UNIVERSITY OF COPENHAGEN
FACULTY OF SCIENCE

PhD Thesis

Realistic Hashing, Online Sorting, and Constrained Correlation Clustering

Jonas Klausen

This PhD project has been supervised by Mikkel Thorup, Ioana Bercea, and Jacob Holm

Submitted March 31, 2025

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

Abstract

This thesis summarizes the work I've done during my PhD program at BARC, covering new results in Realistic Hashing, Online Sorting, and Constrained Correlation Clustering

Realistic Hashing revolves around the study of realistically implementable families of functions which carry strong mathematical guarantees. Here we build upon the work of Dahlgaard, Knudsen, Rotenberg, and Thorup [DKRT15], introducing a new tabulation-based hash function, *Tornado Tabulation*, giving both new and stronger guarantees than Mixed Tabulation.

Online Sorting is a new problem introduced by Aamand, Abrahamsen, Beretta, and Kleist [AABK23]. Real values arrive one by one, and we must place them into an (approximately) sorted list without moving previous elements. We strengthen their lower bound to also cover randomized algorithms in the oblivious setting, and show (positive and negative) results for a range of new variations of the problem.

Finally, Constrained Correlation Clustering, introduced by van Zuylen, Hegde, Jain, and Williamson [vZHJW07], is a generalization of the well-known Correlation Clustering problem where certain preferences are promoted to hard constraints, thus restricting the set of valid solutions. We present a new algorithm that is significantly faster than the one from [vZHJW07], at the cost of providing a slightly worse (but still constant) approximation factor.

Resumé

Denne afhandling opsummerer arbejdet jeg har udført som del af min PhD-uddannelse ved BARC og inderholder resultater inden for realistiske hashfunktioner, Online Sorting og Constrained Correlation Clustering.

Studiet af realistiske hashfunktioner søger at studere praktisk implementerbare familier af funktioner som besidder stærke matematiske garantier. Vi bygger oven på Dahlgaard, Knudsen, Rotenberg og Thorup's arbejde [DKRT15] og introducerer en ny tabulation-baseret hashfunktion, *Tornado Tabulation*, som både besidder nye og stærkere egenskaber end Mixed Tabulation.

Online Sorting er et nyt problem introduceret af Aamand, Abrahamsen, Beretta og Kleist [AABK23]. Reelle tal ankommer ét af gangen, og skal placeres i (omtrentligt) sorteret rækkefølge i en liste uden at flytte på tidligere indsatte elementer. Vi forbedrer deres nedre grænse ved at udvide den til også at dække randomiserede algoritmer og viser (positive såvel som negative) resultater i en række nye variationer af dette problem.

Constrained Correlation Clustering, introduceret af van Zuylen, Hegde, Jain og Williamson [vZHJW07] er en generalisering af det velkendte Correlation Clustering problem hvor visse præferencer gøres til hårde krav som en løsning *må* opfylde. Vi præsenterer en ny algoritme der er væsentligt hurtigere end den fra [vZHJW07], til gengæld garanterer vores algoritme en svagere (men stadigvæk konstant) approksimationsfaktor.

Acknowledgements

Thank you to everyone who helped make this thesis a reality. It goes without saying that the papers attached in the appendix wouldn't have come into being without my co-authors, but they only make up the tip of the iceberg when it comes to the people who have helped me through my years as a PhD student.

First off, thank you to Mikkel, my advisor. You believed in me, sometimes more than I did myself. We've had more fun times than I can count; working, traveling, and partying together. It's been a pleasure working with you – rarely easy, but always fun. Thank you for bringing me aboard, I love being a part of BARC.

To my co-advisors, Ioana and Jacob, for laughs, drinks, and discussions through the years. An extra round of thanks to Ioana for always being supportive, helping me navigate and survive the chaos of BARC, Mikkel, and life in general. Thank you for telling me when to relax, but also when to get to work. We've been on some great adventures together, and I'm very happy to call you my friend.

To my family, for supporting me all the way, and for always being interested in my work – even if I never quite managed to explain what it is that I do. To the friends I made at DIKU; especially Asbjørn, Gabriel, and Rasmus, the best (and most ambitious) study group one could ask for. I fully believe that our endless discussions (of the curriculum, at times, but mostly of what lies beyond) played a key part in preparing me for a life of research.

Thanks to awesome friends at BARC: Viktor, for many great discussions and good times, both in and out of the office. I think back on our course on probability theory with fondness, where, although equally unqualified, we managed to keep each other afloat. What a ride! Evangelos, you made me feel welcome before I even considered applying for the PhD. Keep it real! And Joel, for always being good company, even if we are terrible at getting together. It saddens me that none of you are in Copenhagen as I'm writing this, but such is the life of short-term employment.

Thanks to Alessandro and Luciano, for hosting me at Tor Vergata, and for an awesome time in both Egham, London, and Rome.

Thank you Karol and Evangelos (again) for telling me to come visit MPI, for showing me around, and for convincing me to apply for a job. It couldn't have worked out better, and I'm super excited about joining you in Saarbrücken after summer.

Thank you Anne, for being amazing. For always listening and giving advice, no matter what trouble I've found myself in, and for teaching me a thing or two about friendships. Whether across the hallway or across the world, you've always been there for me.

Thank you Helen, for picking me up, calming me down, and for getting excited with me. You taught me that kindness sometimes takes the form of a firm "No", which I sorely needed to learn.

And, finally, thank you Monika. You may be the most recent addition to this list, but you have turned my life upside down and I don't want it back the way it was. Having you by my side makes me remember that there's more to life than doing research, and each new day with you is more exciting than the one that came before it. This thesis signals the end of a chapter in my life, and I'm beyond thrilled at the thought of the future chapters we will be writing together.

Contents

Chapter 1

Introduction

This thesis deals with three separate topics in algorithms research, on the basis of four papers written during my PhD project at BARC:

1. *Locally Uniform Hashing*, presented at FOCS '23,
   with Ioana Bercea, Lorenzo Beretta, Jakob Houen, and Mikkel Thorup.

   Available at https://doi.org/10.1109/FOCS57990.2023.00089 [BBK+23].

2. *Hashing for Sampling-Based Estimation*, in submission,
   with Anders Aamand, Ioana Bercea, Jakob Houen, and Mikkel Thorup.

   Available at https://doi.org/10.48550/arXiv.2411.19394 [ABH+24].

3. *Online Sorting and Online TSP*, presented at ESA '24,
   with Mikkel Abrahamsen, Ioana Bercea, Lorenzo Beretta, and László Kozma.

   Available at https://doi.org/10.4230/LIPIcs.ESA.2024.5 [ABB+24].

4. *A Faster Algorithm for Constrained Correlation Clustering*, presented at STACS '25,
   with Nick Fischer, Evangelos Kipouridis, and Mikkel Thorup.

   Available at https://doi.org/10.4230/LIPIcs.STACS.2025.32 [FKKT25].

In accordance with the rules of the PhD program at the Faculty of Science at the University of Copenhagen, this thesis is split into two parts: First, chapters 2 to 4 introduce the four papers above, giving an overview of their contributions and techniques. Second, the papers themselves are attached in their full versions as appendices A to D.

The two papers on hashing make up the majority of the thesis and represent the main topic of my PhD project. The remaining two papers are self-contained with no relation to the others, although a common thread across them all is the topic of randomization.

In *Locally Uniform Hashing* we tackle the fundamental question of constructing practical hash functions with provable mathematical guarantees, in this case that of *Local Uniformity*. Local Uniformity is an extension of Uniform Hashing, which is an alternative to k-independent hash functions. Rather than guarantee that each set of k keys will be hashed independently, uniform hashing gives independence for significantly larger sets. In exchange for this strong property, certain error events are introduced – these events are highly unlikely, but no guarantees are made when they do occur. Local Uniformity further certifies that keys assigned hash values in the same interval $[a, b]$ are hashed independently. That is, even when conditioning on keys having hash values close to each other, we still find that their hash values are independently distributed.

