),$$

where $h : \mathbb{R}_{\ge 0} \to \mathbb{R}_{\ge 0}$ is defined by $h(x) = (x+1)\log(x+1) - x$. The same tail bound holds on the probability $\Pr[S \le -t]$.

Remark 2.4. For $x \ge 0$, $\frac{1}{2}x\log(x+1) \le h(x) \le x\log(x+1)$. We will use these asymptotic bounds repeatedly in this paper.

A corollary of Bennett's inequality is the classic Chernoff bounds.

Theorem 2.5 (Chernoff [Che52])**.** *Let $X_1, \dots, X_n \in [0,1]$ be independent random variables and $S = \sum_{i=1}^n X_i$. Let $\mu = \mathbb{E}[S]$. Then*

$$\Pr[S \ge (1+\delta)\mu] \le \exp(-\mu h(\delta)).$$

Even weaker than Chernoff's inequality is Hoeffding's inequality.

Theorem 2.6 (Hoeffding [Hoe63])**.** *Let $X_1, \dots, X_n \in [0,1]$ be independent random variables. Let $S = \sum_{i=1}^n X_i$. Then*

$$\Pr[S - \mathbb{E}[S] \ge t] \le e^{-\frac{2t^2}{n}}.$$

3 Tight Bounds for Count-Min with Zipfians

For both Count-Min and Count-Sketch we aim at analyzing the expected value of the variable $\sum_{i \in [n]} f_i \cdot |\tilde{f}_i - f_i|$ where $f_i = 1/i$ and $\tilde{f}_i$ is the estimate of f_i output by the relevant sketching algorithm. Throughout this paper we use the following notation: For an event E we denote by $[E]$ the random variable in $\{0,1\}$ which is 1 if and only if E occurs. We begin by presenting our improved analysis of Count-Min with Zipfians. The main theorem is the following.

Theorem 3.1. *Let $n, B, k \in \mathbb{N}$ with $k \ge 2$ and $B \le n/k$. Let further $h_1, \dots, h_k : [n] \to [B]$ be independent and truly random hash functions. For $i \in [n]$ define the random variable $\tilde{f}_i = \min_{\ell \in [k]} \left(\sum_{j \in [n]} [h_\ell(j) = h_\ell(i)] f_j\right)$. For any $i \in [n]$ it holds that $\mathbb{E}[|\tilde{f}_i - f_i|] = \Theta\left(\frac{\log(\frac{n}{B})}{B}\right)$.*

Replacing B by B/k in Theorem 3.1 and using linearity of expectation we obtain the desired bound for Count-Min in the upper right hand side of Table 1. The natural assumption that $B \le n/k$ simply says that the total number of buckets is upper bounded by the number of items.

To prove Theorem 3.1 we start with the following lemma which is a special case of the theorem.

Lemma 3.2. *Suppose that we are in the setting of Theorem 3.1 and further that*[4] $n = B$. *Then*

$$\mathbb{E}[|\tilde{f}_i - f_i|] = O\left(\frac{1}{n}\right).$$

Proof. It suffices to show the result when $k = 2$ since adding more hash functions and corresponding tables only decreases the value of $|\tilde{f}_i - f_i|$. Define $Z_\ell = \sum_{j\in[n]\setminus\{i\}}[h_\ell(j) = h_\ell(i)]f_j$ for $\ell \in [2]$ and note that these variables are independent. For a given $t \geq 3/n$ we wish to upper bound $\Pr[Z_\ell \geq t]$. Let $s < t$ be such that t/s is an integer, and note that if $Z_\ell \geq t$ then either of the following two events must hold:

E_1: There exists a $j \in [n] \setminus \{i\}$ with $f_j > s$ and $h_\ell(j) = h_\ell(i)$.

E_2: The set $\{j \in [n] \setminus \{i\} : h_\ell(j) = h_\ell(i)\}$ contains at least t/s elements.

To see this, suppose that $Z_\ell \geq t$ and that E_1 does not hold. Then

$$t \leq Z_\ell = \sum_{j\in[n]\setminus\{i\}} [h_\ell(j) = h_\ell(i)]f_j \leq s|\{j \in [n] \setminus \{i\} : h_\ell(j) = h_\ell(i)\}|,$$

so it follows that E_2 holds. By a union bound,

$$\Pr[Z_\ell \geq t] \leq \Pr[E_1] + \Pr[E_2] \leq \frac{1}{ns} + \binom{n}{t/s} n^{-t/s} \leq \frac{1}{ns} + \left(\frac{es}{t}\right)^{t/s}.$$

Choosing $s = \Theta(\frac{t}{\log(tn)})$ such that t/s is an integer, and using $t \geq \frac{3}{n}$, a simple calculation yields that $\Pr[Z_\ell \geq t] = O\left(\frac{\log(tn)}{tn}\right)$. Note that $|\tilde{f}_i - f_i| = \min(Z_1, Z_2)$. As Z_1 and Z_2 are independent, $\Pr[|\tilde{f}_i - f_i| \geq t] = O\left(\left(\frac{\log(tn)}{tn}\right)^2\right)$, so

$$\mathbb{E}[|\tilde{f}_i - f_i|] = \int_0^\infty \Pr[Z \geq t]\, dt \leq \frac{3}{n} + O\left(\int_{3/n}^\infty \left(\frac{\log(tn)}{tn}\right)^2 dt\right) = O\left(\frac{1}{n}\right).$$

□

We can now prove the full statement of Theorem 3.1.

Proof of Theorem 3.1. We start out by proving the upper bound. Let $N_1 = [B] \setminus \{i\}$ and $N_2 = [n] \setminus ([B] \cup \{i\})$. Let $b \in [k]$ be such that $\sum_{j\in N_1} f_j \cdot [h_b(j) = h_b(i)]$ is minimal. Note that b is itself a random variable. We also define

$$Y_1 = \sum_{j\in N_1} f_j \cdot [h_b(j) = h_b(i)], \text{ and } Y_2 = \sum_{j\in N_2} f_j \cdot [h_b(j) = h_b(i)].$$

Then, $|\tilde{f}_i - f_i| \leq Y_1 + Y_2$. Using Lemma 3.2, we obtain that $\mathbb{E}[Y_1] = O(\frac{1}{B})$. For Y_2 we observe that

$$\mathbb{E}[Y_2 \mid b] = \sum_{j\in N_2} \frac{f_j}{B} = O\left(\frac{\log\left(\frac{n}{B}\right)}{B}\right).$$

[4] In particular we dispose with the assumption that $B \leq n/k$.

We conclude that

$$\mathbb{E}[|\tilde{f}_i - f_i|] \le \mathbb{E}[Y_1] + \mathbb{E}[Y_2] = \mathbb{E}[Y_1] + \mathbb{E}[\mathbb{E}[Y_2 \mid b]] = O\left(\frac{\log\left(\frac{n}{B}\right)}{B}\right).$$

Next we prove the lower bound. We have already seen that the main contribution to the error comes from the tail of the distribution. As the tail of the distribution is relatively "flat" we can simply apply a concentration inequality to argue that with probability $\Omega(1)$, we have this asymptotic contribution for each of the k hash functions. To be precise, for $j \in [n]$ and $\ell \in [k]$ we define $X_\ell^{(j)} = f_j \cdot \left([h_\ell(j) = h_\ell(i)] - \frac{1}{B}\right)$. Note that the variables $(X_\ell^{(j)})_{j\in[n]}$ are independent. We also define $S_\ell = \sum_{j\in N_2} X_\ell^{(j)}$ for $\ell \in [k]$. Observe that $|X_\ell^{(j)}| \le f_j \le \frac{1}{B}$ for $j \ge B$, $\mathbb{E}[X_\ell^{(j)}] = 0$, and that

$$\mathrm{Var}[S_\ell] = \sum_{j\in N_2} f_j^2 \left(\frac{1}{B} - \frac{1}{B^2}\right) \le \frac{1}{B^2}.$$

Applying Bennett's inequality(Theorem 2.3 of Section 2), with $\sigma^2 = \frac{1}{B^2}$ and $M = 1/B$ thus gives that

$$\Pr[S_\ell \le -t] \le \exp\left(-h\left(tB\right)\right).$$

Defining $W_\ell = \sum_{j\in N_2} f_j \cdot [h_\ell(j) = h_\ell(i)]$ it holds that $\mathbb{E}[W_\ell] = \Theta\left(\frac{\log\left(\frac{n}{B}\right)}{B}\right)$ and $S_\ell = W_\ell - \mathbb{E}[W_\ell]$, so putting $t = \mathbb{E}[W_\ell]/2$ in the inequality above we obtain that

$$\Pr[W_\ell \le \mathbb{E}[W_\ell]/2] = \Pr[S_\ell \le -\mathbb{E}[W_\ell]/2] \le \exp\left(-h\left(\Omega\left(\log\frac{n}{B}\right)\right)\right).$$

Appealing to Remark 2.4 and using that $B \le n/k$ the above bound becomes

$$\begin{aligned}\Pr[W_\ell \le \mathbb{E}[W_\ell]/2] &\le \exp\left(-\Omega\left(\log\frac{n}{B}\cdot\log\left(\log\frac{n}{B}+1\right)\right)\right)\\ &= \exp(-\Omega(\log k \cdot \log(\log k + 1))) = k^{-\Omega(\log(\log k+1))}.\end{aligned} \tag{3}$$

By the independence of the events $(W_\ell > E[W_\ell]/2)_{\ell\in[k]}$, we have that

$$\Pr\left[|\tilde{f}_i - f_i| \ge \frac{\mathbb{E}[W_\ell]}{2}\right] \ge (1 - k^{-\Omega(\log(\log k+1))})^k = \Omega(1),$$

and so $\mathbb{E}[|\tilde{f}_i - f_i|] = \Omega(\mathbb{E}[W_\ell]) = \Omega\left(\frac{\log\left(\frac{n}{B}\right)}{B}\right)$, as desired. □

Remark 3.3. We have stated Theorem 3.1 for truly random hash functions but it suffices with $O(\log B)$-independent hashing to prove the upper bound. Indeed, the only step in which we require high independence is in the union bound in Lemma 3.2 over the $\binom{n}{t/s}$ subsets of $[n]$ of size t/s. To optimize the bound we had to choose $s = t/\log(tn)$, so that $t/s = \log(tn)$. As we only need to consider values of t with $t \le \sum_{i=1}^n f_i = O(\log n)$, in fact $t/s = O(\log n)$ in our estimates. Finally, we applied Lemma 3.2 with $n = B$ so it follows that $O(\log B)$-independence is enough to obtain our upper bound.

4 (Nearly) Tight Bounds for Count-Sketch with Zipfians

In this section we proceed to analyze Count-Sketch for Zipfians either using a single or more hash functions. We start with two simple lemmas which for certain frequencies $(f_i)_{i\in[n]}$ of the items in the stream can be used to obtain respectively good upper and lower bounds on $\mathbb{E}[|\tilde{f}_i - f_i|]$ in Count-Sketch with a single hash function. We will use these two lemmas both in our analysis of standard and learned Count-Sketch for Zipfians.

Lemma 4.1. *Let $w = (w_1, \dots, w_n) \in \mathbb{R}^n$, $\eta_1, \dots, \eta_n$ Bernoulli variables taking value 1 with probability p, and $\sigma_1, \dots, \sigma_n \in \{-1, 1\}$ independent Rademachers, i.e.,* $\Pr[\sigma_i = 1] = \Pr[\sigma_i = -1] = 1/2$. *Let $S = \sum_{i=1}^n w_i\eta_i\sigma_i$. Then, $\mathbb{E}[|S|] = O\left(\sqrt{p}\|w\|_2\right)$.*

Proof. Using that $\mathbb{E}[\sigma_i\sigma_j] = 0$ for $i \neq j$ and Jensen's inequality $\mathbb{E}[|S|]^2 \leq \mathbb{E}[S^2] = \mathbb{E}\left[\sum_{i=1}^n w_i^2\eta_i\right] = p\|w\|_2^2$, from which the result follows. □

Lemma 4.2. *Suppose that we are in the setting of Lemma 4.1. Let $I \subset [n]$ and let $w_I \in \mathbb{R}^n$ be defined by $(w_I)_i = [i \in I] \cdot w_i$. Then*

$$\mathbb{E}[|S|] \geq \frac{1}{2}p\,(1-p)^{|I|-1}\,\|w_I\|_1.$$

Proof. Let $J = [n] \setminus I$, $S_1 = \sum_{i\in I} w_i\eta_i\sigma_i$, and $S_2 = \sum_{i\in J} w_i\eta_i\sigma_i$. Let E denote the event that S_1 and S_2 have the same sign or $S_2 = 0$. Then $\Pr[E] \geq 1/2$ by symmetry. For $i \in I$ we denote by A_i the event that $\{j \in I : \eta_j \neq 0\} = \{i\}$. Then $\Pr[A_i] = p(1-p)^{|I|-1}$ and furthermore A_i and E are independent. If $A_i \cap E$ occurs, then $|S| \geq |w_i|$ and as the events $(A_i \cap E)_{i\in I}$ are disjoint it thus follows that $\mathbb{E}[|S|] \geq \sum_{i\in I} \Pr[A_i \cap E] \cdot |w_i| \geq \frac{1}{2}p\,(1-p)^{|I|-1}\,\|w_I\|_1$. □

With these tools in hand, we proceed to analyse Count-Sketch for Zipfians with one and more hash functions in the next two sections.

4.1 One hash function

By the same argument as in the discussion succeeding Theorem 3.1, the following theorem yields the desired result for a single hash function as presented in Table 1.

Theorem 4.3. *Suppose that $B \leq n$ and let $h : [n] \to [B]$ and $s : [n] \to \{-1, 1\}$ be truly random hash functions. Define the random variable $\tilde{f}_i = \sum_{j\in[n]}[h(j) = h(i)]s(j)f_j$ for $i \in [n]$. Then*

$$\mathbb{E}[|\tilde{f}_i - s(i)f_i|] = \Theta\left(\frac{\log B}{B}\right).$$

Proof. Let $i \in [n]$ be fixed. We start by defining $N_1 = [B] \setminus \{i\}$ and $N_2 = [n] \setminus ([B] \cup \{i\})$ and note that

$$|\tilde{f}_i - s(i)f_i| \leq \left|\sum_{j\in N_1}[h(j) = h(i)]s(j)f_j\right| + \left|\sum_{j\in N_2}[h(j) = h(i)]s(j)f_j\right| := X_1 + X_2.$$

Using the triangle inequality $\mathbb{E}[X_1] \le \frac{1}{B}\sum_{j\in N_1} f_j = O(\frac{\log B}{B})$. Also, by Lemma 4.1, $\mathbb{E}[X_2] = O\left(\frac{1}{B}\right)$ and combining the two bounds we obtain the desired upper bound. For the lower bound we apply Lemma 4.2 with $I = N_1$ concluding that

$$\mathbb{E}[|\tilde{f}_i - s(i)f_i|] \ge \frac{1}{2B}\left(1-\frac{1}{B}\right)^{|N_1|-1}\sum_{i\in N_1} f_i = \Omega\left(\frac{\log B}{B}\right).$$

□

4.2 Multiple hash functions

Let $k \in \mathbb{N}$ be odd. For a tuple $x = (x_1, \dots, x_k) \in \mathbb{R}^k$ we denote by $\mathsf{median}\, x$ the median of the entries of x. The following theorem immediately leads to the result on CS with $k \ge 3$ hash functions claimed in Table 1.