Building upon the work of Dahlgaard, Knudsen, Rotenberg, and Thorup [DKRT15], we introduce Tornado Tabulation Hashing, show that it is locally uniform and detail how this property can be used for implementing Linear Probing. When implemented this way, Linear Probing yields performance guarantees matching those obtained with fully random hashing (up to lower order terms).

In the second paper, we further show that Tornado Tabulation exhibits Chernoff-style concentration bounds and derive tail inequalities with explicit probability bounds. For any concrete set of parameters we can thus compute bounds on the probability that the number of elements hashing to a specific value (or set of values) deviates significantly from its expectation. This makes Tornado Tabulation suitable for the computation of sampling-based estimators. Here, statistics on a large data stream are estimated by computing statistics on a sampled subset of the stream. The quality of the final estimate crucially depends on the concentration given by the applied hash function. With better concentration we thus need fewer independent trials to reach any fixed quality guarantee.

The work on hashing is discussed in chapter 2, following this introduction.

Online Sorting is an online problem recently introduced by Aamand, Abrahamsen, Beretta, and Kleist [AABK23] where n reals arrive one-by-one and must immediately and irrevocably be placed into an array T of length n. The goal is to minimize the total sum of neighboring differences, $\sum_i |T[i] - T[i+1]|$. Offline, this objective is easily minimized by placing elements in sorted order, but in an online setting it was shown by [AABK23] that no deterministic algorithm could achieve competitive ratio $o(\sqrt{n})$. We extended this lower bound to cover randomized algorithms, and made contributions to a number of variations on this problem. Notably, we saw that significantly better bounds can be given (with high probability) when the input stream consists of independently and uniformly drawn values in the interval $[0, 1]$. This work in presented in chapter 3.

Finally, in chapter 4, we discuss *Correlation Clustering*, a clustering problem on graphs. Each edge of the graph describes that its endpoints prefer to be clustered together, while the absence of an edge describes how a pair of elements prefers to be in different clusters. The objective is to partition the vertices into clusters violating as few preferences as possible. We consider a slight variation, Constrained Correlation Clustering, where some preferences are instead hard constraints that must be obeyed. That is, we wish to minimize the number of violated preferences, but don't allow any of the hard constraints to be broken. The only known approximation to Constrained Correlation Clustering, given by van Zuylen and Williamson [vZW09], is based on solving large linear programs, naturally leading to a high running time. Specifically, theirs is a 3-approximation running in time $\mathcal{O}(n^{3\omega})$, where $\omega \geq 2$ is the matrix multiplication constant. In our work, we present an algorithm running in $\tilde{\mathcal{O}}(n^3)$ time, giving a 16-approximation. The new algorithm is thus significantly faster at the cost of a worse, yet still constant, approximation factor.

Chapter 2

Realistic Hashing

In this chapter I discuss the main topic of my PhD project, the study of realistic hash functions. Specifically, the analysis of Tornado Tabulation Hashing, which we introduced in [BBK+23] (appendix A), a new family of functions based on Zobrist's Tabulation Hashing [Zob70].

In the first section below I give a high-level motivation for this direction of study as well as an overview of prior approaches to realistic hashing. In section 2.2 I define our new family of functions, together with a general introduction to tabulation-based hashing and central related concepts. Section 2.3 presents the results and (some) techniques of [BBK+23] (appendix A). In it I introduce and motivate our extension to local uniformity, and sketch how we show uniformity for Tornado Tabulation. Section 2.4 likewise covers [ABH+24] (appendix B) where we show concentration bounds for Tornado Tabulation.

Finally, section 2.5 contains a discussion on some loose ends left by the two papers, presenting possible next steps in developing the theory around Tornado Tabulation.

2.1 Why Realistic Hashing, and How?

When analyzing randomized data structures, one commonly assumes access to fully random (or *uniform*) hashing. That is, access to a random function f such that the hash value $f(x)$ is distributed uniformly and independently of all other hash values. This function possesses powerful qualities that make it easier to analyze complicated processes and data structures, but with it comes an important drawback: its space usage. Storing a fully random hash function requires storing the hash values of all keys in the universe, making the implementation of f intractable for all but trivial problem instances.

Results and analyses using this assumption thus carry limited practical value. Developers wishing to implement systems based on cutting-edge research can't implement the fully random function and must instead replace it with some other other function of their choice – preferably on that *looks* random – losing all mathematical guarantees promised by the research paper.

When studying realistic hashing, the emphasis is instead placed on devising solutions that rely on hash functions which can feasibly be implemented, even if this comes at a cost in terms of slightly worse running time or otherwise weaker guarantees as compared to what is otherwise considered to be the state of the art.

Besides keeping this goal in mind when designing new data structures, a second approach to producing practically relevant research is to revisit published results that base their analyses on fully random hashing. By carefully inspecting the ways they rely on their supplied hash functions, it may be possible to prove some of the stated guarantees while making weaker assumptions on the

hash function applied. The hope is that the fully random function can be swapped for a realistic alternative while keeping the guarantees afforded by the original analysis – approximately, if not exactly.

2.1.1 k-independence

The most well-known families of implementable hash functions with such formal guarantees are captured by the hierarchy of k-independent hash functions introduced by [WC81].

Definition 2.1. A hash function $h\colon \mathcal{U} \to \mathcal{R}$ is *k-independent* if, for all $\{x_1, \ldots, x_k\} \subseteq \mathcal{U}$ and all $y_1, \ldots, y_k \in \mathcal{R}$,

$$\Pr\left[\bigwedge_{i=1}^{k} h(x_i) = y_i\right] = \left(\frac{1}{|\mathcal{R}|}\right)^k.$$

Intuitively, the behavior of a k-independent function is indistinguishable from that of fully random hashing whenever the event under inspection is defined on no more than k keys at a time. A k-independent function can be implemented as a polynomial of degree $k-1$ with random coefficients, and can thus be stored in $\mathcal{O}(k)$ words of space and evaluated with $k-1$ multiplications.

Naturally, a larger parameter k gives a function with stronger properties. An example of this is seen in the work of [PPR09, PT10] studying Linear Probing, a simple way of implementing hash tables. In [PPR09] it is shown that the update time of Linear Probing implemented with 5-independent hashing is only a constant factor away from that achieved when "implemented" with fully random hashing, as studied by [Knu63]. 4-independence, on the other hand, is insufficient [PT10], leading to significantly worse performance when exposed to certain structured update sequences.

But to get stronger properties, like Chernoff-style concentration bounds, a high degree of independence may be required. To obtain concentration bounds giving error probability p, independence of order $\log(1/p)$ is required in the analysis of [SSS95]. At this point the running time of the hash function becomes an issue for high-throughput systems.

Although an important tool for making research relevant for implementation, k-independence (as obtained through polynomials) is not a silver bullet allowing us to implement everything without significant sacrifices. Instead we look to functions with a slightly different property.

2.1.2 Uniform Hashing

Rather than requiring the hash function to behave independently on *all* sets of a given size, we can inspect the behavior of the function on specific sets of keys.

Definition 2.2. A hash function $h\colon \mathcal{U} \to \mathcal{R}$ is *uniform* on a set of keys $\{x_1, \ldots, x_\ell\} \subseteq \mathcal{U}$ if the ordered set of hash values $(h(x_i))_{i=1}^{\ell}$ is uniformly distributed over $\mathcal{R}^\ell$.

By this definition, the k-independent functions are exactly those that are uniform on all sets of size (up to) k, while the fully random function is uniform on the full universe $\mathcal{U}$.

Work by Pagh and Pagh [PP08] as well as Dietzfelbinger and Woelfel [DW03] constructed hash functions that are uniform on larger key sets *whp*. That is, for any *fixed* set of keys X (up to some size), there is an associated good event $\mathcal{E}$ such that, when conditioning on $\mathcal{E}$, h is uniform on X. Event $\mathcal{E}$ happens with high probability, but in the rare case where it fails no guarantees are made on the behavior of the hash function.

If we could let X be the full set of keys presented to our data structure, all claims made under the assumption of fully random hashing would immediately apply when implemented with

these hash functions instead, only with an added error probability of $\Pr[\neg\mathcal{E}]$. This works great for smaller problem instances, but these families naturally become more expensive when they need to support larger sets.

For data structures like the Invertible Bloom Filter [EG11], this is less of a problem. An IBF can be exposed to an unbounded number of insertions, with the promise that the vast majority of elements will end up being removed again (for more on IBFs see section 2.5) The structure of the IBF makes it *deletion-independent*, in the sense that the state of the data structure is independent of prior insertion/removal pairs, and future performance is thus only dependent on the (few) keys left in the structure. Letting X be this set of leftover keys brings the requirements on our hash function down to a more manageable level.

2.1.3 Locally Uniform Hashing

The main issue with the procedure outlined in the preceding section is that we include far too many keys in X, making the hash functions expensive. The work of [DKRT15] introduced a layer of indirection, allowing for the set X to be chosen in a more intelligent manner. In the same way that the state of an IBF only relies on a subset of the keys passed through it, other data structures exhibit properties that we can exploit in order to define a narrower set of "important" keys.

Two such examples, as discussed by [DKRT15], are MinHash (by [Bro97]) and HyperLogLog (by [FEFGM07]) – both approaches to estimating the number of distinct elements in data streams. Although the two methods differ in their way of computing estimates, they share a common trait: They only save the *smallest* hash value(s) encountered while processing the stream. Thus the estimates given are independent of the vast majority of hash values, and if we could guarantee uniformity on the few keys that end up deciding the estimate, the scheme would performing as if implemented with fully random hashing. One way of "selecting" this set of important keys, would be to let X be all keys with hash values below a set threshold – a threshold set high enough that, with high probability, it will capture the necessary key(s), yet low enough that not too many keys will be selecting.

Obviously, the tools presented thus far doesn't allow us to make claims about a set defined in this manner. Picking keys based on their hash values reveals information about the random function, and thus invalidates a crucial assumption made in the previous publications: that X is defined independently of the hash function h.

Yet this is exactly what the work of [DKRT15] allows for, with certain technical restrictions. Specifically, they show that their function is uniform (whp) on the set X defined as the preimage $h^{-1}([a,b)) = \{x \in S \colon h(x) \in [a,b)\}$, given that $[a,b)$ forms a dyadic interval. That is, a and b must be of the form $a = \ell \cdot 2^p$, $b = (\ell+1) \cdot 2^p$ for non-negative integers ℓ, p, thus defining an interval of length 2^p.

We say that their hash function, *Mixed Tabulation* (see section 2.2), is *locally uniform* due to the way it is uniform on keys whose hash values are close to each other – one could say that it is uniform on the interval $[a,b) \subseteq \mathcal{R}$. For this claim to make sense, we need to generalize our understanding of uniformity a bit. Clearly, $h(X)$ isn't uniformly distributed over $\mathcal{R}^{|X|}$ when we already know that $h(x) \in [a,b)$ for all $x \in X$. What uniformity gives us instead is a guarantee that $h(X)$ is uniformly distributed over $[a,b)^{|X|}$, as if this was the codomain of h.

Unfortunately, it isn't clear for all applications which interval $[a,b)$ will contain the important keys. We introduce the concept of a *query key* in [BBK+23] as a remedy, and discuss this addition in section 2.3.

2.1.4 Feasibility of Implementation

A concern we haven't touched upon till now is whether the hash functions covered can feasibly be implemented. The functions proposed by [PP08, DW03] for obtaining uniformity are rather complex – perhaps too complex to be useful in a practical setting, even for providing uniformity on smaller sets. As far as I'm aware, no one has successfully implemented these schemes and it thus seems doubtful that this can be done in a way that yields the throughput required of a modern system.

In section 2.2 we meet a family of remarkably simple functions, *Simple Tabulation*, that are uniform on very large sets of keys, under the condition that the key sets are free from a specific type of bad pattern (see section 2.3.1). For adversarially chosen keys this family may perform poorly, yet it forms the main building block for both Mixed Tabulation of [DKRT15], as well as for Tornado Tabulation which we introduce in the following section.

A second concern is the nature of the probabilistic guarantees given. The results on the functions covered up to this point all state their error probabilities in terms of $\mathcal{O}$-notation, thus placing the emphasis on how the probability scales with the parameters involved. Although this dependency is interesting in a theoretical context, an engineer implementing these systems won't have the luxury of letting parameters tend to infinity in order to bring down the error probability. Rather, they need to balance a budget between time and space usage and the associated error probability afforded by the hash function. In this context $\mathcal{O}$-notation carries little value, as it isn't even able to guarantee that the error probability will be strictly below 1 for a concrete set of parameters – only that the functions are useful for sufficiently large parameters, which may or may not be compatible with current hardware.

2.2 Tabulation-Based Hash Functions

In [BBK+23] we introduce *Tornado Tabulation Hashing*, a new family of functions based on fast table lookups and xor operations, building blocks dating back to Simple Tabulation of [Zob70].

A Simple Tabulation function interprets each key as be a sequence of c characters belonging to an alphabet Σ, and maps the key to a bitstring of length r. It does this through a sequence of lookups into tables of size $|\Sigma|$, returning the xor of the lookup values as the hash value.

Definition 2.3 (Simple Tabulation, [Zob70])**.** For positive integers c, r and universe Σ the *Simple Tabulation* function $h\colon \Sigma^c \to [2^r]$ is defined as

$$h(x) = \bigoplus_{i=1}^{c} T_i[x_i]\,,$$

where $x = (x_1, \ldots, x_c)$, and $T_1, \ldots, T_c$ is a collection of tables indexed by Σ, each entry containing an independently drawn value from $[2^r]$. $\oplus$ denotes the bitwise xor-operation.

Tornado Tabulation builds upon the ideas of previous tabulation based schemes like Double Tabulation [Tho13] and Mixed Tabulation [DKRT15]. Tornado Tabulation can be considered an iterated version of Simple Tabulation, computing a derived key $\tilde{x} \in \Sigma^{c+d}$ from $x \in \Sigma^c$, and then applying a Simple Tabulation function $h_{d+1}\colon \Sigma^{c+d} \to [2^r]$ to $\tilde{x}$.

We construct the derived key in the following way: The first $c-1$ characters of $\tilde{x}$, denoted $\tilde{x}_1, \ldots, \tilde{x}_{c-1}$, are the same as in x, and the c'th character $\tilde{x}_c$ is the xor of x_c and the result of a Simple Tabulation function applied to x. Each of the following d *derived characters* are computed through the use of (independent) Simple Tabulation functions, $h_i\colon \Sigma^{c+i-1} \to \Sigma$, applied to prefixes of the derived key:

Definition 2.4. For a key $x \in \Sigma^c$ the associated derived key $\tilde{x} \in \Sigma^{c+d}$ is defined as

$$\tilde{x}_i = \begin{cases} x_i & \text{for } i < c \\ x_c \oplus h_0(x_1 \dots x_{c-1}) & \text{for } i = c \\ h_{i-c}(\tilde{x}_1 \tilde{x}_2 \dots \tilde{x}_{i-1}) & \text{for } i \in \{c+1, \dots, c+d\} \end{cases},$$

where $\{h_i\}_{i=0}^{d}$ are independent Simple Tabulation functions $\Sigma^{c+i-1} \to \Sigma$.

Definition 2.5 (Tornado Tabulation, [BBK+23])**.** For positive integers c, r, d and universe Σ the *Tornado Tabulation* function $h\colon \Sigma^c \to [2^r]$ with d *derived characters* is defined as

$$h(x) = h_{d+1}(\tilde{x}),$$

where $h_{d+1}\colon \Sigma^{c+d} \to [2^r]$ is a Simple Tabulation function.