Theorem 4.4. *Let $k \ge 3$ be odd, $n \ge kB$, and $h_1, \dots, h_k : [n] \to [B]$ and $s_1, \dots, s_k : [n] \to \{-1, 1\}$ be truly random hash functions. Define $\tilde{f}_i = \mathsf{median}_{\ell\in[k]}\left(\sum_{j\in[n]}[h_\ell(j) = h_\ell(i)]s_\ell(j)f_j\right)$ for $i \in [n]$. Assume that*[5] *$k \le B$. Then*

$$\mathbb{E}[|\tilde{f}_i - s(i)f_i|] = \Omega\left(\frac{1}{B\sqrt{k}\log k}\right), \quad \textit{and} \quad \mathbb{E}[|\tilde{f}_i - s(i)f_i|] = O\left(\frac{1}{B\sqrt{k}}\right)$$

The assumption $n \ge kB$ simply says that the total number of buckets is upper bounded by the number of items. Again using linearity of expectation for the summation over $i \in [n]$ and replacing B by B/k we obtain the claimed upper and lower bounds of $\frac{\sqrt{k}}{B\log k}$ and $\frac{\sqrt{k}}{B}$ respectively. We note that even if the bounds above are only tight up to a factor of $\log k$ they still imply that it is asymptotically optimal to choose $k = O(1)$, e.g. $k = 3$. To settle the correct asymptotic growth is thus of merely theoretical interest.

In proving the upper bound in Theorem 4.4, we will use the following result by Minton and Price (Corollary 3.2 of [MP14]) proved via an elegant application of the Fourier transform.

Lemma 4.5 (Minton and Price [MP14])**.** *Let $\{X_i : i \in [n]\}$ be independent symmetric random variables such that $\Pr[X_i = 0] \ge 1/2$ for each i. Let $X = \sum_{i=1}^n X_i$ and $\sigma^2 = \mathbb{E}[X^2] = \mathrm{Var}[X]$. For $\varepsilon < 1$ it holds that $\Pr[|X| < \varepsilon\sigma] = \Omega(\varepsilon)$*

Proof of Theorem 4.4. If B (and hence k) is a constant, then the results follow easily from Lemma 4.1, so in what follows we may assume that B is larger than a sufficiently large constant. We subdivide the exposition into the proofs of the upper and lower bounds.

Upper bound Define $N_1 = [B] \setminus \{i\}$ and $N_2 = [n] \setminus ([B] \cup \{i\})$. Let for $\ell \in [k]$, $X_1^{(\ell)} = \sum_{j\in N_1}[h_\ell(j) = h_\ell(i)]s_\ell(j)f_j$ and $X_2^{(\ell)} = \sum_{j\in N_2}[h_\ell(j) = h_\ell(i)]s_\ell(j)f_j$ and let $X^{(\ell)} = X_1^{(\ell)} + X_2^{(\ell)}$.

As the absolute error in Count-Sketch with one pair of hash functions (h, s) is always upper bounded by the corresponding error in Count-Min with the single hash function h, we can use the bound in the proof of Lemma 3.2 to conclude that $\Pr[|X_1^{(\ell)}| \ge t] = O(\frac{\log(tB)}{tB})$, when $t \ge 3/B$. Also

[5]This very mild assumption can probably be removed at the cost of a more technical proof. In our proof it can even be replaced by $k \le B^{2-\varepsilon}$ for any $\varepsilon = \Omega(1)$.

$\mathrm{Var}[X_2^{(\ell)}] = (\frac{1}{B} - \frac{1}{B^2}) \sum_{j \in N_2} f_j^2 \leq \frac{1}{B^2}$, so by Bennett's inequality (Theorem 2.3) with $M = 1/B$ and $\sigma^2 = 1/B^2$ and Remark 2.4,

$$\Pr[|X_2^{(\ell)}| \geq t] \leq 2\exp\left(-h(tB)\right) \leq 2\exp\left(-\frac{1}{2}tB\log\left(tB+1\right)\right) = O\left(\frac{\log(tB)}{tB}\right),$$

for $t \geq \frac{3}{B}$. It follows that for $t \geq 3/B$,

$$\Pr[|X^{(\ell)}| \geq 2t] \leq \Pr[(|X_1^{(\ell)}| \geq t)] + \Pr(|X_2^{(\ell)}| \geq t)] = O\left(\frac{\log(tB)}{tB}\right).$$

Let C be the implicit constant in the O-notation above. If $|\tilde{f}_i - s(i)f_i| \geq 2t$, at least half of the values $(|X^{(\ell)}|)_{\ell \in [k]}$ are at least $2t$. For $t \geq 3/B$ it thus follows by a union bound that

$$\Pr[|\tilde{f}_i - s(i)f_i| \geq 2t] \leq 2\binom{k}{\lceil k/2 \rceil}\left(C\frac{\log(tB)}{tB}\right)^{\lceil k/2 \rceil} \leq 2\left(4C\frac{\log(tB)}{tB}\right)^{\lceil k/2 \rceil}. \tag{4}$$

If $\alpha = O(1)$ is chosen sufficiently large it thus holds that

$$\begin{aligned}\int_{\alpha/B}^{\infty} \Pr[|\tilde{f}_i - s(i)f_i| \geq t]\,dt &= 2\int_{\alpha/(2B)}^{\infty} \Pr[|\tilde{f}_i - s(i)f_i| \geq 2t]\,dt \\ &\leq \frac{4}{B}\int_{\alpha/2}^{\infty}\left(4C\frac{\log(t)}{t}\right)^{\lceil k/2 \rceil} dt \\ &\leq \frac{1}{B2^k} \leq \frac{1}{B\sqrt{k}}.\end{aligned}$$

Here the first inequality uses eq. (4) and a change of variable. The second inequality uses that $\left(4C\frac{\log t}{t}\right)^{\lceil k/2 \rceil} \leq (C'/t)^{2k/5}$ for some constant C' followed by a calculation of the integral. Now,

$$\mathbb{E}[|\tilde{f}_i - s(i)f_i|] = \int_0^{\infty} \Pr[|\tilde{f}_i - s(i)f_i| \geq t]\,dt,$$

so for our upper bound it therefore suffices to show that $\int_0^{\alpha/B} \Pr[|\tilde{f}_i - s(i)f_i| \geq t]\,dt = O\left(\frac{1}{B\sqrt{k}}\right)$. For this we need the following claim:

Claim 4.6. *Let $I \subset \mathbb{R}$ be the closed interval centered at the origin of length $2t$, i.e., $I = [-t, t]$. Suppose that $0 < t \leq \frac{1}{2B}$. For $\ell \in [k]$, $\Pr[X^{(\ell)} \in I] = \Omega(tB)$.*

Proof. Note that $\Pr[X_1^{(\ell)} = 0] \geq \Pr[\bigwedge_{j \in N_1}(h_\ell(j) \neq h_\ell(i))] = (1 - \frac{1}{B})^{N_1} = \Omega(1)$. Secondly $\mathrm{Var}[X_2^{(\ell)}] = (\frac{1}{B} - \frac{1}{B^2})\sum_{j \in N_2} f_j^2 \leq \frac{1}{B^2}$. Using that $X_1^{(\ell)}$ and $X_2^{(\ell)}$ are independent and Lemma 4.5 with $\sigma^2 = \mathrm{Var}[X_2^{(\ell)}]$, it follows that $\Pr[X^{(\ell)} \in I] = \Omega\left(\Pr[X_2^{(\ell)} \in I]\right) = \Omega(tB)$. □

Let us now show how to use the claim to establish the desired upper bound. For this let $0 < t \leq \frac{1}{2B}$ be fixed. If $|\tilde{f}_i - s(i)f_i| \geq t$, at least half of the values $(X^{(\ell)})_{\ell \in [k]}$ are at least t or at most $-t$. Let us focus on bounding the probability that at least half are at least t, the other bound being symmetric giving an extra factor of 2 in the probability bound. By symmetry and Claim 4.6,

$\Pr[X^{(\ell)} \geq t] = \frac{1}{2} - \Omega(tB)$. For $\ell \in [k]$ we define $Y_\ell = [X^{(\ell)} \geq t]$, and we put $S = \sum_{\ell \in [k]} Y_\ell$. Then $\mathbb{E}[S] = k\left(\frac{1}{2} - \Omega(tB)\right)$. If at least half of the values $(X^{(\ell)})_{\ell \in [k]}$ are at least t then $S \geq k/2$. By Hoeffding's inequality (Theorem 2.6) we can bound the probability of this event by

$$\Pr[S \geq k/2] = \Pr[S - \mathbb{E}[S] = \Omega(ktB)] = \exp(-\Omega(kt^2B^2)).$$

It follows that $\Pr[|\tilde{f}_i - s(i)f_i| \geq t] \leq 2\exp(-\Omega(kt^2B^2))$. Thus

$$\begin{aligned}\int_0^{\alpha/B} \Pr[|\tilde{f}_i - s(i)f_i| \geq t]\, dt &\leq \int_0^{\frac{1}{2B}} 2\exp(-\Omega(kt^2B^2))\, dt + \int_{\frac{1}{2B}}^{\alpha/B} 2\exp(-\Omega(k))\, dt \\ &\leq \frac{1}{B\sqrt{k}} \int_0^{\sqrt{k}/2} \exp(-t^2)\, dt + \frac{2\alpha \exp(-\Omega(k))}{B} = O\left(\frac{1}{B\sqrt{k}}\right).\end{aligned}$$

Here the second inequality used a change of variable. The proof of the upper bound is complete.

Lower Bound Fix $\ell \in [k]$ and let $M_1 = [B\log k] \setminus \{i\}$ and $M_2 = [n] \setminus ([B\log k] \cup \{i\})$. Write

$$S := \sum_{j \in M_1} [h_\ell(j) = h_\ell(i)] s_\ell(j) f_j + \sum_{j \in M_2} [h_\ell(j) = h_\ell(i)] s_\ell(j) f_j := S_1 + S_2.$$

We also define $J := \{j \in M_1 : h_\ell(j) = h_\ell(i)\}$. Let $I \subseteq \mathbb{R}$ be the closed interval around $s_\ell(i)f_i$ of length $\frac{1}{B\sqrt{k}\log k}$. We now upper bound the probability that $S \in I$ conditioned on the value of S_2. To ease the notation, the conditioning on S_2 has been left out in the notation to follow. Note first that

$$\Pr[S \in I] = \sum_{r=0}^{|M_1|} \Pr[S \in I \mid |J| = r] \cdot \Pr[|J| = r]. \tag{5}$$

For a given $r \geq 1$ we now proceed to bound $\Pr[S \in I \mid |J| = r]$. This probability is the same as the probability that $S_2 + \sum_{j \in R} \sigma_j f_j \in I$, where $R \subseteq M_1$ is a uniformly random r-subset and the σ_j's are independent Rademachers. Suppose that we sample the elements from R as well as the corresponding signs $(\sigma_i)_{i \in R}$ sequentially, and let us condition on the values and signs of the first $r-1$ sampled elements. At this point at most $\frac{B\log k}{\sqrt{k}} + 1$ possible samples for the last element in R can cause that $S \in I$. Indeed, the minimum distance between distinct elements of $\{f_j : j \in M_1\}$ is at least $1/(B\log k)^2$ and furthermore I has length $\frac{1}{B\sqrt{k}\log k}$. Thus, at most

$$\frac{1}{B\sqrt{k}\log k} \cdot (B\log k)^2 + 1 = \frac{B\log k}{\sqrt{k}} + 1$$

choices for the last element of R ensure that $S \in I$. For $1 \leq r \leq (B\log k)/2$ we can thus upper bound

$$\Pr[S \in I \mid |J| = r] \leq \frac{\frac{B\log k}{\sqrt{k}} + 1}{|M_1| - r + 1} \leq \frac{2}{\sqrt{k}} + \frac{2}{B\log k} \leq \frac{3}{\sqrt{k}}.$$

Note that $\mu := \mathbb{E}[|J|] \leq \log k$ so for $B \geq 6$, it holds that

$$\Pr[|J| \geq (B\log k)/2] \leq \Pr\left[|J| \geq \mu\frac{B}{2}\right] \leq \Pr\left[|J| \geq \mu\left(1 + \frac{B}{3}\right)\right] \leq \exp\left(-\mu h(B/3)\right) = k^{-\Omega(h(B/3))},$$

where the last inequality follows from the Chernoff bound of Theorem 2.5. Thus, if we assume that B is larger than a sufficiently large constant, then $\Pr[|J| \geq B\log k/2] \leq k^{-1}$. Finally, $\Pr[|J| = 0] = (1-1/B)^{B\log k} \leq k^{-1}$. Combining the above, we can continue the bound in (5) as follows.

$$\begin{aligned}\Pr[S \in I] \leq& \Pr[|J| = 0] + \sum_{r=1}^{(B\log k)/2} \Pr[S \in I \mid |J| = r] \cdot \Pr[|J| = r] \\ &+ \sum_{r=(B\log k)/2+1}^{|M_1|} \Pr[|J| = r] = O\left(\frac{1}{\sqrt{k}}\right), \qquad (6)\end{aligned}$$

which holds even after removing the conditioning on S_2. We now show that with probability $\Omega(1)$ at least half the values $(X^{(\ell)})_{\ell \in [k]}$ are at least $\frac{1}{2B\sqrt{k}\log k}$. Let p_0 be the probability that $X^{(\ell)} \geq \frac{1}{2B\sqrt{k}\log k}$. This probability does not depend on $\ell \in [k]$ and by symmetry and (6), $p_0 = 1/2 - O(1/\sqrt{k})$. Define the function $f : \{0,\dots,k\} \to \mathbb{R}$ by

$$f(t) = \binom{k}{t} p_0^t (1-p_0)^{k-t}.$$

Then $f(t)$ is the probability that exactly t of the values $(X^{(\ell)})_{\ell \in [k]}$ are at least $\frac{1}{B\sqrt{k}\log k}$. Using that $p_0 = 1/2 - O(1/\sqrt{k})$, a simple application of Stirling's formula gives that $f(t) = \Theta\left(\frac{1}{\sqrt{k}}\right)$ for $t = \lceil k/2 \rceil, \dots, \lceil k/2 + \sqrt{k} \rceil$ when k is larger than some constant C. It follows that with probability $\Omega(1)$ at least half of the $(X^{(\ell)})_{\ell \in [k]}$ are at least $\frac{1}{B\sqrt{k}\log k}$ and in particular

$$\mathbb{E}[|\tilde{f}_i - f_i|] = \Omega\left(\frac{1}{B\sqrt{k}\log k}\right).$$

Finally we handle the case where $k \leq C$. It follows from simple calculations (e.g., using Lemma 4.2) that $X^{(\ell)} = \Omega(1/B)$ with probability $\Omega(1)$. Thus this happens for all $\ell \in [k]$ with probability $\Omega(1)$ and in particular $\mathbb{E}[|\tilde{f}_i - f_i|] = \Omega(1/B)$, which is the desired for constant k. $\square$

5 Learned Count-Sketch for Zipfians

We now proceed to analyze the learned Count-Sketch algorithm. In Section 5.1 we estimate the expected error when using a single hash function and in Section 5.2 we show that the expected error only increases when using more hash functions. Recall that we assume that the number of buckets B_h used to store the heavy hitters that $B_h = \Theta(B - B_h) = \Theta(B)$.