There's a lot to unpack in these definitions, so let us try to establish an overview of the parameters and the roles they play:

Σ Arguably the most important parameter to settle when deciding how to use Tornado Tabulation hashing. Not only does Σ decide the size of the tables stored, affecting space usage, it also defines the size of sets X we can claim uniformity on – our results apply to sets of size $|\Sigma|/2$.

Besides its direct effect on space usage, a connection to running time has been observed due to the relationship between space usage and cache efficiency.

c The length of the input keys. Must be set such that Σ^c corresponds to the universe of keys.

d Decides the number of derived characters to compute, and decreases the probability of leaving bad patterns in the derived keys (see section 2.3.1). d also has a direct impact on running time, as it influences the number of lookups performed.

r The function produces hash values of r bits. This parameter is important for the application that Tornado Tabulation is to support, but it has little impact on the function itself. From an implementation standpoint, r sets requirements on the data types used for implementing lookup tables and carrying intermediate values, which can influence the evaluation time.

For comparison, consider the definition of Mixed Tabulation, whose derived keys are computed in one go:

Definition 2.6 (Mixed Tabulation, [DKRT15])**.** For positive integers c, r, d and universe Σ, let $h_1\colon \Sigma^c \to \Sigma^d$ and $h_2\colon \Sigma^{c+d} \to [2^r]$ be independent Simple Tabulation functions. The *Mixed Tabulation* function $h\colon \Sigma^c \to [2^r]$ is defined as

$$h(x) = h_2(x \circ h_1(x)),$$

where $\circ$ denotes the concatenation of character-sequences.

Intuitively, the parameters given above play much the same roles between Mixed and Tornado Tabulation, the main difference being that Mixed computes all d derived characters through the use of a single function while Tornado Tabulation computes them in sequence.

An implementation faithful to the way Tornado Tabulation is defined above will end up evaluating d Simple Tabulation functions in sequence, leading to $\approx d(c+d)$ lookups. However, by observing that we keep using the same values to perform lookups into new tables – once a character has been computed, it doesn't change – we can implement Tornado Tabulation by performing just $c+d$ lookups in sufficiently large tables. This brings the evaluation time significantly closer to that of Mixed Tabulation (which can likewise be implemented in $c+d$ lookups). C-code implementing Tornado Tabulation in this way is included in appendix A.

2.3 Locally Uniform Hashing with Tornado Tabulation

The results we show for Tornado Tabulation in [BBK+23] closely mirror those for Mixed Tabulation, but with better dependencies on the parameters of the hash function and *explicit* probability bounds. Thus, whereas the error probabilities given in [DKRT15] are stated as asymptotic functions of the involved parameters, the guarantees we give can be computed based on a concrete set of parameters – showing that the function is actually useful when implemented with tractable parameters. A brief overview of the way the proofs differ is given in section 2.3.1.

We further extend the applicability of local uniformity by introducing the concept of a *query key*. Whereas the results of [DKRT15] required the "important" interval defining X to be fixed in advance, we can instead inspect the hash value of the query key and base our choice of interval on this value. To see the value of this addition (which is admittedly rather technical) I will briefly sketch our application to Linear Probing in section 2.3.2.

2.3.1 Techniques for Showing Uniformity

In [TZ12] it was shown that Simple Tabulation is uniform on a set X if and only if X is free of *zero sets*. A set of keys forms a zero set if, when hashed by Simple Tabulation, each table entry is touched an even number of times (possibly zero).

The simplest example of a zero set is the keys $x_1 = \alpha\alpha, x_2 = \alpha\beta, x_3 = \beta\alpha$, and $x_4 = \beta\beta$. By inspecting the lookups performed when the Simple Tabulation function h computes their hash values, it is seen that $h(x_4) = \oplus_{i=1}^{3} h(x_i)$. For any set of three keys, on the other hand, the keys can be "ordered" in such a way that x_2 depends on a table entry not touched by x_1, and x_3 depends on a table entry not used by any of the two – hence the three keys are hashed independently of each other, and it is seen that Simple Tabulation is 3-independent.

As a Tornado Tabulation function h (like Mixed Tabulation) can be decomposed into a preprocessing step (computing the derived keys) and a Simple Tabulation function, showing that h is uniform on X boils down to showing that the set of derived keys, $\tilde{X}$, produced by the first step doesn't contain any zero sets.

In [BBK+23] we exploit that a zero set among the derived keys would imply that the derived keys can be grouped into pairs of two, with each pair sharing the same derived character $\tilde{x}_{c+d}$ in the final position. This is shown to happen with probability at most $\mathcal{O}(1/|\Sigma|)$, but as this must happen at each level (and these events are, to some extent, independent), the risk of producing a zero set drops by $\mathcal{O}(1/|\Sigma|)$ for each additional derived character added to the keys.

For comparison, the proof of [DKRT15] relies on the stronger property of *peelability*. Peelability extends the argument given for 3-independence at the top of this section, and entails that the derived keys $\tilde{X}$ can be ordered in such a way that, for all $i \leq |X|$, $h(\tilde{x}_i)$ accesses a table entry that no prior key has used – thus certifying that $h(\tilde{x}_i)$ is independent from all prior hash values. At a high level, they show that this is unlikely by considering the derived characters in two specific positions, and arguing that the characters in these positions (across all derived keys) must, like in our proof, form a particular structure if they are not peelable. For a pair of positions, this structure will occur with probability roughly $\mathcal{O}(1/|\Sigma|)$. Adding more derived characters will decrease the error probability, but only by a factor of $\mathcal{O}(1/|\Sigma|)$ for every *two* derived characters, due to their structure relying on two positions.

For a fixed number of table lookups, Tornado Tabulation thus achieves a significantly smaller error probability.

2.3.2 Application to Linear Probing

Linear Probing is a simple method for implementing a hash table T. For inserting an element x, a location $T[h(x)]$ is computed using a hash function h, and x is inserted into this cell if it not already occupied. If a different element occupies the cell, we will continue probing the following cells, $T[h(x)+1], T[h(x)+2], \ldots$, until an available cell is found, into which we place x.

Intuitively, there is some *local* nature to Linear Probing, seeing as only a contiguous range of entries are probed during insertion. However, it is not known in advance which part of the table is of interest to us; this wholly depends on the value $h(x)$. What we *do* know, is that elements whose hash values are close to $h(x)$ will play a more direct role in the placement of x than elements with hash values in the opposite end of the table.

With our introduction of a *query key* in [BBK+23] we can define the set of "important keys" X to be the preimage centered around $h(x)$, namely $h^{-1}([h(x)-\Delta, h(x)+\Delta])$ for some appropriately chosen Δ, and thus capture the elements having the largest impact on our insertion procedure.

Combine this with work done by Pătraşcu and Thorup [PT12] showing that, for Linear Probing with Simple Tabulation (which Tornado Tabulation is a generalization of), the longest interval of occupied cells will, whp, be $\mathcal{O}(\log n)$, where n is the number of inserted elements. Setting $\Delta = \Theta(\log n)$, we thus arrive at a situation where the insertion of x is entirely defined by the distribution of $h(X)$ in the table.

When Tornado Tabulation is uniform on X the insertion of x will (whp) proceed exactly as if hash values of all inserted elements had been computed with fully random hashing – conditioning on the event where the fully random hash function maps exactly the elements of X into our chosen interval $[h(x)-\Delta, h(x)+\Delta]$. This is an unlikely event, and makes for a difficult analysis. Instead, we consider the case where *at least* $|X|$ elements are mapped into the interval. Then the two experiments (one using Tornado Tabulation, the other employing fully random hashing) can be coupled, and it is seen that the number of probes performed when inserting x in one experiment is stochastically dominated by the insertion time of the other.

It should be noted, as mentioned in section 2.1.1, that Linear Probing implemented with 5-independent hashing [PPR09] gives the same performance as when implemented with fully random hashing, asymptotically. That is, up to some constant multiplicative factor. What we show is that Linear Probing with Tornado Tabulation performs *as if* fully random hashing had been applied, with only additive lower order terms in excess. We do pay some small error terms in order to establish the comparison given above, but when we have reduced Linear Probing to the setting where we claim uniformity we can reuse the classic analysis done by Knuth [Knu63].

2.4 Sampling-Based Estimation

In our second paper, we wish to show concentration bounds for Tabulation Hashing. That is, we let X be the set of keys hashed to a specific value (or range of values) and wish to bound the deviation of $|X|$ from $\mu = \mathbf{E}[|X|]$. We say that keys in X have been *selected*.

Had the hash values been decided through the use of fully random hashing, the indicator variables $[x \in X]$ would be independent for all keys x. In this case we could have applied the classic Chernoff bound, giving

$$\Pr[||X| - \mu| > \delta\mu] < 2\exp(-\delta^2\mu/3)$$

for all $\delta \leq 1$. When a practical hash function is used, however, the indicators are no longer independent and we need to establish such tail bounds through other means.

While the Chernoff bound describes the behavior of sums of independent indicator variables, we can't generally expect tail bounds for practical hash functions to be as strong. Yet, we already showed the "upper tail" in [BBK+23], bounding the probability that X significantly exceeds its expected size, which we also relied on in the argument presented in section 2.3.2. The subject of [ABH+24] is thus the "lower tail", bounding the probability that fewer elements than expected are selected.

Thus we can finally implement the applications covered in [DKRT15] (such as MinHash) as well as other sampling problems like Bernoulli Sampling. For the applications in [DKRT15] they showed a single concentration bound for Mixed Tabulation, bounding the probability that the number of selected keys deviates from its expectation by a factor of $\delta = \tilde{\mathcal{O}}\left(\sqrt{(\log|\Sigma|)/|\Sigma|}\right)$. We instead show a general tail bound that can be evaluated at user-specified values of δ.

Our upper tail bound in [BBK+23] followed from a simple rewriting of the proof of the classic Chernoff bound, and gave the same expression as the usual one-sided Chernoff bound. Morally, the upper tail bound is a statement about the behavior of the keys in X – those that we claim uniformity on with high probability – and thus it seems appropriate that these are well-behaved and abide by the same bound as independent indicators.

The lower tail, on the other hand, is about bounding how many keys fall *outside* of X, and we generally know very little about these keys. Establishing a lower tail bound ended up requiring significant effort and resulted in an expression with worse constants than those known from Chernoff's bound – yet still constants. This makes Tornado Tabulation the first realistic hash function with Chernoff-style tail bounds containing explicit constants.

2.4.1 Showing Lower Tails for Tornado Tabulation

In order to prove our lower tail bound we wish to find a way of expressing $|X|$, the number of selected elements, in terms of indicators that are independent.

At a high level, our setup is as follows: For each $\alpha \in \Sigma$, let Y_α be the set of *all* keys whose final derived character is α, and let $X_\alpha \subseteq Y_\alpha$ be the subset of those keys that are selected (that is, whose hash value is contained in our interval of choice). We refer to the X_α as *bins*. For each $i \in \{1, 2, \dots\}$, let $S_i = |\{\alpha \colon |X_\alpha| \geq i\}|$. S_i thus counts the number of bins containing at least i keys. This allows us to express the number of selected keys as $|X| = \sum_{i=1}^{\infty} S_i$,

Next, observe that the computation of derived keys is what decides the sets $\mathcal{Y} = \{Y_\alpha\}_{\alpha\in\Sigma}$ and that, when conditioning on Y_α, the inclusion of elements in X_α is entirely decided by the final application of Simple Tabulation. Seeing as each Y_α depends on a unique character α, the process of choosing X_α from Y_α is independent from the computation of all other $X_\beta \subseteq Y_\beta$ (again, conditioning on $\mathcal{Y}$ already being settled).

When conditioning on $\mathcal{Y}$, the X_α's are thus independent, and the sum S_i becomes a sum of independent indicator variables. Our goal is to bound the deviation of $|X|$ from $\mathbf{E}[|X|] = \mu$ by bounding the differences between S_i and $\mathbf{E}[S_i]$. With S_i being a sum of independent indicators, we can bound its deviation through the use of a regular Chernoff bound. But due to the conditioning on variables $\mathcal{Y}$, S_i is no longer concentrated around $\mathbf{E}[S_i]$ but must instead be compared to the conditional expectation $\mathbf{E}[S_i \,|\, \mathcal{Y}]$. Fortunately, $\sum_i \mathbf{E}[S_i \,|\, \mathcal{Y}] = \sum_i \mathbf{E}[S_i] = \mu$, so our overall strategy for bounding the deviation still holds, but we need a way to control the conditional expectations as they influence the concentration of the S_i's. Specifically, the event $(S_i < \mathbf{E}[S_i \,|\, \mathcal{Y}] - \Delta)$ is more likely the larger the conditional expectation is. Somewhat counterintuitively, we thus need to establish an *upper* bound on $\mathbf{E}[S_i \,|\, \mathcal{Y}]$ in order to get the desired lower bound on S_i.

To bound the conditional expectation, we first give an upper bound on $S_i = \sum_\alpha [|X_\alpha| \geq i]$. The indicator variables are not independent (as one bin being large suggests that other bins

are small), but we can still bound their sum through a Chernoff bound. Finally, we again use that $S_i = \sum_\alpha [|X_\alpha| \geq i]$ is a sum of independent indicators when conditioned on $\mathcal{Y}$, and thus its median equals $\mathbf{E}[S_i \mid \mathcal{Y}] \pm 1$ (by [JS68]). This allows us to "translate" any upper bound on S_i to a bound on $\mathbf{E}[S_i \mid \mathcal{Y}]$, giving us the final piece needed for our bound.

To summarize: For each *layer* $i \geq 1$, we give an upper bound on S_i, translate it to a bound on the conditional expectation $\mathbf{E}[S_i \mid \mathcal{Y}]$, and use it to give a bound on $\Delta_i = \mathbf{E}[S_i \mid \mathcal{Y}] - S_i$. Summing the Δ_i's, we arrive at a bound on the deviation between $|X|$ and μ. The concrete way we perform these bounds depends on the layer in question, namely the relationship between $\mathbf{E}[S_i]$ and the error probability we are trying to show.

2.5 What's Next?

Tornado Tabulation is showing some promise as a hash function that could find practical application, but there's always more work to be done. The following is a collection of ideas for next steps in Tornado Tabulation as well as realistic hashing more generally.

Running Time An experimental study investigating the running time of several variants of tabulation-based hashing was done by [ADK+22], adding to their credence as practically relevant families of hash functions. Despite their similarities, preliminary benchmarks suggest that Tornado Tabulation can't quite match the running time of Mixed Tabulation. This owes to the fact that Tornado Tabulation is of a very iterative nature, whereas Mixed Tabulation can perform several lookups in parallel. Still, Tornado Tabulation is observed to outperform the evaluation of polynomials of modest degree, with the evaluation time of Tornado Tabulation matching that of performing just a handful of multiplications, depending on the parameters used.

It is not known if a more careful analysis of Mixed Tabulation could lead to the same stronger guarantees as those we've shown for Tornado Tabulation, or if the iterative approach of the latter is somehow inherent to our results. Preliminary work suggests that the number of sequential steps of Tornado Tabulation can be brought down somewhat, bringing its performance closer to that of Mixed Tabulation. Answers to these questions, along with a more systematic investigation of the runtime of Tornado Tabulation, would be interesting additions to the literature on realistic hashing.

Poisson Sampling A next step, building on the work in [ABH+24], would be to implement Poisson Sampling with Tornado Tabulation. As opposed to Bernoulli Sampling, which can be implemented with our concentration bounds, each item in the stream now has its own *threshold* between 0 and 1, corresponding to the probability that the element in question should be included in the sample. Poisson Sampling forms the foundation of Priority Sampling [DLT07], which is used to estimate subset sums, in a setting where the elements of the stream all carry different weights. If we can generalize the obtained concentration bound to also apply to sums of weighted variables (rather than just indicators), it seems feasible to obtain Poisson Sampling. The proof we present in [ABH+24] is of a very combinatorial nature, though, which makes it difficult to adapt to the weighted setting.