5.1 One hash function

By taking $B_1 = B_h = \Theta(B)$ and $B_2 = B - B_h = \Theta(B)$ in the theorem below, the result on L-CS for $k = 1$ claimed in Table 1 follows immediately.

Theorem 5.1. *Let $h : [n] \setminus [B_1] \to [B_2]$ and $s : [n] \to \{-1, 1\}$ be truly random hash functions where $n, B_1, B_2 \in \mathbb{N}$ and*[6] *$n - B_1 \geq B_2 \geq B_1$. Define the random variable $\tilde{f}_i = \sum_{j=B_1+1}^{n} [h(j) = h(i)] s(j) f_j$*

[6]The first inequality is the standard assumption that we have at least as many items as buckets. The second inequality says that we use at least as many buckets for non-heavy items as for heavy items (which doesn't change the asymptotic space usage).

for $i \in [n] \setminus [B_1]$*. Then*

$$\mathbb{E}[|\tilde{f}_i - s(i)f_i|] = \Theta\left(\frac{\log \frac{B_2+B_1}{B_1}}{B_2}\right)$$

Proof. Let $N_1 = [B_1 + B_2] \setminus ([B_1] \cup \{i\})$ and $N_2 = [n] \setminus ([B_1 + B_2] \cup \{i\})$. Let $X_1 = \sum_{j \in N_1} [h(j) = h(i)] s(j) f_j$ and $X_2 = \sum_{j \in N_2} [h(j) = h(i)] s(j) f_j$. By the triangle inequality and linearity of expectation,

$$\mathbb{E}[|X_1|] = O\left(\frac{\log \frac{B_2+B_1}{B_1}}{B_2}\right).$$

Moreover, it follows directly from Lemma 4.1 that $\mathbb{E}\left[|X_2|\right] = O\left(\frac{1}{B_2}\right)$. Thus

$$\mathbb{E}[|\tilde{f}_i - s(i)f_i|] \le \mathbb{E}[|X_1|] + \mathbb{E}[|X_2|] = O\left(\frac{\log \frac{B_2+B_1}{B_1}}{B_2}\right),$$

as desired. For the lower bound on $\mathbb{E}\left[\left|\tilde{f}_i - s(i)f_i\right|\right]$ we apply Lemma 4.2 with $I = N_1$ to obtain that,

$$\mathbb{E}\left[\left|\tilde{f}_i - s(i)f_i\right|\right] \ge \frac{1}{2B_2}\left(1 - \frac{1}{B_2}\right)^{|N_1|-1} \sum_{i \in N_1} f_i = \Omega\left(\frac{\log \frac{B_2+B_1}{B_1}}{B_2}\right).$$

□

Corollary 5.2. *Let* $h : [n] \setminus [B_h] \to [B - B_h]$ *and* $s : [n] \to \{-1, 1\}$ *be truly random hash functions where* $n, B, B_h \in \mathbb{N}$ *and* $B_h = \Theta(B) \le B/2$*. Define the random variable* $\tilde{f}_i = \sum_{j=B_h+1}^{n} [h(j) = h(i)] s(j) f_j$ *for* $i \in [n] \setminus [B_h]$*. Then* $\mathbb{E}[|\tilde{f}_i - s(i)f_i|] = \Theta(1/B)$*.*

Remark 5.3. The upper bounds of Theorem 5.1 and Corollary 5.2 hold even without the assumption of fully random hashing. In fact, we only require that h and s are 2-independent. Indeed Lemma 4.1 holds even when the Rademachers are 2-independent (the proof is the same). Moreover, we need h to be 2-independent as we condition on $h(i)$ in our application of Lemma 4.1. With 2-independence the variables $[h(j) = h(i)]$ for $j \ne i$ are then Bernoulli variables taking value 1 with probability $1/B_2$.

5.2 More hash functions

We now show that, like for Count-Sketch, using more hash functions does not decrease the expected error. We first state the Littlewood-Offord lemma as strengthened by Erdős.

Theorem 5.4 (Littlewood-Offord [LO39], Erdős [Erd45])**.** *Let* $a_1, \ldots, a_n \in \mathbb{R}$ *with* $|a_i| \ge 1$ *for* $i \in [n]$*. Let further* $\sigma_1, \ldots, \sigma_n \in \{-1, 1\}$ *be random variables with* $\Pr[\sigma_i = 1] = \Pr[\sigma_i = -1] = 1/2$ *and define* $S = \sum_{i=1}^{n} \sigma_i a_i$*. For any* $v \in \mathbb{R}$ *it holds that* $\Pr[|S - v| \le 1] = O(1/\sqrt{n})$*.*

Setting $B_1 = B_h = \Theta(B)$ and $B_2 = B - B_2 = \Theta(B)$ in the theorem below gives the final bound from Table 1 on L-CS with $k \ge 3$.

Theorem 5.5. *Let $n \geq B_1 + B_2 \geq 2B_1$, $k \geq 3$ odd, and $h_1, \dots, h_k : [n] \setminus [B_1] \to [B_2/k]$ and $s_1, \dots, s_k : [n] \setminus [B_1] \to \{-1, 1\}$ be independent and truly random. Define the random variable $\tilde{f}_i = \mathsf{median}_{\ell \in [k]} \left(\sum_{j \in [n] \setminus [B_1]} [h_\ell(j) = h_\ell(i)] s_\ell(j) f_j \right)$ for $i \in [n] \setminus [B_1]$. Then*

$$\mathbb{E}[|\tilde{f}_i - s(i) f_i|] = \Omega\left(\frac{1}{B_2}\right).$$

Proof. Like in the proof of the lower bound of Theorem 4.4 it suffices to show that for each i the probability that the sum $S_\ell := \sum_{j \in [n] \setminus ([B_1] \cup \{i\})} [h_\ell(j) = h_\ell(i)] s_\ell(j) f_j$ lies in the interval $I = [-1/(2B_2), 1/(2B_2)]$ is $O(1/\sqrt{k})$. Then at least half the $(S_\ell)_{\ell \in [k]}$ are at least $1/(2B_2)$ with probability $\Omega(1)$ by an application of Stirling's formula, and it follows that $\mathbb{E}[|\tilde{f}_i - s(i) f_i|] = \Omega(1/B_2)$.

Let $\ell \in [k]$ be fixed, $N_1 = [2B_2] \setminus ([B_2] \cup \{i\})$, and $N_2 = [n] \setminus (N_1 \cup \{i\})$, and write

$$S_\ell = \sum_{j \in N_1} [h_\ell(j) = h_\ell(i)] s_\ell(j) f_j + \sum_{j \in N_2} [h_\ell(j) = h_\ell(i)] s_\ell(j) f_j := X_1 + X_2.$$

Now condition on the value of X_2. Letting $J = \{j \in N_1 : h_\ell(j) = h_\ell(i)\}$ it follows by Theorem 5.4 that

$$\Pr[S_\ell \in I \mid X_2] = O\left(\sum_{J' \subseteq N_1} \frac{\Pr[J = J']}{\sqrt{|J'| + 1}} \right) = O\left(\Pr[|J| < k/2] + 1/\sqrt{k} \right).$$

An application of Chebyshev's inequality gives that $\Pr[|J| < k/2] = O(1/k)$, so $\Pr[S_\ell \in I] = O(1/\sqrt{k})$. Since this bound holds for any possible value of X_2 we may remove the conditioning and the desired result follows. $\square$

Remark 5.6. The bound above is probably only tight for $B_1 = \Theta(B_2)$. Indeed, we know that it cannot be tight for all $B_1 \leq B_2$ since when B_1 becomes very small, the bound from the standard Count-Sketch with $k \geq 3$ takes over — and this is certainly worse than the bound in the theorem. It is an interesting open problem (that requires a better anti-concentration inequality than the Littlewood-Offord lemma) to settle the correct bound when $B_1 \ll B_2$.

5.3 Learned Count-Sketch using a noisy heavy hitter oracle

In [HIKV19] it was demonstrated that if the heavy hitter oracle is noisy, misclassifying an item with probability δ, then the expected error incurred by Count-Min for Zipfians is

$$O\left(\frac{1}{\log n} \frac{\delta^2 \ln^2 B_h + \ln^2(n/B_h)}{B - B_h} \right).$$

Here B_h is the number of buckets used to store the heavy hitters and B is the total number of buckets. Taking $B_h = \Theta(B) = \Theta(B - B_h)$, this bound becomes $O\left(\frac{\delta^2 \ln^2 B + \ln^2(n/B)}{B \log n}\right)$. As δ varies in $[0, 1]$, this interpolates between the expected error incurred in respectively the learned case with a perfect heavy hitter oracle and the classic case. In particular it is enough to assume that $\delta = O(\ln(n/B)/\ln(B))$ in order to obtain the results in the idealized case with a perfect oracle.

We now provide a similar analysis for the learned Count-Sketch. More precisely we assume that we allocate B_h buckets to the heavy hitters and $B - B_h$ to the lighter items. We moreover assume access to a heavy hitter oracle $\mathbf{HH}_\delta$ such that for each $i \in [n]$, $\Pr[\mathbf{HH}_\delta(i) \neq \mathbf{HH}_0(i)] \leq \delta$, where $\mathbf{HH}_0$ is a perfect heavy hitter oracle that correctly classifies the B_h heaviest items.

Theorem 5.7. *Learned Count-Sketch with a single hash functions, a heavy hitter oracle $\boldsymbol{HH}_\delta$, $B_h = \Theta(B)$ bins allocated to store the B_h items classified as heavy and $B - B_h = \Theta(B)$ bins allocated to a Count-Sketch of the remaining items, incurs an expected error of*

$$O\left(\frac{(\delta \log B + \log(n/B))(1 + \delta \log B)}{B \log n}\right).$$

Proof. Let $h : [n] \to [B - B_h]$ and $s : [n] \to \{-1, 1\}$ be the hash functions used for the Count-Sketch. In the analysis to follow, it is enough to assume that they are 2-independent. Suppose item i is classified as non-heavy. For $j \in [n]$, let $\eta_j = [h(j) = h(i)]$, and let α_j be the indicator for item j being classified as non-heavy. Then

$$|\tilde{f}_i - f_i| = \left| \sum_{j \in [n] \setminus \{i\}} \alpha_j \eta_j s(j) f_j \right| \le \sum_{j \in [B_h] \setminus \{i\}} \alpha_j \eta_j f_j + \left| \sum_{j \in [n] \setminus (B_h \cup \{i\})} \alpha_j \eta_j s(j) f_j \right| := S_1 + S_2$$

Note that $\mathbb{E}[S_1] = O\left(\frac{\delta \log B_h}{B - B_h}\right) = O\left(\frac{\delta \log B}{B}\right)$. For S_2, we let $p_j = \Pr[\alpha_j \eta_j = 1] \le \frac{1}{B - B_h} = O(\frac{1}{B})$. Then

$$\mathbb{E}[S_2] \le (\mathbb{E}[S_2^2])^{1/2} = \left(\sum_{j \in [n] \setminus (B_h \cup \{i\})} p_j f_j^2 \right)^{1/2} = O\left(\frac{1}{B}\right),$$

using that $\mathbb{E}[s(i)s(j)] = 0$ for $i \neq j$ as s is 2-independent. It follows that $\mathbb{E}[|\tilde{f}_i - f_i|] = O\left(\frac{1 + \delta \log B}{B}\right)$, given that item i is classified as non-heavy. Let $N = \sum_{i \in [n]} f_i = \Theta(\log n)$. As the probability of item $i \in [B_h]$ being classified as non-heavy is at most δ, the the expected error is upper bounded by

$$\frac{1}{N}\left(\delta \sum_{j \in [B_h] \setminus \{i\}} f_i + \sum_{j \in [n] \setminus (B_h \cup \{i\})} f_i \right) \cdot O\left(\frac{1 + \delta \log B}{B}\right) = O\left(\frac{(\delta \log B + \log(n/B))(1 + \delta \log B)}{B \log n}\right),$$

as desired. ◻

We see that with $\delta = 1$, we recover the bound of $\frac{\log B}{B}$ presented in Table 1 for the classic Count-Sketch. On the other hand, it is enough to assume that $\delta = O(1/\log B)$ in order to obtain the bound of $O\left(\frac{\log(n/B)}{B \log n}\right)$, which is what we obtain with a perfect heavy hitter oracle.

6 Experiments

In this section, we provide the empirical evaluation of CountMin, CountSketch and their learned counterparts under Zipfian distribution. Our empirical results complement the theoretical analysis provided earlier in this paper.

Experiment setup. We consider a *synthetic* stream of $n = 10K$ items where the frequencies of the items follow the standard Zipfian distribution (i.e., with $\alpha = 1$). To be consistent with our assumption in our theoretical analysis, we scale the frequencies so that the frequency of item i is $1/i$. In our experiments, we vary the values of the number of buckets (B) and the number of rows in the sketch (k) as well as the number of predicted heavy items in the learned sketches. We remark that in this section we assume that the heavy hitter oracle predicts *without errors*.

We run each experiment 20 times and take the average of the estimation error defined in eq. (2).

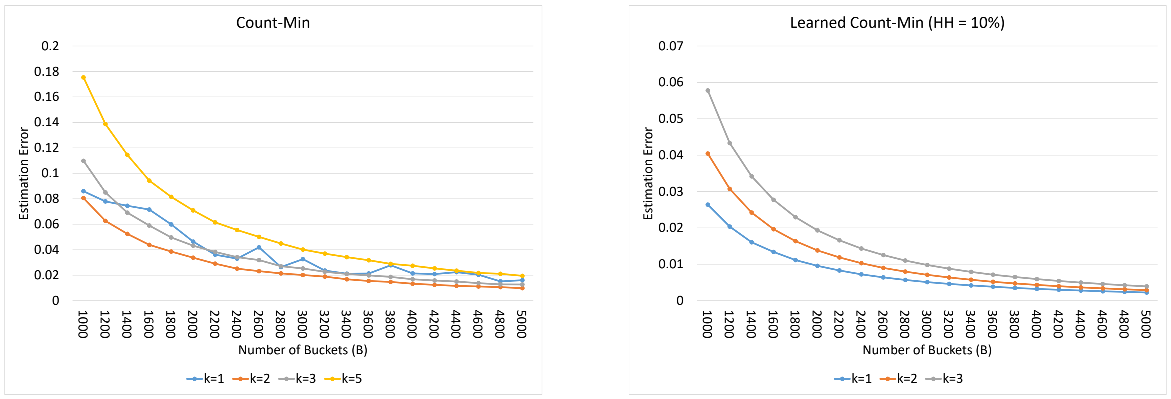

Figure 2: The performance of (Learned) Count-Min with different number of rows.

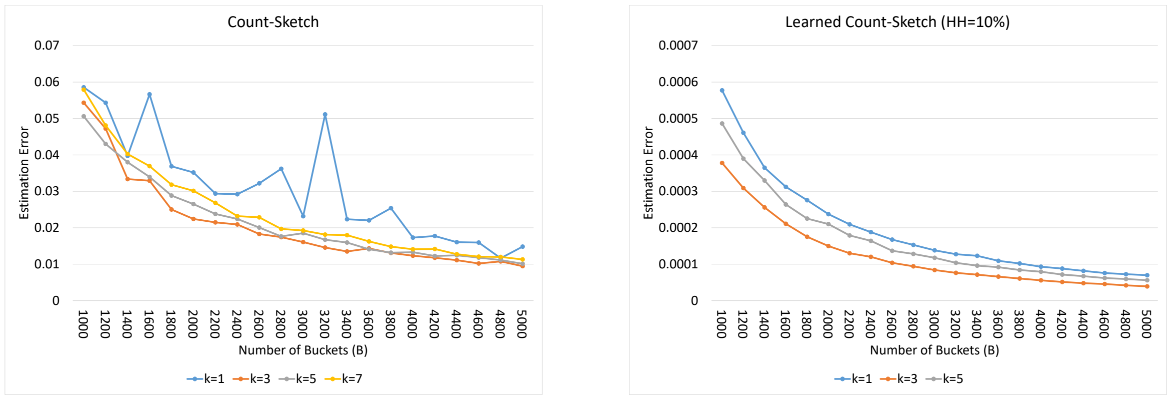

Figure 3: The performance of (Learned) Count-Sketch with different number of rows.

Sketches with the same number of buckets but different shapes. Here, we compare the empirical performances of both standard and learned variants of Count-Min and Count-Sketch with varying choices for the parameter. More precisely, we fix the sketch size and vary the number of rows (i.e., number of hash functions) in the sketch.

As predicted in our theoretical analysis, Figures 2 and 3 show that setting the number of rows to some constant larger than 1 for standard CM and CS, leads to a smaller estimation error as we increase the size of the sketch. In contrast, in the learned variant, the average estimation error increases in k being smallest for $k = 1$, as was also predicted by our analysis.

B	CM ($k = 1$)	CM ($k = 2$)	L-CM	CS ($k = 1$)	CS ($k = 3$)	L-CS
1000	0.085934	0.080569	0.026391	0.058545	0.054315	0.000577138
1200	0.077913	0.06266	0.020361	0.054322	0.047214	0.000460688
1400	0.074504	0.052464	0.016036	0.03972	0.033348	0.00036492
1600	0.071528	0.043798	0.01338	0.056626	0.032925	0.000312238
1800	0.059898	0.038554	0.011142	0.036881	0.025003	0.000275648
2000	0.046389	0.033746	0.009556	0.035172	0.022403	0.000237371
2200	0.036082	0.029059	0.008302	0.029388	0.02148	0.000209376
2400	0.032987	0.025135	0.007237	0.02919	0.020913	0.00018811
2600	0.041896	0.023157	0.006399	0.032195	0.018271	0.00016743
2800	0.026351	0.021402	0.005694	0.036197	0.017431	0.000152933
3000	0.032624	0.020155	0.005101	0.023175	0.016068	0.000138081
3200	0.023614	0.018832	0.004599	0.051132	0.01455	0.000127445
3400	0.021151	0.016769	0.004196	0.022333	0.013503	0.000122947
3600	0.021314	0.015429	0.003823	0.022012	0.014316	0.000109171
3800	0.027798	0.014677	0.003496	0.025378	0.013082	0.000102035
4000	0.021407	0.013279	0.00322	0.017303	0.012312	0.0000931
4200	0.020883	0.012419	0.002985	0.017719	0.011748	0.0000878
4400	0.022383	0.011608	0.002769	0.016037	0.011097	0.0000817
4600	0.020378	0.011151	0.002561	0.015941	0.010202	0.0000757
4800	0.015114	0.010612	0.002406	0.011642	0.010757	0.0000725
5000	0.01603	0.009767	0.002233	0.014829	0.009451	0.0000698

Table 2: The estimation error of different sketching methods under Zipfian distribution. In this example, the number of unique items n is equal to $10K$. In the learned variants, number of rows, k, is equal to 1 and the *perfect* heavy hitter oracles detect top c-frequent items where $c = B/10$.

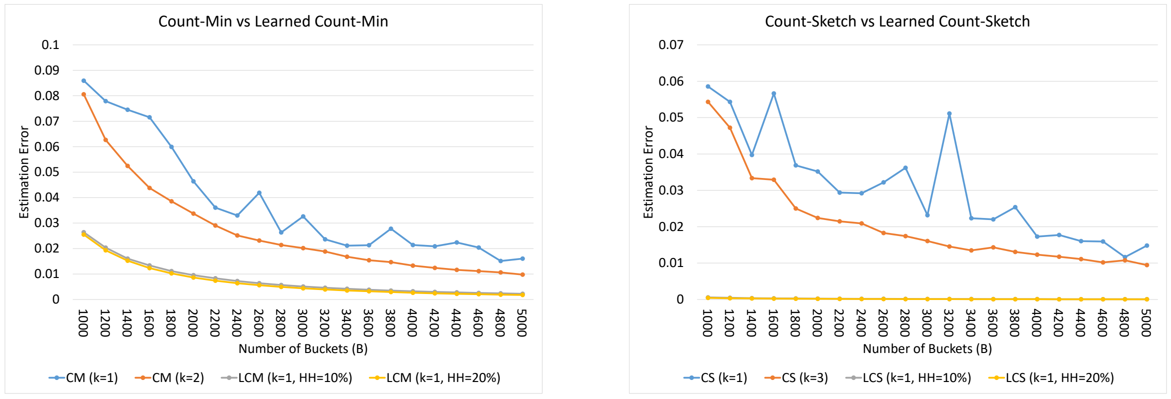

Figure 4: The comparison of the performance of learned and standard variants of Count-Min and Count-Sketch.

Learned vs. Standard Sketches. In Figure 4, we compare the performance of learned variants of Count-Min and Count-Sketch with the standard Count-Min and Count-Sketch. To be fair, we assume that each bucket that is assigned a heavy hitter consumes two bucket of memory:

one for counting the number of times the heavy item appears in the stream and one for indexing the heavy item in the data structure.

We observe that the learned variants of Count-Min and Count-Sketch significantly improve upon the estimation error of their standard "non-learned" variants. We note that the estimation errors for the learned Count-Sketches in Figure 4 are not zero but very close to zero; see Table 2 for the actual values.

References

[ABL$^+$17] Daniel Anderson, Pryce Bevan, Kevin Lang, Edo Liberty, Lee Rhodes, and Justin Thaler. A high-performance algorithm for identifying frequent items in data streams. In *Proceedings of the 2017 Internet Measurement Conference*, pages 268–282, 2017.

[ACC$^+$11] Nir Ailon, Bernard Chazelle, Kenneth L Clarkson, Ding Liu, Wolfgang Mulzer, and C Seshadhri. Self-improving algorithms. *SIAM Journal on Computing*, 40(2):350–375, 2011.

[ACE$^+$20] Antonios Antoniadis, Christian Coester, Marek Elias, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. *arXiv preprint arXiv:2003.02144*, 2020.

[ADJ$^+$20] Spyros Angelopoulos, Christoph Dürr, Shendan Jin, Shahin Kamali, and Marc Renault. Online computation with untrusted advice. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[AKL$^+$19] Daniel Alabi, Adam Tauman Kalai, Katrina Ligett, Cameron Musco, Christos Tzamos, and Ellen Vitercik. Learning to prune: Speeding up repeated computations. In *Conference on Learning Theory*, 2019.

[BCI$^+$17] Vladimir Braverman, Stephen R Chestnut, Nikita Ivkin, Jelani Nelson, Zhengyu Wang, and David P Woodruff. Bptree: an ℓ_2 heavy hitters algorithm using constant memory. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 361–376, 2017.

[BCIW16] Vladimir Braverman, Stephen R Chestnut, Nikita Ivkin, and David P Woodruff. Beating countsketch for heavy hitters in insertion streams. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 740–753, 2016.

[BDSV18] Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In *International Conference on Machine Learning*, pages 353–362, 2018.

[BDV18] Maria-Florina Balcan, Travis Dick, and Ellen Vitercik. Dispersion for data-driven algorithm design, online learning, and private optimization. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 603–614. IEEE, 2018.

[BDW18] Arnab Bhattacharyya, Palash Dey, and David P Woodruff. An optimal algorithm for ℓ_1-heavy hitters in insertion streams and related problems. *ACM Transactions on Algorithms (TALG)*, 15(1):1–27, 2018.

[Ben62] George Bennett. Probability inequalities for the sum of independent random variables. *Journal of the American Statistical Association*, 57(297):33–45, 1962.

[BICS10] Radu Berinde, Piotr Indyk, Graham Cormode, and Martin J Strauss. Space-optimal heavy hitters with strong error bounds. *ACM Transactions on Database Systems (TODS)*, 35(4):1–28, 2010.

[BJPD17] Ashish Bora, Ajil Jalal, Eric Price, and Alexandros G Dimakis. Compressed sensing using generative models. In *International Conference on Machine Learning*, pages 537–546, 2017.

[CCFC02] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.

[CGP20] Edith Cohen, Ofir Geri, and Rasmus Pagh. Composable sketches for functions of frequencies: Beyond the worst case. *arXiv preprint arXiv:2004.04772*, 2020.

[CGT$^+$19] Shuchi Chawla, Evangelia Gergatsouli, Yifeng Teng, Christos Tzamos, and Ruimin Zhang. Learning optimal search algorithms from data. *arXiv preprint arXiv:1911.01632*, 2019.

[CH08] Graham Cormode and Marios Hadjieleftheriou. Finding frequent items in data streams. *Proceedings of the VLDB Endowment*, 1(2):1530–1541, 2008.

[CH10] Graham Cormode and Marios Hadjieleftheriou. Methods for finding frequent items in data streams. *The VLDB Journal*, 19(1):3–20, 2010.

[Che52] Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23(4):493–507, 1952.

[CM05a] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[CM05b] Graham Cormode and Shan Muthukrishnan. Summarizing and mining skewed data streams. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pages 44–55. SIAM, 2005.

[DIRW19] Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Learning sublinear-time indexing for nearest neighbor search. *arXiv preprint arXiv:1901.08544*, 2019.

[Erd45] Paul Erdös. On a lemma of littlewood and offord. *Bulletin of the American Mathematical Society*, 51(12):898–902, 1945.

[GP19] Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2319–2327, 2019.

[GR17] Rishi Gupta and Tim Roughgarden. A pac approach to application-specific algorithm selection. *SIAM Journal on Computing*, 46(3):992–1017, 2017.

[HIKV19] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In *International Conference on Learning Representations*, 2019.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[IVY19] Piotr Indyk, Ali Vakilian, and Yang Yuan. Learning-based low-rank approximations. In *Advances in Neural Information Processing Systems*, pages 7400–7410, 2019.

[JLL$^+$20] Tanqiu Jiang, Yi Li, Honghao Lin, Yisong Ruan, and David P. Woodruff. Learning-augmented data stream algorithms. In *International Conference on Learning Representations*, 2020.

[KBC$^+$18] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, pages 489–504, 2018.

[KDZ$^+$17] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017.

[Kod19] Rohan Kodialam. Optimal algorithms for ski rental with soft machine-learned predictions. *arXiv preprint arXiv:1903.00092*, 2019.

[LLMV20] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1859–1877. SIAM, 2020.

[LNNT16] Kasper Green Larsen, Jelani Nelson, Huy L Nguyên, and Mikkel Thorup. Heavy hitters via cluster-preserving clustering. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 61–70. IEEE, 2016.

[LO39] John Edensor Littlewood and Albert C Offord. On the number of real roots of a random algebraic equation. ii. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 35, pages 133–148. Cambridge University Press, 1939.

[LV18] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *International Conference on Machine Learning*, pages 3302–3311, 2018.

[M$^+$05] Shanmugavelayutham Muthukrishnan et al. Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science*, 1(2):117–236, 2005.

[MAEA05] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *International Conference on Database Theory*, pages 398–412. Springer, 2005.

[MG82] Jayadev Misra and David Gries. Finding repeated elements. *Science of computer programming*, 2(2):143–152, 1982.

[Mit18] Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *Advances in Neural Information Processing Systems*, pages 464–473, 2018.

[Mit20] Michael Mitzenmacher. Scheduling with predictions and the price of misprediction. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[MM02] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pages 346–357. Elsevier, 2002.

[MP14] Gregory T Minton and Eric Price. Improved concentration bounds for count-sketch. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 669–686. Society for Industrial and Applied Mathematics, 2014.

[MPB15] Ali Mousavi, Ankit B Patel, and Richard G Baraniuk. A deep learning approach to structured signal recovery. In *Communication, Control, and Computing (Allerton), 2015 53rd Annual Allerton Conference on*, pages 1336–1343. IEEE, 2015.

[PSK18] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In *Advances in Neural Information Processing Systems*, pages 9661–9670, 2018.

[RKA16] Pratanu Roy, Arijit Khan, and Gustavo Alonso. Augmented sketch: Faster and more accurate stream processing. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1449–1463, 2016.

[Roh20] Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1834–1845. SIAM, 2020.

[WLKC16] Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. Learning to hash for indexing big data - a survey. *Proceedings of the IEEE*, 104(1):34–57, 2016.

A Count-Min for General Zipfian (with $\alpha \neq 1$)

In this appendix we provide an analysis of the expected error with Count-Min in the case with input coming from a general Zipfian distribution, i.e., $f_i \propto \frac{1}{i^\alpha}$, for some fixed $\alpha > 0$. By scaling we can assume that $f_i = \frac{1}{i^\alpha}$ with no loss of generality. Our results on the expected error is presented in Table 3 below. We start by analyzing the standard Count-Min sketch that does not have access to a machine learning oracle.