Non-Dyadic Intervals Although the technical details have mostly been brushed aside, we briefly touched on the topic of intervals in section 2.1.3: that uniformity is only granted on dyadic intervals for Mixed Tabulation, and same goes for our results on Tornado Tabulation. Any interval can be covered by a small collection of dyadic intervals, but this still means that our

keys are only uniformly distributed within the dyadic interval that they are placed. Specifically, this allows for dependencies between which elements fall into which dyadic interval.

It would make for a cleaner presentation, and much simpler application, if we could get rid of this limitation. It arises quite naturally, however, due to the way our proofs are structured, with selection of keys based on a fixed subset of the bits in their hash values.

Invertible Bloom Filters The Invertible Bloom Filter (IBF) is a data structure for maintaining a *small* set under insertions and deletions, introduced by [EG11]. Usually, the size of a data structure maintaining a set under such updates would be expected to take up space proportional to the set stored, but the trick lies in only requiring the set to be retrievable when the set is of cardinality less than some parameter s. Thus the space usage can be kept small at all times, regardless of the maximum size of the set built by the stream (which could be significantly larger than s). Closely related is the Invertible Bloom Lookup Table (IBLT), which maintains key-value pairs through the use of the same techniques [GM11].

Both of these structures rely on *peelability* to allow for decoding. This property has also been central to much work on tabulation based hashing, as peelability serves as a certificate that a key set will be hashed uniformly (see section 2.3.1). As the proof of uniformity in [DKRT15] is based on showing peelability of hash values computed by a Simple Tabulation function, the simplest version of the IBF/IBLT can be implemented using Simple Tabulation hashing. This version, however, only allows for storing simple sets, and doesn't support *false deletions* where elements are removed without a corresponding insertion.

When allowing these false deletions, along with multiple insertions, the IBF becomes a linear sketch and can be used to compute the symmetric difference between a pair of data streams, assuming that the difference is less than s keys. To add this feature, hash codes can be added to each cell of the structure. Simple Tabulation is ill-suited for computing these hash codes, but we have work suggesting that the local uniformity of Tornado Tabulation could be sufficient for implementing the stronger structure.

An independent direction on realistic IBLTs, as recently pursued by [FLOS24], is to design variations on the classic IBLT, such that it better lends itself to practical hash functions, requiring $O(\log \log s)$-independent hashing.

Chapter 3

Online Sorting

In this section I discuss and introduce the problem of (approximately) sorting numbers online, the topic of appendix C, *Online Sorting and Online TSP: Randomized, Stochastic, and High-Dimensional* ([ABB+24]). I presented a shorter version of this paper (without the appendix) at ESA '24, that version of the paper is available online at https://doi.org/10.4230/LIPIcs.ESA.2024.5. The full version found in appendix C matches the version available on arXiv, https://doi.org/10.48550/arXiv.2406.19257.

3.1 The Problems

The main problem of study in appendix C is *Online Sorting*, introduced by by Aamand, Abrahamasen, Beretta, and Kleist [AABK23] as a tool for proving lower bounds for online Strip Packing.

Definition 3.1 (Online Sorting, [AABK23])**.** In *Online Sorting* a stream of n real values arrive one by one. Each value must, immediately and irrevocably, be placed into an array T of n cells such that the objective

$$\sum_{i=1}^{n-1} |T[i] - T[i+1]|$$

is minimized.

Sorting the input (either in non-decreasing or non-increasing order) minimizes the objective, hence the name Online Sorting. It is easily seen, however, that we can't reliably sort numbers when we have to irrevocably assign them locations in the array in this online manner. We thus study approximations instead.

Throughout this chapter (and in most of [ABB+24]) we assume that all values are between 0 and 1, with both values included in the instance. Hence the optimal value will always be 1, which makes for a simpler analysis of approximation ratios. In [ABB+24] we show that we can remove this assumption without affecting the asymptotic cost of our algorithm.

A second variant of Online Sorting, also introduced by [AABK23], is where the array has size γn for some $\gamma > 1$. Here the cost function is adapted in the obvious way, summing differences between *filled* cells.

We further study a generalization of Online Sorting which we call *Online TSP*. Note that Online Sorting corresponds to Online TSP in one dimension.

Definition 3.2 (Online TSP, [ABB+24])**.** In *Online TSP* a stream of n points from $\mathbb{R}^d$ arrive one by one. Each point must, immediately and irrevocably, be placed into an array T of n cells such that the sum of Euclidean distances,

$$\sum_{i=1}^{n-1} \|T[i] - T[i+1]\|,$$

is minimized.

We show an $\mathcal{O}(\sqrt{n \log n})$-competitive algorithm for Online TSP in constant dimensions in [ABB+24], but will not treat the problem further in this thesis.

3.2 Previous Work

In [AABK23] a deterministic $\mathcal{O}(\sqrt{n})$-competitive algorithm was presented, along with a matching *adaptive* lower bound, showing that no deterministic algorithm could beat $\Omega(\sqrt{n})$.

Their lower bound construction is rather simple: At any point during the execution of an algorithm $\mathcal{A}$, denote the value stored in each cell $T[i]$ an *endpoint*, if any of $T[i-1]$ and $T[i+1]$ are unoccupied. Let a value be *expensive* if is at least $1/\sqrt{n}$ from all endpoints.

Their expensive input is constructed as follows:

1. If an expensive value exists, present it as the next element of the stream.
2. If no value in $[0, 1]$ is expensive, fill the remainder of the stream with all `0`'1 or all `1`'s, whichever would be more expensive.

If the construction ever enters the second step, at least $\Omega(\sqrt{n})$ endpoints are present, and hence the stream of `0`'1 or `1`'s will incur cost $\Omega(\sqrt{n})$. Likewise, if the construction stays in the first step, each neighboring pair will incur cost at least $1/\sqrt{n}$, leading to a total cost of roughly $\sqrt{n}$. Hence any deterministic algorithm $\mathcal{A}$ will, in the worst case, produce a solution of cost $\Omega(\sqrt{n})$.

The main open problem left by [AABK23] was whether a randomized algorithm could do better. In [ABB+24] we show that no such algorithm exists.

3.3 Contributions

We prove a number of results in [ABB+24], related to different variations and settings of Online Sorting and Online TSP. In the following sections I give an overview of my personal favorites.

3.3.1 Randomized Algorithms

We show that no randomized algorithm for Online Sorting can be $o(\sqrt{n})$-competitive by constructing an expensive distribution of inputs. The distribution builds on the ideas used in the lower bound of [AABK23], as discussed in section 3.2.

- With probability $1 - 1/\sqrt{n}$: present the values $\{i/\sqrt{n}\}_{i=1}^{\sqrt{n}}$ as the next $\sqrt{n}$ elements of the input sequence.
- If the experiment above fails, fill the remainder of the input with all `0`'s or all `1`'s, each chosen with equal probability.

The analysis of this distribution relies entirely on the number of endpoints (or, equivalently, "gaps") produced by the algorithm while processing the input. When inserting the sequence of elements from case 1 above, it is beneficial to have $\Omega(\sqrt{n})$ endpoints, such that most values can be placed next to an identical element. The cost of inserting the stream of `1`'s/`0`'s, however, scales linearly with the number of endpoints.

Thus no strategy will be able to handle both cases cheaply. Due to the probability of entering each of the two cases all deterministic algorithms will thus, in expectation, give solutions of cost $\Omega(\sqrt{n})$ on this input distribution. By Yao's Lemma this rules out the existence of a better randomized algorithm for Online Sorting.

3.3.2 Stochastic Input

We study Online Sorting under a new model where the input consists of independently and uniformly distributed values in $[0, 1]$. When designing algorithms for this model we've drawn some inspiration from the study of data structures. Letting array T (or a subarray) represent the number line from 0 to 1, each incoming element maps naturally to a specific cell of T. When values are independent and uniform, this gives the same sequence of accesses to T as when designing hash tables under the assumption that elements are assigned fully random hash values (see chapter 2 for more on this assumption and why it is otherwise usually discouraged).