A.1 Standard Count-Min

We begin by considering the case $\alpha < 1$, in which case we have the following result.

Theorem A.1. *Let $0 < \alpha < 1$ be fixed and $f_i = 1/i^\alpha$ for $i \in [n]$. Let $n, B, k \in \mathbb{N}$ with $k \geq 1$ and $B \leq n/k$. Let further $h_1, \dots, h_k : [n] \to [B]$ be independent and truly random hash functions. For $i \in [n]$ define the random variable $\tilde{f}_i = \min_{\ell \in [k]} \left(\sum_{j \in [n]} [h_\ell(j) = h_\ell(i)] f_j \right)$. For any $i \in [n]$ it holds that $\mathbb{E}[|\tilde{f}_i - f_i|] = \Theta\left(\frac{n^{1-\alpha}}{B}\right)$.*

	$k=1$	$k>1$
CM, $\alpha<1$	$\Theta\left(\frac{n^{2-2\alpha}}{B}\right)$	$\Theta\left(\frac{kn^{2-2\alpha}}{B}\right)$
CM, $\alpha>1$	$O\left(\frac{1}{B}\right)$	$(O(1))^k(\log(B))^{k/\alpha+1}\cdot\left(\frac{k^k}{B^k}+\frac{k^\alpha}{B^\alpha}\right)$ and $\Omega\left(\frac{k^k}{B^k}+\frac{k^\alpha}{(B\log k)^\alpha}\right)$
L-CM, $\alpha<1$	$\Theta\left(\frac{n^{2-2\alpha}}{B}\right)$	$\Omega\left(\frac{n^{2-2\alpha}}{B}\right)$
L-CM, $\alpha>1$	$\Theta\left(B^{1-2\alpha}\right)$	$\Omega\left(B^{1-2\alpha}\right)$

Table 3: The (scaled) expected errors $\mathrm{Err}(\mathcal{F},\tilde{\mathcal{F}}_{\mathcal{A}})=\sum_{i\in[n]}f_i|f_i-\tilde{f}_i|$ of classic and learned Count-Min with k hash functions when the input has a Zipfian distribution with exponent $\alpha\neq 1$. The expected errors can be found by normalizing with $\sum_{i\in[n]}f_i$ which is $\Theta(n^{1-\alpha})$ for $\alpha<1$ and $\Theta(1)$ for $\alpha>1$. We note that when $k>1$ is a constant, the upper and lower bounds for CM for $\alpha>1$ are within logarithmic factors of each other. In particular we obtain the combined bound of $\tilde{\Theta}\left(\frac{1}{B^k}+\frac{1}{B^k}\right)$ in this case, demonstrating that the bounds, even if they appear complicated, are almost tight.

We again note the phenomenon that with a *total* of B buckets, i.e., replacing B by B/k in the theorem, the expected error is $\Theta\left(\frac{kn^{1-\alpha}}{B}\right)$, which only increases as we use more hash functions.

Proof. For a fixed $\ell\in[k]$ we have that

$$\mathbb{E}\left[\sum_{j\in[n]\setminus\{i\}}[h_\ell(j)=h_\ell(i)]f_j\right]=\frac{1}{B}\sum_{j\in[n]\setminus\{i\}}\frac{1}{j^\alpha}=O\left(\frac{n^{1-\alpha}}{B}\right),$$

and so $\mathbb{E}[|\tilde{f}_i-f_i|]=O\left(\frac{n^{1-\alpha}}{B}\right)$.

For the lower bound, we define $N=[n]\setminus([B]\cup\{i\})$ and for $\ell\in[k]$, $X_\ell=\sum_{j\in N}[h_\ell(j)=h_\ell(i)]f_j$. Simple calculations yield that $\mathbb{E}[X_\ell]=\Theta\left(\frac{n^{1-\alpha}}{B}\right)$ and

$$\mathrm{Var}[X_\ell]=\begin{cases}\Theta\left(\frac{\log\left(\frac{n}{B}\right)}{B}\right), & \alpha=1/2,\\ \Theta\left(\frac{n^{1-2\alpha}}{B}\right), & \alpha<1/2,\\ \Theta\left(B^{-2\alpha}\right), & \alpha>1/2.\end{cases}$$

Using Bennett's inequality (Theorem 2.3), with $M=B^{-\alpha}$ we obtain that

$$\Pr[X_\ell\le\mathbb{E}[X_\ell]/2]\le\begin{cases}\exp\left(-\Omega(\log(n/B)h(\left(\frac{n}{B}\right)^{1/2}\frac{1}{\log(n/B)}))\right), & \alpha=1/2,\\ \exp\left(-\Omega\left(\left(\frac{n}{B}\right)^{1-2\alpha}h\left(\left(\frac{n}{B}\right)^{\alpha}\right)\right)\right), & \alpha<1/2,\\ \exp\left(-\Omega\left(h\left(\left(\frac{n}{B}\right)^{1-\alpha}\right)\right)\right), & \alpha>1/2.\end{cases}$$

Using that $n\ge kB$ and Remark 2.4 we in either case obtain that

$$\Pr[X_\ell\le\mathbb{E}[X_\ell]/2]=\exp\left(-\Omega(k^{1-\alpha}\log k)\right)=k^{-\Omega(k^{1-\alpha})}.$$

As the events $(X_\ell > \mathbb{E}[X_\ell]/2)_{\ell\in[k]}$ are independent, they happen simultaneously with probability $(1-k^{-\Omega(k^{1-\alpha})})^k = \Omega(1)$. If they all occur, then $|\tilde{f}_i - f_i| = \Omega\left(\frac{n^{1-\alpha}}{B}\right)$, so it follows that $\mathbb{E}[|\tilde{f}_i - f_i|] = \Omega\left(\frac{n^{1-\alpha}}{B}\right)$, as desired. □

Next, we consider the case $\alpha > 1$. In this case we have the following theorem where we obtain the result presented in Table 3 by replacing B with B/k.

Theorem A.2. *Let $\alpha > 1$ be fixed and $f_i = 1/i^\alpha$ for $i \in [n]$. Let $n, B, k \in \mathbb{N}$ with $k \geq 2$ and $B \leq n/k$. Let further $h_1, \dots, h_k : [n] \to [B]$ be independent and truly random hash functions. For $i \in [n]$ define the random variable $\tilde{f}_i = \min_{\ell\in[k]}\left(\sum_{j\in[n]}[h_\ell(j) = h_\ell(i)]f_j\right)$. For any $i \in [n]$ it holds that*

$$\mathbb{E}[|\tilde{f}_i - f_i|] \leq C^k(\log(B))^{k/\alpha+1} \cdot \left(\frac{1}{B^k} + \frac{1}{B^\alpha}\right),$$

for some constant C depending only on α. Furthermore, $\mathbb{E}[|\tilde{f}_i - f_i|] = \Omega\left(\frac{1}{B^k} + \frac{1}{(B\log k)^\alpha}\right)$.

Proof. Let us start by proving the lower bound. Let $N = [\lfloor B\log k\rfloor]$. With probability

$$\left(1-(1-1/B)^{|N\setminus\{i\}|}\right)^k \geq \left(1-e^{\frac{|N\setminus\{i\}|}{B}}\right)^k = \Omega(1)$$

it holds that for each $\ell \in [k]$ there exists $j \in N\setminus\{i\}$ such that $h_\ell(j) = h_\ell(i)$. In this case $|\tilde{f}_i - f_i| \geq \frac{1}{(B\log k)^\alpha}$, so it follows that also $\mathbb{E}[|\tilde{f}_i - f_i|] \geq \frac{1}{(B\log k)^\alpha}$. Note next that with probability $1/B^k$, $h_\ell(1) = h_\ell(i)$ for each $\ell \in [k]$. If this happens, $|\tilde{f}_i - f_i| \geq 1$, so it follows that $\mathbb{E}[|\tilde{f}_i - f_i|] \geq 1/B^k$ which is the second part of the lower bound.

Next we prove the upper bound. The technique is very similar to the proof of Theorem 3.1. We define $N_1 = [B]\setminus\{i\}$ and $N_2 = [n]\setminus([B]\cup\{i\})$. We further define $X_1^{(\ell)} = \sum_{j\in N_1}[h_\ell(j) = h_\ell(i)]f_j$ and $X_2^{(\ell)} = \sum_{j\in N_2}[h_\ell(j) = h_\ell(i)]f_j$ for $\ell \in [k]$. Note that for any $\ell \in [k]$, $\mathbb{E}[X_2^{(\ell)}] = O\left(\frac{1}{B^\alpha}\right)$, so it suffices to bound $\mathbb{E}[\min_{\ell\in[k]}(X_1^{(\ell)})]$. Let $t \geq 3/B^\alpha$ be given. A similar union bound to that given in the proof of Theorem 3.1 gives that for any $s \leq t$,

$$\Pr[X_1^{(\ell)} \geq t] \leq \binom{B}{t/s}\frac{1}{B^{t/s}} + \frac{1}{Bs^{1/\alpha}} \leq \left(\frac{es}{t}\right)^{t/s} + \frac{(t/s)^{1/\alpha}}{Bt^{1/\alpha}}.$$

Choosing s such that $t/s = \Theta(\log(Bt^{1/\alpha}))$ is an integer, we obtain the bound

$$\Pr[X_1^{(\ell)} \geq t] \leq C_1\frac{(\log(Bt^{1/\alpha}))^{1/\alpha}}{Bt^{1/\alpha}} = C_1\frac{(\log(Bt^\gamma))^\gamma}{Bt^\gamma},$$

where we have put $\gamma = 1/\alpha$ and C_1 is a universal constant. Let $Z = \min_{\ell\in[k]}(X_1^{(\ell)})$. Note that $Z \leq \sum_{j=1}^\infty 1/j^\alpha \leq C_2$, where C_2 is a constant only depending on α. Thus

$$\begin{aligned}\mathbb{E}[Z] &\leq \frac{3}{B^\alpha} + \int_{3/B^\alpha}^{C_2}\Pr[Z \geq t]\,dt \leq \frac{3}{B^\alpha} + \int_{3/B^\alpha}^{C_2}\left(C_1\frac{(\log(Bt^\gamma))^\gamma}{Bt^\gamma}\right)^k dt\\ &\leq \frac{3}{B^\alpha} + \frac{C_3^k\log(B)^{k/\alpha}}{B^k}\int_{3/B^\alpha}^{C_2}\frac{1}{t^{k/\alpha}}\,dt\end{aligned}$$

for some constant C_3 (depending on α). If $k \leq \alpha$, the integral is $O(\log B)$ and this bound suffices. If $k > \alpha$, the integral is $O(B^{k-\alpha})$, which again suffices to give the desired bound. □

Remark A.3. As discussed in Remark 3.3 we only require the hash functions to be $O(\log B)$-independent in the proof of the upper bound of Theorem A.2. In the upper bound of Theorem A.1 we only require the hash functions to be 2-independent.

A.2 Learned Count-Min

We now proceed to analyse the learned Count-Min algorithm which has access to an oracle which, given an item, predicts whether it is among the B heaviest items. The algorithm stores the frequencies of the B heaviest items in B individual buckets, always outputting the exact frequency when queried one of these items. On the remaining items it performs a regular Count-Min sketch with a single hash function hashing to B buckets.

Theorem A.4. *Let $\alpha > 0$ be fixed and $f_i = 1/i^\alpha$ for $i \in [n]$. Let $n, B \in \mathbb{N}$ with $2B \leq n$ and $h : [n] \to [B]$ be a 2-independent hash functions. For $i \in [n]$ define the random variable $\tilde{f}_i = \sum_{j \in [n] \setminus [B]} [h(j) = h(i)] f_j$. Then*

$$\mathbb{E}[|\tilde{f}_i - f_i|] = \begin{cases} \Theta\left(\frac{n^{1-\alpha}}{B}\right), & \alpha < 1 \\ \Theta\left(B^{-\alpha}\right), & \alpha > 1. \end{cases}$$

Proof. Both results follows using linearity of expectation.

$$\mathbb{E}[|\tilde{f}_i - f_i|] = \frac{1}{B} \sum_{j \in [n] \setminus ([B] \cup \{i\})} \frac{1}{j^\alpha} = \begin{cases} \Theta\left(\frac{n^{1-\alpha}}{B}\right), & \alpha < 1, \\ \Theta\left(B^{-\alpha}\right), & \alpha > 1. \end{cases}$$

□

Corollary A.5. *Using the learned Count-Min on input coming from a Zipfian distribution with exponent α, it holds that*

$$\mathbb{E}\left[\sum_{i \in [n]} f_i \cdot |\tilde{f}_i - f_i|\right] = \begin{cases} \Theta\left(\frac{n^{2-2\alpha}}{B}\right), & \alpha < 1, \\ \Theta\left(B^{1-2\alpha}\right), & \alpha > 1. \end{cases}$$

Why are we only analysing learned Count-Min with a single hash function? After all, might it not be conceivable that more hash functions can reduce the expected error? It turns out that if our aim is to minimize the expected error $\text{Err}(\mathcal{F}, \tilde{\mathcal{F}}_\mathcal{A})$ we cannot do better than in Corollary A.5. Indeed, we can employ similar techniques to those used in [HIKV19] to prove the following lower bound extending their result to general exponents $\alpha \neq 1$.

Theorem A.6. *Let $\alpha > 0$ be fixed and $f_i = 1/i^\alpha$ for $i \in [n]$. Let $n, B \in \mathbb{N}$ with $n \geq cB$ for some sufficiently large constant c and let $h : [n] \to [B]$ be **any** function. For $i \in [n]$ define the random variable $\tilde{f}_i = \sum_{j \in [n]} [h(j) = h(i)] f_j$. Then*

$$\sum_{i \in [n]} f_i \cdot |\tilde{f}_i - f_i|] = \begin{cases} \Omega\left(\frac{n^{2-2\alpha}}{B}\right), & \alpha < 1 \\ \Omega\left(B^{1-2\alpha}\right), & \alpha > 1. \end{cases}$$

A simple reduction shows that Count-Min with a total of B buckets and any number of hash functions cannot provide and expected error that is lower than the lower bound in Theorem A.6 (see [HIKV19]).

Proof. We subdivide the exposition into the cases $0 < \alpha < 1$ and $\alpha > 1$.

Case 1: $0 < \alpha < 1$. In this case

$$\sum_{i\in[n]} f_i \cdot |\tilde{f}_i - f_i| \geq \sum_{i\in[n]\setminus[B]} f_i \cdot |\tilde{f}_i - f_i| = \sum_{b\in[B]} \left(\sum_{j\in[n]\setminus[B]:h(j)=b} f_j \right)^2 - \sum_{i\in[n]\setminus[B]} f_i^2$$
$$= \sum_{b\in[B]} S_b^2 - \sum_{i\in[n]\setminus[B]} f_i^2, \qquad (7)$$

where we have put $S_b = \sum_{j\in[n]\setminus[B]:h(j)=b} f_j$, the total weight of items hashing to bucket b. Now by Jensen's inequality

$$\sum_{b\in[B]} S_b^2 \geq \frac{1}{B} \left(\sum_{i\in[n]\setminus[B]} f_i \right)^2$$

Furthermore, we have the estimates

$$\sum_{i\in[n]\setminus[B]} f_i = \sum_{i=B}^{n} \frac{1}{i^\alpha} - \frac{1}{B^\alpha} \geq \int_B^n x^{-\alpha}\, dx - \frac{1}{B^\alpha} = \frac{1}{1-\alpha}(n^{1-\alpha} - B^{1-\alpha}) - \frac{1}{B^\alpha},$$

and

$$\sum_{i\in[n]\setminus[B]} f_i^2 \leq \int_B^n x^{-2\alpha} = \begin{cases} \frac{1}{1-2\alpha}(n^{1-2\alpha} - B^{1-2\alpha}), & \alpha \neq 1/2 \\ \log(n/B), & \alpha = 1/2. \end{cases}$$

Here we have used the standard technique of comparing a sum to an integral. Assuming that $n \geq cB$ for some sufficiently large constant c (depending on α), it follows that $\sum_{b\in[B]} S_b^2 = \Omega\left(\frac{n^{2-2\alpha}}{B}\right)$. It moreover follows (again for n sufficiently large) that,

$$\sum_{i\in[n]\setminus[B]} f_i^2 = \begin{cases} O(\log(n/B)), & \alpha = 1/2, \\ O(n^{1-2\alpha}), & \alpha < 1/2, \\ O(B^{1-2\alpha}), & \alpha > 1/2. \end{cases}$$

Plugging into (7), we see that in each of the three cases $\alpha < 1/2$, $\alpha = 1/2$ and $\alpha > 1/2$ it holds that $\sum_{b\in[B]} S_b^2 - \sum_{i\in[n]} f_i^2 = \Omega\left(\frac{n^{2-2\alpha}}{B}\right)$.

Case 2: $0\alpha > 1$. For this case we simply assume that $n \geq 3B$. Let $I \subseteq [3B] \setminus [B]$ consist of those i satisfying that $h(i) = h(j)$ for some $j \in [3B] \setminus [B]$, $j \neq i$. Then $|I| \geq B$ and if $i \in I$, then $f_i \geq (3B)^{-\alpha}$ and $|\tilde{f}_i - f_i| \geq (3B)^{-\alpha}$. Thus

$$\sum_{i\in[n]} f_i \cdot |\tilde{f}_i - f_i| \geq \sum_{i\in I} f_i \cdot |\tilde{f}_i - f_i| \geq B(3B)^{-2\alpha} = \Omega(B^{1-2\alpha}).$$

$\square$

A.3 Learned Count-Min using a noisy oracle

As we did in the case $\alpha = 1$ (Theorem 5.7), we now present an analogue to Theorem A.4 when the heavy hitter oracle is noisy. Note that the results in Table 3 demonstrates that we obtain no asymptotic improvement using the heavy hitter oracle when $0 < \alpha < 1$ and therefore we only consider the case $\alpha > 1$. We show the following trade-off between the classic and learned case, as the error probability, δ, that the heavy hitter oracle misclassifies an item, varies in $[0, 1]$.

Theorem A.7. *Suppose that the input follows a generalized Zipfian distribution with $n \geq B$ different items and exponent α for some constant $\alpha > 1$. Learned Count-Sketch with a single hash functions, a heavy hitter oracle $\boldsymbol{HH}_\delta$, $B_h = \Theta(B)$ bins allocated to the B_h items classified as heavy and $B - B_h = \Theta(B)$ bins allocated to a Count-Sketch of the remaining items, incurs an expected error of*

$$O\left(\frac{1}{B}\left(\delta + B^{1-\alpha}\right)^2\right)$$

Proof. The proof is very similar to that of Theorem 5.7 Let $h : [n] \to [B - B_h]$ be the hash function used for the Count-Min. In the analysis to follow, it is enough to assume that it is 2-independent. Suppose item i is classified as non-heavy. The expected error incurred by item i is then

$$\frac{1}{B - B_h}\left(\delta \sum_{j \in [B_h] \setminus \{i\}} f_j + \sum_{j \in [n] \setminus ([B_h] \cup \{i\})} f_j\right) = O\left(\frac{\delta + B^{1-\alpha}}{B}\right).$$

Letting $N = \sum_{i \in [n]} f_i = O(1)$, the expected error (as defined in (2)) is at most

$$\frac{1}{N}\left(\delta \sum_{j \in [B_h] \setminus \{i\}} f_j + \sum_{j \in [n] \setminus ([B_h] \cup \{i\})} f_j\right) \cdot O\left(\frac{\delta + B^{1-\alpha}}{B}\right) = O\left(\frac{1}{B}\left(\delta + B^{1-\alpha}\right)^2\right),$$

as desired. □

For $\delta = 1$, we recover the bound for the classic Count-Min. We also see that it suffices that $\delta = O(B^{1-\alpha})$ in order to obtain the same bound as with a perfect heavy hitter oracle.

Appendix E

One-Way Trail Orientations

One-Way Trail Orientations

Anders Aamand[*1], Niklas Hjuler[†1], Jacob Holm[*1], and Eva Rotenberg[‡2]

[1]University of Copenhagen
[2]Technical University of Denmark

Abstract

Given a graph, does there exist an orientation of the edges such that the resulting directed graph is strongly connected? Robbins' theorem [Robbins, Am. Math. Monthly, 1939] asserts that such an orientation exists if and only if the graph is 2-edge connected. A natural extension of this problem is the following: Suppose that the edges of the graph are partitioned into trails. Can the trails be oriented consistently such that the resulting directed graph is strongly connected?

We show that 2-edge connectivity is again a sufficient condition and we provide a linear time algorithm for finding such an orientation.

The generalised Robbins' theorem [Boesch, Am. Math. Monthly, 1980] for mixed multigraphs asserts that the undirected edges of a mixed multigraph can be oriented to make the resulting directed graph strongly connected exactly when the mixed graph is strongly connected and the underlying graph is bridgeless.

We consider the natural extension where the undirected edges of a mixed multigraph are partitioned into trails. It turns out that in this case the condition of the generalised Robbin's Theorem is not sufficient. However, we show that as long as each cut either contains at least 2 undirected edges or directed edges in both directions, there exists an orientation of the trails such that the resulting directed graph is strongly connected. Moreover, if the condition is satisfied, we may start by orienting an arbitrary trail in an arbitrary direction. Using this result one obtains a very simple polynomial time algorithm for finding a strong trail orientation if it exists, both in the undirected and the mixed setting.

[*]This research is supported by Mikkel Thorup's Advanced Grant DFF-0602-02499B from the Danish Council for Independent Research under the Sapere Aude research career programme.

[†]This work is supported by the Innovation Fund Denmark through the DABAI project.

[‡]This research was conducted during the fourth author's time as a PhD student at University of Copenhagen.

1 Introduction and motivation

Suppose that the mayor of a small town decides to make all the streets one-way such that it is possible to get from any place to any other place without violating the orientations of the streets[1]. If all the streets are initially two-way then Robbins' theorem [10] asserts that this can be done exactly when the corresponding graph is 2-edge connected. If, on the other hand some of the streets were already one-way in the beginning then the generalised Robbins' theorem by Boesch [1] states that it can be done exactly when the corresponding "mixed" graph is strongly connected and the underlying graph is bridgeless.

However, the proofs of both of these results assume that every street of the city corresponds to exactly one edge in the graph. This assumption hardly holds in any city in the world and therefore a more natural assumption is that every street corresponds to a trail (informally, a potentially self-crossing path) in the graph and that the edges of each trail must be oriented consistently[2].

In this paper we consider such graphs having their edges partitioned into trails. We prove that the trails can be oriented to make the resulting directed graph strongly connected exactly if the initial graph is 2-edge connected (note that this is precisely the condition of Robbins' theorem).

Not only do we show that the strong trail orientation problem in undirected 2-edge connected graphs always has a solution, we also provide a linear time algorithm for finding such an orientation. In doing so, we use an interesting combination of techniques that allow us to reduce to a graph with a number of 3-edge connected components that is linear in the number of edges. Using that the average size of these components is constant and that we can piece together solutions for the individual components we obtain an efficient algorithm.

Finally, we will consider the generalised Robbins' theorem in this new setting by allowing some edges to be oriented initially and supposing that the remaining edges are partitioned into trails. We will show that if each cut (V_1, V_2) in the graph has either at least 2 undirected edges going between V_1 and V_2 or at least 1 directed edge in each direction then it is possible to orient the trails making the resulting graph strongly connected. In fact, we show that if this condition is satisfied we may start by orienting an *arbitrary* trail in an *arbitrary* direction. Although this condition is not necessary it does give a simple algorithm for finding a trail orientation if it exists. Indeed, initially the graph may contain undirected edges that are *forced* in one direction by some cut. For finding a trail orientation if it exists we can thus orient forced trails in the forced direction. If there are no forced trails we orient any trail arbitrarily.

Note that in the mixed setting the feasibility depends on the trail decomposition which is not the case for the other results. That the condition from the generalised Robbins' theorem is not sufficient can be seen from Figure 1.

Earlier methods Several methods have already been applied for solving orientation problems in graphs where the goal is to make the resulting graph strongly connected.

One approach used by Robbins [10] is to use that a 2-edge connected graph has an *ear-decomposition.* An ear decomposition of a graph is a partition of the set of edges into a cycle C and paths $P_1, \dots, P_t$ such that P_i has its two endpoints but none of its internal vertices on $C \cup \left(\bigcup_{j=1}^{i-1} P_j\right)$. Given the existence of an ear decomposition of a 2-edge connected graph it is easy to prove Robbins'

[1]The motivation for doing so is that the streets of the town are very narrow and thus it is a great hassle when two cars unexpectedly meet.

[2]This version of the problem was given to us through personal communication with Professor Robert E. Tarjan.

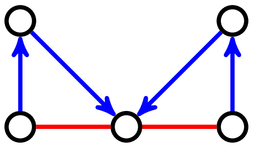

Figure 1: The graph is strongly connected and the underlying graph is 2-edge connected, but irrespective of the choice of orientation of the red trail, the graph will no longer be strongly connected.

theorem. Indeed, any choice of consistent orientations of the paths and the cycle gives a strongly connected graph.

A second approach introduced by Tarjan [4] gives another simple proof of Robbins' theorem. One can create a DFS tree in the graph rooted at a vertex v and orient all edges in the DFS tree away from v. The remaining edges are all back edges (see [4]) and are oriented towards v. It is easily verified that this gives a strong orientation if the graph is 2-edge connected. A similar approach was used by Chung et al. [2] in the context of the generalized Robbins' theorem for mixed multigraphs.

The above methods not only prove Robbins' theorem, they also provide linear time algorithms for finding strong orientations of undirected or mixed multigraphs.

However, none of the above methods have proven fruitful in our case. In case of the ear decomposition we would need one that is somehow compatible with the partitioning into trails, and this seems hard to guarantee. Similar problems appear when trying a DFS-approach. Neither does the proof by Boesch [1] of Robbins' theorem for mixed multigraphs generalise to prove our result. Most importantly, the corresponding theorem would no longer be true for trail orientations as is shown by the example in Figure 1.

Since the classical linear time algorithms rely on ear-decompositions and DFS searches, and since these approaches do not immediately work for trail partitions, our linear time algorithm will be a completely new approach to solving orientation problems.

Structure of the paper The structure of this paper is as follows. In section 3 we prove our generalisation of Robbins' theorem for undirected graphs partitioned into trails. In section 4 we study the case of mixed graphs. Finally in section 5 we provide our linear time algorithm for trail orientation in an undirected graph.

2 Preliminaries

Let us briefly review the concepts from graph theory that we will need.

A graph having some subset of its edges oriented is said to be a *mixed* graph. We will write $\{u, v\}$ for an undirected edge between u and v and (u, v) for an edge directed from u to v.

A *walk* in a graph is an alternating sequence of vertices and edges $v_0, e_1, v_1, e_2, \dots, v_k$, such that for $1 \leq i \leq k$ the edge e_i has v_{i-1} and v_i as its two endpoints. In a directed or mixed graph we further require that either e_i be undirected or directed from v_{i-1} to v_i

A *trail* is a walk without repeated edges. A *path* is a trail without repeated vertices (except possibly $v_0 = v_k$). Finally, a *cycle* is a path for which $v_0 = v_k$.

Next, a mixed multigraph $G = (V, E)$ is called *strongly connected* if for each pair of vertices $u, v \in V$ there exists a walk from u to v. In case the graph is undirected this is equivalent to saying that it consists of exactly one connected component. If $A \subseteq V$ we will say that A is *strongly connected in* G if for each pair of vertices $u, v \in A$ there is a walk in G from u to v.

A *cut* or *edge-cut* (V_1, V_2) in a graph is a partition of its vertices into two non-empty subsets V_1, V_2. We recall the definition of k-edge connectivity. A graph $G = (V, E)$ is said to be k*-edge connected* if and only if $G' = (V, E - X)$ is connected for all $X \subseteq E$ where $|X| < k$. A trivially equivalent condition is that each cut (V_1, V_2) in the graph has at least k edges going between V_1 and V_2.

Finally, if $G = (V, E)$ is a mixed multigraph and $A \subseteq V$ we define G/A to be the graph obtained by contracting A to a single vertex (maintaining duplicate edges and self-loops) and $G[A]$ to be the subgraph of G induced by A. The following simple observation will be used repeatedly in this paper.

Observation 2.1. *If $G = (V, E)$ is k-edge connected and $A \subseteq V$ then G/A is k-edge connected. Also, if G is a strongly connected mixed multigraph then G/A is too.*

3 Robbins Theorem Revisited

We are now ready to state and prove our generalisation of Robbins' theorem.

Theorem 3.1. *Let $G = (V, E)$ be an undirected multigraph with E partitioned into trails. An orientation of each trail such that the resulting directed graph is strongly connected exists if and only if G is 2-edge connected.*

We note that this theorem could also be proven using a general result from Király and Szigeti [6] which relies on theorems by Nash-Willams [9]. However, our proof is significantly simpler (in fact we believe that restating the theorem by Király and Szigeti and explaining the reduction would be more cumbersome) and more suitable for constructing algorithms.

Proof. If G is not 2-edge connected, such an orientation obviously doesn't exist, so we only need to prove the converse. Suppose therefore that G is 2-edge connected.