Our first result in this direction is an algorithm $\mathcal{A}$ which, with probability at least $1 - 2/n$, produces a solution of cost $\tilde{\mathcal{O}}(n^{1/4})$.

$\mathcal{A}$ partitions T into M *buckets* and a small *backyard*. Each bucket is a contiguous subarray of T, with the i'th bucket meant to hold values in the interval $[i/M, (i+1)/M)$. For each element x arriving, $\mathcal{A}$ attempts to place it in the corresponding bucket. If the bucket is already full, x is instead placed in the backyard.

As the set of elements placed within a bucket are themselves independent and uniformly distributed in $[i/M, (i+1)/M)$, we can apply $\mathcal{A}$ recursively in the bucket. For deciding the locations of elements within the backyard, the deterministic Online Sorting algorithm of [AABK23] is applied.

The algorithm terminates correctly if all buckets are filled with appropriate elements. In other words, the algorithm only fails if too few elements of some size is found in the input. The probability of this event can be bounded by a union bound over all buckets, applying a standard Chernoff bound for each corresponding interval of $[0, 1]$.

Larger Arrays A second result is in the setting of larger arrays, where not all cells of T will be filled. Specifically, we assume that $|T| = \gamma n$ for a constant $\gamma > 1$. We present an algorithm $\mathcal{A}$ which, in this setting, produces a solution of cost $\mathcal{O}(1 + 1/(\gamma - 1))$ with high probability.

$\mathcal{A}$ is heavily inspired by Linear Probing, a simple way of implementing hash tables (Linear Probing has appeared once before in this thesis, see section 2.3.2). We set aside a buffer at the end of T, and let the remainder of T (call it T') represent the number line from 0 to 1. Then each value x maps into a specific cell of T', computed by $h(x) = \lfloor x/|T'| \rfloor$. Algorithm $\mathcal{A}$ first attempts to insert x into $T[h(x)]$. If this fails due to a different element occupying the cell, $T[h(x) + 1]$ is attempted, and so forth. Note that this process may insert x into the buffer, but with high probability we will never wrap around to $T[0]$ during insertion. In the worst case, if we do need to wrap around, we conservatively bound the cost of the solution at n.

To analyze $\mathcal{A}$, we rely on observations linking the cost of our solution to the procedure of Linear Probing: Consider the point in time where an element x is inserted into a cell $T[i]$ where $T[i+1]$ is already occupied. The cost $|T[i] - T[i+1]|$ incurred by this insertion, can be bounded by $(i - h(x))/|T'|$, which is $1/|T'|$ times the number of probes performed during insertion.

A similar argument applies to the cost $|T[i] - T[i+1]|$ where $T[i]$ is filled before $T[i+1]$. The classic analysis by Knuth [Knu63] bounds the expected number of probes during insertion, and hence allows us to bound the expected cost of the solution produced by $\mathcal{A}$.

3.4 Open Questions

We raise a number of open questions:

- Can we find a better solution in the case of stochastic input?
- Can we extend the arguments of section 3.3.2 to cover other input distributions?
- Can we say anything about the setting of larger arrays if the input is adversarial?

And finally, can a useful connection be made to *Online List Labeling* (or *Order Maintenance*)? Online List Labeling is, in some sense, the opposite of Online Sorting. Again, elements arrive sequentially, but their positions in T are no longer final. Instead the (partial) solution must be sorted at every step of the algorithm, while minimizing the total number of times elements have to be moved. First treated in the early 1980's [IKR81, Die82], this is a significantly older problem with a rich literature. Can any interesting connections between the two be established?

Chapter 4

Constrained Correlation Clustering

In this chapter I present and discuss the contents of appendix D, *A Faster Algorithm for Constrained Correlation Clustering*. At STACS '25 I presented a shortened version of this paper, which is available online at https://doi.org/10.4230/LIPIcs.STACS.2025.32. The full version given in appendix D matches the version available on arXiv (https://doi.org/10.48550/arXiv.2501.03154).

I first introduce the problems we study before presenting our results on Constrained Correlation Clustering in section 4.2 and, in section 4.3, presenting one of our results on Node-Weighted Correlation Clustering, a new problem we introduce in [FKKT25].

4.1 The Problems

Correlation Clustering, introduced by Bansal, Blum, and Chawla [BBC04], is a clustering problem on graphs. The input consists of a vertex set V which must be clustered (that is, partitioned) along with a *preference* for each pair of vertices, given as a set of edges E. If $\{u, v\} \in E$ vertices u, v prefer to be clustered together (that is, to be included in the same cluster), while $\{u, v\} \notin E$ denotes that u, v prefers to be in different clusters. The objective of Correlation Clustering is to find a clustering of the vertices violating as few of these preferences as possible.

When $\mathcal{C}$ is a clustering we say that it induces the edge set $E_{\mathcal{C}} = \bigcup_{C \in \mathcal{C}} \binom{C}{2}$, corresponding to the set of preferences satisfied by $\mathcal{C}$. Our objective can then equivalently be stated as finding the clustering $\mathcal{C}$ which minimizes the symmetric difference between $\mathcal{C}$ and E.

Definition 4.1 (Correlation Clustering, [BBC04])**.** A graph $G = (V, E)$ is given as input. Return a clustering $\mathcal{C}$ of V minimizing the symmetric difference $|E_{\mathcal{C}} \Delta E|$ between E and the edge set induced by $\mathcal{C}$.

One trait setting Correlation Clustering apart from other popular clustering objectives like k-median is that the number of clusters isn't specified as part of the input. Thus $\mathcal{C} = \{V\}$ and $\mathcal{C} = \{\{v_1\}, \ldots, \{v_n\}\}$ – a single cluster, or n singleton clusters – are both valid solutions to Correlation Clustering, but the optimal solution is likely found somewhere between these two extremes.

Constrained Correlation Clustering (CCC) was introduced shortly after by van Zuylen, Hegde, Jain, and Williamson [vZHJW07] and promotes a subset of the preferences to *hard constraints*.

That is, a clustering *must* satisfy all constraints, while still minimizing the number of violated preferences.

Definition 4.2 (Constrained Correlation Clustering, [vZHJW07])**.** A graph $G = (V, E)$ is given as input, along with hard constraints $F \subseteq E$ and $H \subseteq \left(\binom{G}{2} \setminus E\right)$. Return a clustering $\mathcal{C}$ of V minimizing the symmetric difference $|E_{\mathcal{C}} \Delta E|$ between E and the edge set induced by $\mathcal{C}$, such that $(F \cup H) \cap (E_{\mathcal{C}} \Delta E) = \emptyset$.

We say that the set F denotes pairs of *friendly* vertices (which *must* be clustered together) while the pairs in H are said to be *hostile* (and must be kept apart).

While introducing the problem, [vZHJW07] also gave a 3-approximation algorithm to CCC, running in time $\mathcal{O}(n^{3\omega})$, with $\omega \geq 2$ being the matrix multiplication constant. Our main result in [FKKT25] is a significantly faster algorithm computing a 16-approximation in $\tilde{\mathcal{O}}(n^3)$ time. I present the ideas behind this algorithm in section 4.2.

In [FKKT25] we further introduce a new variant of Correlation Clustering, where each vertex carries a weight denoting how important its preferences are:

Definition 4.3 (Node-Weighted Correlation Clustering, [FKKT25])**.** A graph $G = (V, E)$ with vertex weights $w \colon V \to \mathbb{R}^+$ is given as input. Return a clustering $\mathcal{C}$ of V minimizing

$$\sum_{\{u,v\} \in E_{\mathcal{C}} \Delta E} w(u) \cdot w(v) \,.$$

We present a linear time randomized algorithm computing a 3-approximation to NWCC in [FKKT25]. In section 4.3 I give an overview of this algorithm.