Our proof is by induction on the number of edges in G. If there are no edges, the graph consists of a single vertex, and the statement is obviously true. Assume now the statement holds for all graphs with strictly fewer edges than G. Pick an arbitrary edge e that sits at the end of its corresponding trail.

If $G - e$ is 2-edge connected, then by the induction hypothesis there is a strong orientation of $G - e$ that respects the trails of G. Such an orientation clearly extends to the required orientation of G.

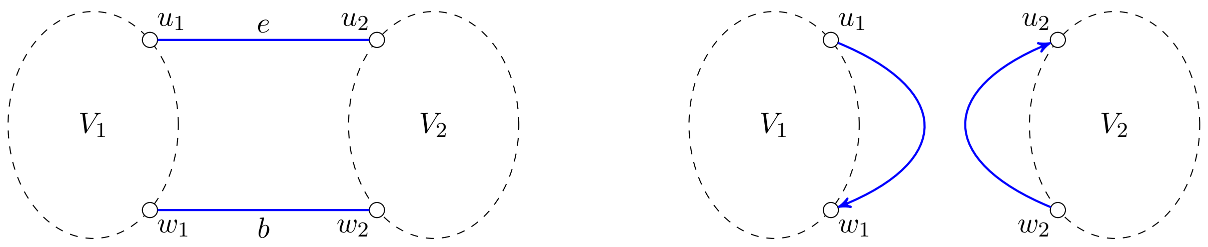

Figure 2: A 2-edge cut and the two graphs $G_1 = G[V_1] \cup \{\{u_1, w_1\}\}$ and $G_2 = G[V_2] \cup \{\{u_2, w_2\}\}$. The orientations of the two new edges are obtained from the strong trail orientations of G_1 and G_2.

If $G - e$ is not 2-edge connected, there exists a bridge b in $G - e$ (see Figure 2). Let V_1, V_2 be the two connected components of $G - \{e, b\}$, and let $e = \{u_1, u_2\}$ and $b = \{w_1, w_2\}$ such that for $i \in \{1, 2\}$, $u_i, w_i \in V_i$ (note that we don't necessarily have that u_i and w_i are distinct for $i \in \{1, 2\}$).

Now for $i \in \{1, 2\}$ construct the graph $G_i = G[V_i] \cup \{\{u_i, w_i\}\}$ (note that $\{u_i, w_i\}$ might be a self-loop but this causes no problems for the argument), and define the trails in G_i to be the trails of G that are completely contained in G_i, together with a single trail combined from the (possibly empty) partial trail of e contained in G_i and ending at u_i, followed by the edge $\{u_i, w_i\}$, followed by the (possibly empty) partial trail of b contained in G_i starting at w_i. Both G_1 and G_2 are 2-edge connected since they can each be obtained as contractions of G with some self-loops deleted. Furthermore, they each have strictly fewer edges than G, so inductively each has a strong orientation that respects the given trails. Further, we can assume that the orientations are such that the new edges are oriented (u_1, w_1) and (w_2, u_2) by flipping the orientation of all edges in either graph if necessary. We claim that this orientation, together with e oriented as (u_1, u_2) and b oriented as (w_2, w_1), is the required orientation of G. To see this first note that (by our choice of flips) this orientation respects the trails. Secondly, suppose $v_1 \in V_1$ and $v_2 \in V_2$ are arbitrary. Since G_1 is strongly connected $G[V_1]$ contains a directed path from v_1 to u_1. Similarly, $G[V_2]$ contains a directed path from u_2 to v_2. Thus G contains a directed path from v_1 to v_2. A similar argument gives a directed path from v_2 to v_1 and since v_1 and v_2 were arbitrary this proves that G is strongly connected and our induction is complete. □

The construction in the proof can be interpreted as a naive algorithm for finding the required orientation when it exists.

Corollary 3.2. *The one-way trail orientation problem on a graph with n vertices and m edges can be solved in $\mathcal{O}(n + m \cdot f(m, n))$ time, where $f(m, n)$ is the time per operation for fully dynamic bridge finding (a.k.a. 2-edge connectivity).*

At the time of this writing [3], this is $\mathcal{O}(n + m(\log n \log\log n)^2)$. In Section 5 we will provide a less naive algorithm which runs in linear time.

4 Extension to Mixed graphs

Now we will extend our result to the case of mixed graphs. We are going to prove the following.

Theorem 4.1. *Let $G = (V, E)$ be a strongly connected mixed multigraph. Then $G - e$ is strongly connected for all undirected $e \in E$ if and only if for any partition $\mathcal{P}$ of the undirected edges of G into trails, and any $T \in \mathcal{P}$, any orientation of T can be extended to a strong trail orientation of $(G, \mathcal{P})$.*

Suppose $G = (V, E)$ is as in the theorem. We will say that $e \in E$ is *forced* if it is undirected and satisfies that $G - e$ is not strongly connected. This terminology is natural as it is equivalent to saying that there exists a cut (V_1, V_2) in G such that e is the only undirected edge in this cut and such that all the directed edges go from V_1 to V_2. If we want an orientation of the trails making the graph strongly connected we are clearly forced to orient e from V_2 to V_1.

Theorem 4.1 is a proper extension of Theorem 3.1 since if G is undirected and 2-edge connected then no $e \in E$ is forced. Furthermore, the theorem suggests a very simple polynomial time algorithm (see Algorithm 1) for finding a strong orientation of the trails if it exists. Indeed, if the mixed graph contains forced edges we direct the corresponding trails in the forced direction. If there are no forced edges then either the graph is no longer strongly connected in which case we know that a strong trail orientation doesn't exist. Otherwise, we may by Theorem 4.1 orient any trail in an arbitrary direction.

Algorithm 1: Algorithm for mixed graphs.

```
Input: A mixed multigraph $G$ and a partition $\mathcal{P}$ of the undirected edges of $G$ into trails.
Output: True if $(G, \mathcal{P})$ has a strong trail orientation, otherwise false. If $G$ has a bridge or is
        not strongly connected, $G$ is left unmodified. Otherwise $G$ is modified, either to
        have such a strong trail orientation, or to a forced graph that is not strongly
        connected.
1  if $G$ has a bridge or is not strongly connected then
2      return false
3  end
4  while $|\mathcal{P}| > 0$ do
5      if for some undirected edge $e$, $G - e$ is not strongly connected then
6          Let $T \in \mathcal{P}$ be the trail containing $e$.
7          if some orientation of $T$ leaves $G$ strongly connected then
8              Apply such an orientation of $T$ to $G$
9          else
10             return false
11         end
12     else
13         Let $T \in \mathcal{P}$ be arbitrary.
14         Update $G$ by orienting $T$ in an arbitrary direction.
15     end
16     Remove $T$ from $\mathcal{P}$.
17 end
18 return true
```

For proving Theorem 4.1 we will need the following lemma.

Lemma 4.2. *Let G be a directed graph, and let (A, B) be a cut with exactly one edge crossing from A to B. Then G is strongly connected if and only if G/A and G/B are.*

Proof. Strong connectivity is preserved by contractions, so if G is strongly connected then G/A and G/B both are. For the other direction, let (a_1, b_1) be the edge going from A to B. As G/A is strongly connected and (a_1, b_1) is the only edge going from A to B we can for any edge (b_2, a_2) going from B to A find a path from b_1 to b_2 that stays in B. Since G/B is strongly connected, it follows that A is strongly connected in G. By a symmetric argument, B is also strongly connected in G and since the cut has edges in both directions (as e.g. G/A is strongly connected), G must be strongly connected. $\square$

Now we provide the proof of Theorem 4.1.

Proof of Theorem 4.1. If there exists an undirected edge e such that $G - e$ is not strongly connected, then the trail T containing e can at most be directed one way since e is forced, so there is an orientation of T that does not extend to a strong trail orientation of $(G, \mathcal{P})$. To prove the converse suppose $G - e$ is strongly connected for all undirected $e \in E$.

The proof is by induction on $|\mathcal{P}|$. If $|\mathcal{P}| = 0$ the result is trivial. So suppose $|\mathcal{P}| \geq 1$ and that the theorem holds for all $(G', \mathcal{P}')$ with $|\mathcal{P}'| < |\mathcal{P}|$.

Consider the chosen trail $T \in \mathcal{P}$. If both orientations of T leave a graph where the condition in the theorem is still satisfied we are home by induction. Otherwise, there must exist a cut (A, B) of the following form: (1) T crosses the cut exactly once, (2) exactly one undirected edge from a different undirected trail $T' \in \mathcal{P}$ crosses the cut and (3) every directed edge crossing the cut goes from A to B.

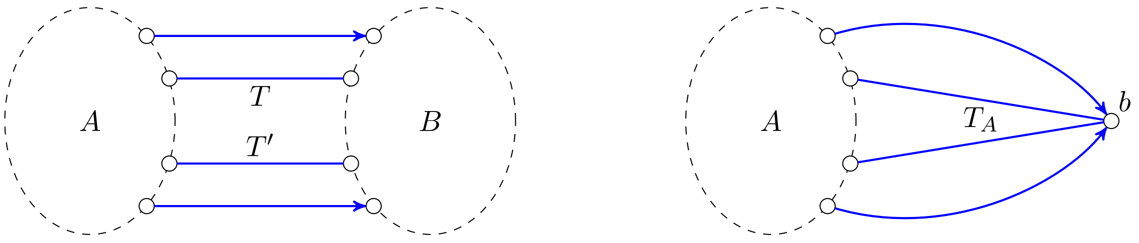

Figure 3: A cut with two undirected edges and all directed edges going from A to B followed by a contraction of B.

Now suppose there is such a cut (A, B) (see Figure 3). Consider the graph G/B and let b be the node corresponding to B in G/B. Let $\mathcal{P}_A$ consist of all trails in $\mathcal{P}$ that are completely contained in A, together with a single trail T_A combined from the (possibly empty) fragments of T and T', joined at b. Since any cut in G/B corresponds to a cut in G, G/B is strongly connected and remains so after deletion of any single undirected edge. By construction $|\mathcal{P}_A| \leq |\mathcal{P}| - 1$, so by induction any orientation of T_A in G/B extends to a strong orientation of $(G/B, \mathcal{P}_A)$. Let G/A, a, $\mathcal{P}_B$ and T_B be defined symmetrically, then by the same argument any orientation of T_B in G/A extends to a strong orientation of $(G/A, \mathcal{P}_B)$. Now for any orientation of T, we can choose orientations of T_A and T_B that are compatible. The result then follows by Lemma 4.2. $\square$

Theorem 4.1 gives a sufficient condition for the existence of a strong orientation and we deal with the other cases by first orienting all forced edges. However, the generalised Robbins' theorem provides a simple equivalent condition, which we lack. Finding such an equivalent condition in our setting is an essential open problem for strong trail orientations. As seen by the example of Figure 1 such a condition will necessarily have to depend on the structure of the trail partition.

5 Linear time algorithm

In this section we provide our linear time algorithm for solving the trail orientation problem in undirected graphs. For this, we make two crucial observations. First, we show that there is an easy linear time reduction from general graphs or multigraphs to cubic multigraphs. Second, we show that in a cubic multigraph with n vertices, we can in linear time find and delete a set of edges that are at the end of their trails, such that the resulting graph has $\Omega(n)$ 3-edge connected components. We further show that we can compute the required orientation recursively from an orientation of each 3-edge connected component together with the cactus graph of 3-edge connected components.

Since the average size of these components is constant, we can compute the orientations of most of them in constant time individually and thus in linear time taken together. The rest contains at most a constant fraction of the vertices, and so a simple geometric sum argument tells us that the total time is also linear.

We start out by making the following reduction.

Lemma 5.1. *The one-way trail problem on a* 2*-edge connected graph or multigraph with* n *vertices and* m *edges, reduces in* $\mathcal{O}(m+n)$ *time to the same problem on a* 2*-edge connected cubic multigraph with* $2m$ *vertices and* $3m$ *edges.*

Proof. Cyclically order the edges adjacent to each vertex such that two edges that are adjacent on the same trail are consecutive in the order. Replace each single vertex v with a cycle of length $\deg(v)$, with each vertex of the new cycle inheriting a corresponding neighbour of v such that the order of the vertices on the cycle corresponds to the cyclic ordering (see Figure 4). Note that for a vertex of degree 2, this creates a pair of parallel edges, so the result may be a multigraph. By the choice of cyclic ordering, we can make the cycle-edge between the two vertices on the same trail belong to that trail. The rest of the cycle edges form new length 1 trails. Clearly the new graph is also 2-edge connected so by Theorem 3.1 it has a strong trail orientation, and any strong trail orientation on this graph translates to a strong trail orientation of the original graph. The new graph has exactly $2m$ vertices and $3m$ edges, and is constructed in $\mathcal{O}(m+n)$ time.

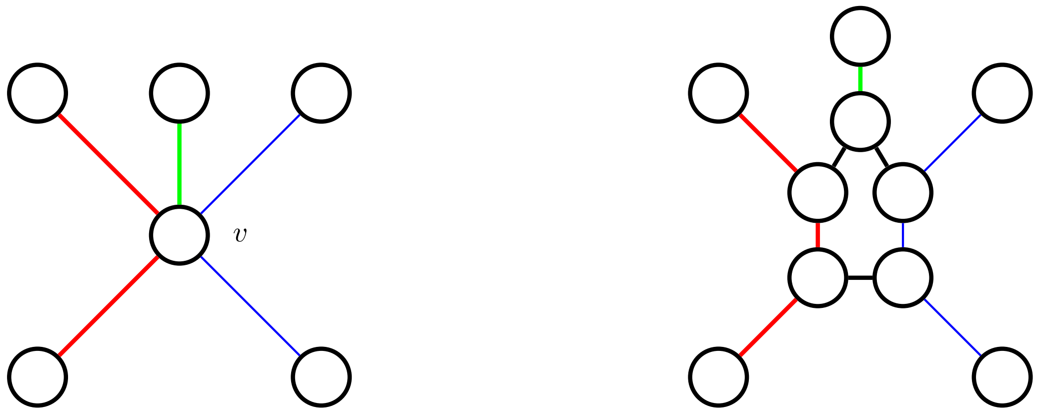

Figure 4: A node of degree 5 turns into a cycle of length 5.

$\square$

Recall now that a multigraph C is called a *cactus* if it is connected and each edge is contained in at most one cycle. If G is any connected graph we let $C_1, \ldots, C_k$ be its 3-edge connected components. It is well known that if we contract each of these we obtain a cactus graph. For a proof of this result see section 2.3.5 of [8]. As the cuts in a contracted graph are also cuts in the original graph we have that if G is 2-edge connected then the cactus graph is 2-edge connected. The edges of the cactus are exactly the edges of G which are part of a 2-edge cut. We will call these edges *2-edge critical.*

It is easy to check that if a cactus has m edges and n vertices then $m \leq 2(n-1)$. We will be using this result in the proof of the following lemma.

Lemma 5.2. *Let $G = (V, E)$ be a cubic 2-edge connected multigraph, let $X \subseteq E$, and let $F \subseteq E$ with $F \supseteq E \setminus X$ minimal (w.r.t inclusion) such that $H = (V, F)$ is 2-edge connected. Then H has at least $\frac{2}{5}|X|$ distinct 3-edge connected components.*

Proof. Let $X_{\text{del}} = X \setminus F$ be the set of edges deleted from G to obtain H, and let $X_{\text{keep}} = X \setminus X_{\text{del}}$ be the remaining edges in X.

By minimality of H there are at least $|X_{\text{keep}}|$ 2-edge-critical edges in H i.e. edges of the corresponding cactus, and thus, if $|X_{\text{keep}}| \geq \frac{4}{5}|X|$, there are at least $\frac{1}{2}|X_{\text{keep}}| + 1 \geq \frac{2}{5}|X| + 1$ distinct 3-edge connected components.

If $|X_{\text{keep}}| \leq \frac{4}{5}|X|$ then $|X_{\text{del}}| \geq \frac{1}{5}|X|$, and since G is cubic and the removal of each edge creates two vertices of degree 2 we must have that H has at least $2|X_{\text{del}}| \geq \frac{2}{5}|X|$ distinct 3-edge connected components. ☐

Lemma 5.3. *Let $G = (V, E)$ be a connected cubic multigraph with E partitioned into trails. Then G has a spanning tree that contains all edges that are not at the end of their trail.*

Proof. Let F be the set of edges that are not at the end of their trail. Since G is cubic, the graph (V, F) is a collection of vertex-disjoint paths, and in particular it is acyclic. Since G is connected, F can be extended to a spanning tree. ☐

Note that we can find this spanning tree in linear time e.g. by contracting all edges internal to a trail, finding a spanning tree of the resulting graph, and adding the internal trail edges to the edges of this spanning tree.

Lemma 5.4. *Let $G = (V, E)$ be a cubic 2-edge connected multigraph with E partitioned into trails. Let T be a spanning tree of G containing all edges that are not at the end of their trail. Let H be a minimal subgraph of G (w.r.t inclusion) that contains T and is 2-edge connected. Then for any $k \geq 5$, less than $\frac{4}{5}\frac{k}{k-1}|V|$ of the vertices in H are in a 3-edge connected component with at least k vertices.*

Proof. Let X be the set of edges that are not in T. Since G is cubic, $|X| = \frac{1}{2}|V| + 1$. By Lemma 5.2 H has at least $\frac{2}{5}|X| > \frac{1}{5}|V|$ 3-edge connected components. Each such component contains at least one vertex, so the total number of vertices in components of size at least k is less than $\frac{k}{k-1}\left(|V| - \frac{1}{5}|V|\right) = \frac{4}{5}\frac{k}{k-1}|V|$. ☐

Definition 5.5. Let C be a 3-edge connected component of some 2-edge connected graph H, whose edges are partitioned into trails. Define $\Gamma_H(C)$ to be the 3-edge connected graph obtained from C by inserting a new edge $\{e_1, f_1\}$ for each min-cut $\{e, f\}$ where $e = \{e_1, e_2\}$ and $f = \{f_1, f_2\}$ and $e_1, f_1 \in C$. Define the corresponding partition of the edges of $\Gamma_H(C)$ into trails by taking every trail that is completely contained in C, together with new trails combined from the fragments of the trails that were broken by the min-cuts together with the new edges that replaced them. See Figure 5.

At this point the idea of the algorithm can be explained. We remove as many of the edges that sit at the end of their trails as possible, while maintaining that the graph is 2-edge connected. Lemma 5.4 guarantees that we obtain a graph H with $\Omega(|V|)$ many 3-edge connected components of size $O(1)$. We solve the problem for each $\Gamma_H(C)$ for every 3-edge connected component. Finally, we combine the solutions for the different components like in the proof of Theorem 3.1.

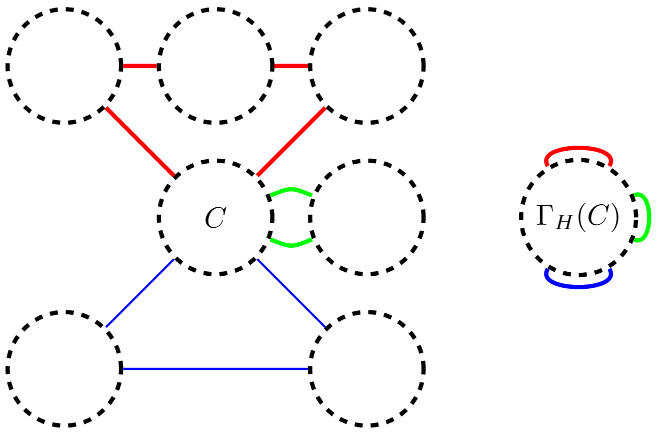

Figure 5: The 3-edge connected components of a 2-edge connected graph. Notice that every edge leaving a 3-edge connected component C becomes part of a cycle if all 3-edge components are contracted. The right hand side shows $\Gamma_H(C)$ where C is the component in the middle.

Theorem 5.6. *The one-way trail orientation problem can be solved in $\mathcal{O}(m+n)$ time on any 2-edge connected undirected graph or multigraph with n vertices and m edges.*

Proof. By Lemma 5.1, we can assume the graph is cubic. For the algorithm we will use two subroutines. First of all, when we have found a minimum spanning tree T containing the edges that are not on the end of their trail we can use the algorithm of Kelsen et al. [5] to, in linear time, find a minimal (w.r.t. inclusion) subgraph H of G that contains T and is 2-edge connected. Secondly, we will use the algorithm by Mehlhorn et al. [7] to, in linear time, build the cactus graph of 3-edge connected components. The algorithm runs as follows:

1. Construct a spanning tree T of G that contains all edges that are not at the end of their trail.
2. Construct a minimal subgraph H of G that contains T and is 2-edge connected[3].
3. Find the cactus of 3-edge connected components of[4] H.
4. For each 3-edge connected component C_i, construct $\Gamma_H(C_i)$.
5. Recursively compute an orientation for each[5] $\Gamma_H(C_i)$.
6. Combine the orientations from each component to a strong trail orientation of H. A such is also a strong trail orientation of G.

First we will show correctness and then we will determine the running time.

Recall that we can flip the orientation in each $\Gamma_H(C_i)$ and still obtain a strongly connected graph respecting the trails in $\Gamma_H(C_i)$. The way we construct the orientation of the edges of G is by flipping the orientation of each $\Gamma_H(C_i)$ in such a way that each cycle in the cactus graph becomes a

[3]See Kelsen [5].
[4]See Mehlhorn [7].
[5]Note that $\Gamma_H(C_i)$ is cubic unless it consists of exactly one node. In this case however we don't need to do anything.

directed cycle[6]. This can be done exactly because no edge of the cactus is contained in two cycles. By construction this orientation respects the trails so we need to argue that it gives a strongly connected graph.

For showing that the resulting graph is strongly connected, consider the graph in which every 3-edge connected component is contracted to a single point. This is exactly the cactus of 3-edge connected component of G which is strongly connected as the cycles of the cactus graph have become directed cycles. Now assume inductively that we have uncontracted some of the 3-edge connected components obtaining a graph G_1 which is strongly connected. We then uncontract another component C (see Figure 6) and obtain a new graph G_2 which we will show is strongly connected. If $u, v \in C$, then since $\Gamma_H(C)$ is strongly connected there is a path from u to v in $\Gamma_H(C)$. If this path only contains edges which are edges of C it will also exist in G_2. If the path uses one of the added (now oriented) edges (e_1, f_1), it is because there are edges (e_1, e_2) and (f_2, f_1) forming a cut and thus being part of a cycle in the cactus. In this case we use edge (e_1, e_2) to leave component C and then go from e_2 back to component C which is possible since G_1 was strongly connected. When we get back to the component C we must arrive at f_1 since otherwise there would be two cycles in the cactus containing the edge (e_1, e_2). Hence we succeeded in disposing of the edge (e_1, f_1) with a directed path in G_2. This argument can be used for any of the edges of $\Gamma_H(C)$ that are not in C and thus C is strongly connected in G_2. Since G_1 was strongly connected this suffices to show that G_2 is strongly connected. By induction this implies that after uncontracting all components the resulting graph is strongly connected.

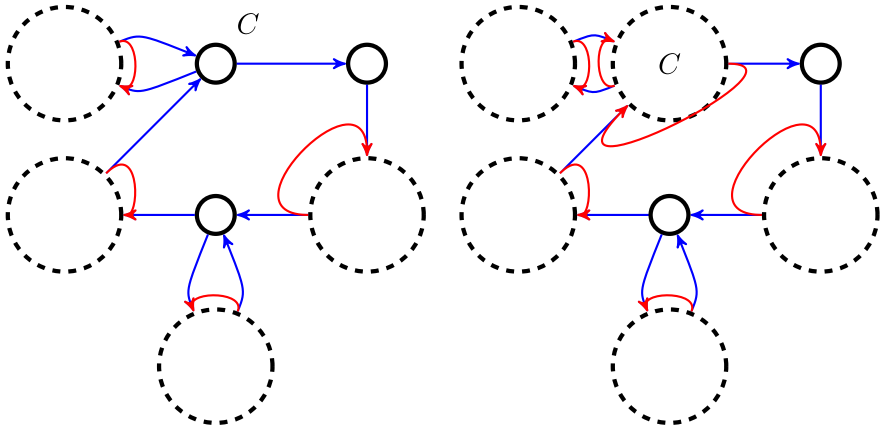

Figure 6: Before and after uncontracting component C. The blue edges are the edges of the cactus i.e. the 2-edge critical edges of H. The red edges are the ones obtained from the 2-edge cuts of H as described in the construction of the $\Gamma_H(C_i)$.

Now for the running time. By Lemma 5.4 each level of recursion reduces the number of vertices in "large" components by a constant fraction, for instance for $k = 10$ we reduce the number of vertices in components of size at least 10 by a factor of $\frac{8}{9}$. Let $f(n)$ be the worst case running time with n nodes for a cubic graph, and pick c large enough such that cn is larger than the time it takes to go through steps 1-4 and 6 as well as computing the orientations in the "small" components. This includes the linear time needed to construct the new set of trails (in 4), and the linear time to

[6]In practice this is done by making a DFS (or any other search tree one likes) of the cactus and repeatedly orienting each component in a way consistent with the previous ones.

reassemble the directed trails (in 6). Let $a_1, \dots, a_k$ be the number of vertices in the "large" 3-edge connected components. Then $\sum_i a_i \leq \frac{8n}{9}$ and

$$f(n) \leq cn + \sum_i f(a_i).$$

Inductively, we may assume that $f(a_i) \leq 9cn$ and thus obtain

$$f(n) \leq cn + \sum_i f(a_i) \leq cn + \sum_i 9ca_i = cn + 8cn = 9cn$$

proving that $f(n) \leq 9cn$ for all n. □

Algorithm 2: Linear time algorithm for cubic graphs.

```
Input: A 2-edge connected undirected cubic multigraph $G$ and a partition $\mathcal{P}$ of the edges of
       $G$ into trails.
Output: $G$ is modified to a strong trail orientation of $(G, \mathcal{P})$.
1  Construct a spanning tree $T$ of $G$ that contains all edges that are not at the end of their trail.
2  Construct a minimal subgraph $H$ of $G$ that contains $T$ and is 2-edge connected.
3  Find the cactus $C$ of 3-edge connected components of $H$.
4  for each 3-edge connected component $C_i$ in $C$ in DFS preorder do
5      Construct $G_i = \Gamma_H(C_i)$.
6      Recursively compute an orientation for $G_i$.
7      if the orientation of $G_i$ is not compatible with its DFS parent then
8          Flip orientation of $G_i$
9      end
10 end
11 for each edge $e$ deleted from $G$ to create $H$ do
12     if no edge on the trail of $e$ has been oriented yet then
13         Pick an arbitrary orientation for $e$.
14     else
15         Set the orientation of $e$ to follow the trail.
16     end
17 end
```

6 Open problems

We here mention two problems concerning trail orientations which remain open.

First of all, our linear time algorithm for finding trail orientations only works for undirected graphs and it doesn't seem to generalise to the trail orientation problem for mixed graphs. It would be interesting to know whether there also exists a linear time algorithm working for mixed graphs. If so it would complete the picture of how fast an algorithm we can obtain for any variant of the trail orientation problem.

Secondly, our sufficient condition for when it is possible to solve the trail orientation problem for mixed multigraphs is clearly not necessary. It would be interesting to know whether there is a simple necessary and sufficient condition like there is in the undirected case. Since in the mixed case the answer to the problem actually depends on the given trail decomposition and not just on the structure of the mixed graph it is harder to provide such a condition. One can however give the following condition. It is possible to orient the trails making the resulting graph strongly connected if and only if when we repeatedly direct the forced trails end up with a graph satisfying our condition in Theorem 4.1. This condition is not simple and is not easy to check directly. Is there a more natural condition?

References

[1] Frank Boesch and Ralph Tindell. Robbins's theorem for mixed multigraphs. *The American Mathematical Monthly*, 87(9):716–719, 1980.

[2] Fan R. K. Chung, Michael R. Garey, and Robert E. Tarjan. Strongly connected orientations of mixed multigraphs. *Networks*, 15(4):477–484, 1985.

[3] Jacob Holm, Eva Rotenberg, and Mikkel Thorup. Dynamic bridge-finding in Õ(log2 n) amortized time. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, pages 35–52, 2018.

[4] John Hopcroft and Robert Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, June 1973.

[5] Pierre Kelsen and Vijaya Ramachandran. On finding minimal 2-connected subgraphs. In *Proceedings of the Second Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 28-30 January 1991, San Francisco, California.*, pages 178–187, 1991.

[6] Zoltán Király and Zoltán Szigeti. Simultaneous well-balanced orientations of graphs. *J. Comb. Theory, Ser. B*, 96(5):684–692, 2006.

[7] Kurt Mehlhorn, Adrian Neumann, and Jens M. Schmidt. Certifying 3-edge-connectivity. *Algorithmica*, 77(2):309–335, 2017.

[8] Hiroshi Nagamochi and Toshihide Ibaraki. *Algorithmic Aspects of Graph Connectivity.* Cambridge University Press, New York, NY, USA, 1 edition, 2008.

[9] J. A. Nash-Willams. On orientations, connectivity and odd-vertex-pairings in finite graphs. *Canad. J. Math.*, 12(5):555–567, 1960.

[10] H. E. Robbins. A theorem on graphs, with an application to a problem of traffic control. *The American Mathematical Monthly*, 46(5):281–283, 1939.