4.2 A Faster Approximation Algorithm

Our algorithm can be broken down into three steps:

1. Transform the instance to one having *nice* neighborhoods – without significantly altering the cost of the optimal solution.
2. "Forget" the hard constraints.
3. Run a Pivot algorithm on the produced (unconstrained) instance.

The class of Pivot algorithms is defined in section 4.2.1. We say that the neighborhoods of the graph are nice if they satisfy the following:

1. If u and v are friendly they have the same neighborhood.
2. If u and v are hostile their neighborhoods are disjoint.

We obtain these properties through a sequence of simple modifications, like computing the transitive closure of all friendly constraints and removing edges between components containing a hostile pair. The most advanced step is a final rounding step to unify the neighborhoods of friendly vertices.

The change to the optimal solution's cost is bounded by the number of modified edges, which we bound by $(1+\sqrt{5})$ times the optimal cost through an (admittedly cumbersome) case analysis.

The properties of nice neighborhoods are sufficient to guarantee that any solution produced by a Pivot algorithm (which only knows the preferences of the modified graph) will produce a solution that adheres to all hard constraints, see section 4.2.2.

The algorithm produces a solution of cost at most $\left(1+\sqrt{5}+(2+\sqrt{5})\cdot\alpha\right)$ times the optimal solution, with α being the approximation factor of the chosen Pivot algorithm. The algorithm runs in time $\mathcal{O}(nm)$ plus the time for executing the Pivot algorithm. Two Pivot algorithms are presented in the coming section, both of which yield a 16-approximation when used with our algorithm. One is a linear time randomized algorithm due to [ACN08] while the second is a new deterministic algorithm running in time $\tilde{\mathcal{O}}(n^3)$.

4.2.1 Pivoting Algorithms

The class of Pivot algorithms is a family of approximation algorithms for (unconstrained) Correlation Clustering with the structure given in algorithm 1. That is, a Pivot algorithm chooses a pivot (according to some rule), adds its neighborhood as a new cluster, removes the cluster from the graph, and recurses on what remains.

Algorithm 1: The generic Pivot algorithm.
Input: Graph $G = (V, E)$
$\mathcal{C} \leftarrow \emptyset$
while $V \neq \emptyset$ **do**
  Choose a pivot node u **(but how?)**
  Add a cluster containing u and all of its neighbors to $\mathcal{C}$
  Remove u and its neighborhood from G
Output: Clustering $\mathcal{C}$

The first Pivot algorithm was CC-Pivot presented by [ACN08]. It chooses its pivot uniformly at random among the vertices of the (remaining) graph at each step. This gives a 3-approximation in expectation, running in linear time.

In [FKKT25] we introduce a deterministic alternative, which picks its pivot based on the solution of a covering LP – that is, a linear program of the form $\min_x \{cx : Ax \geq b\}$ for non-negative vectors b, c, x, and matrix A. This algorithm gives a $(3+\epsilon)$-approximation for any constant $\epsilon > 0$ and, as covering LPs can be solved efficiently [WRM16], runs in time $\tilde{\mathcal{O}}(n^3)$.

4.2.2 Correctness of Pivoting

To see that Pivot algorithms indeed produce valid solutions when used in the way presented in the preceding section, we note the following:

1. If a friendly constraint $\{u, v\}$ is violated, a pivot was chosen which was only neighboring one of u and v. But these vertices have the exact same neighborhood after our transformation procedure.

2. If a hostile constraint $\{u, v\}$ is violated, a pivot was chosen such that both u and v were in its neighborhood. But no such vertex exists, as their neighborhoods are disjoint.

Hence no hard constraints of the input will be violated when a Pivot algorithm is applied to the modified graph with nice neighborhoods.

4.2.3 A Lower Bound for Pivoting

In [FKKT25] we show that no Pivot algorithm will beat an approximation ratio of 3 on all inputs through a very simple counterexample: For even n, consider the complete graph on n

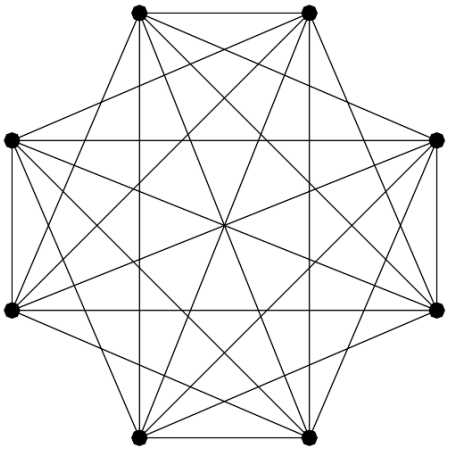


Figure 4.1: Construction of the lower bound described in section 4.2.3.

vertices with a perfect matching removed (see fig. 4.1 for the case $n = 8$). The optimal clustering for this graph is to put all n vertices in a single cluster, at cost $n/2$ due to the missing matching.

A Pivot algorithm, no matter what decision rule is applied, will produce a cluster of $n - 1$ vertices and a singleton cluster. This will incur a total cost of $n/2 - 1$, for the missing edges in the large cluster, plus $n - 2$ edges crossing the clusters.

Thus *any* Pivot algorithm will, for large n, end up with a solution at thrice the cost of the optimal.

4.3 Node-Weighted Correlation Clustering

First, assume that all weights are integers. Then Node-Weighted Correlation Clustering can easily be reduced to an instance of Constrained Correlation Clustering by replacing each vertex v with $w(x)$ copies, all having the same neighborhood and all connected to each other. Adding hard constraints between all such pairs of copies prevents this "supernode" from being split by the solution, and each solution to the CCC-instance corresponds to a solution of same value to the original NWCC-instance, and vice versa.

In fact, we can remove the hard constraints and run CC-Pivot (or any other Pivot algorithm) directly on this graph as Pivot algorithms won't split the supernodes (see section 4.2.1), thus obtaining a 3-approximation as desired.

The issue at this point is that the produced graph can be significantly larger than the original instance, and we can't afford to run an algorithm on this huge graph. Instead, we can simulate CC-Pivot by sampling each vertex v of the input graph with probability $w(v)/\sum_{u \in V} w(u)$; this is equivalent to running CC-Pivot on the produced graph.

In order to implement this simulation procedure efficiently, we design a data structure supporting our sampling operation, as well as an operation for removing the sampled elements and their neighbors. The data structure allows us to sample and remove all elements in linear time, with high probability. The data structure supports non-integer weights, generalizing the procedure to weight functions $w \colon V \to \mathbb{R}^+$.

4.4 Open Questions

The main open question we leave in this work is whether we can approximate Constrained Correlation Clustering to within a better factor while keeping the running time low. This goal could potentially be reached through a better analysis of the algorithm presented in section 4.2 as we don't have an example showing tightness of the approximation factor given.

Second, can we give approximation algorithms for other flavors of weighted Correlation Clustering? The (very general) setting where arbitrary weights are put on the *edges* was approximated to within a factor $\mathcal{O}(\log n)$ by [DEFI06], who also showed a connection to Multicut which makes further progress unlikely. But we now have approximation algorithms for graphs with edge-weights in $\{1, \infty\}$ (CCC) and for graphs where weights are the product of node-weights. What other objectives could lead to interesting results?

Bibliography

[AABK23] Anders Aamand, Mikkel Abrahamsen, Lorenzo Beretta, and Linda Kleist. Online sorting and translational packing of convex polygons. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 1806–1833. SIAM, 2023.

[ABB+24] Mikkel Abrahamsen, Ioana O. Bercea, Lorenzo Beretta, Jonas Klausen, and László Kozma. Online sorting and online TSP: randomized, stochastic, and high-dimensional. In Timothy M. Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 308 of *LIPIcs*, pages 5:1–5:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[ABH+24] Anders Aamand, Ioana O. Bercea, Jakob Bæk Tejs Houen, Jonas Klausen, and Mikkel Thorup. Hashing for sampling-based estimation. arXiv:2411.19394, 2024.

[ACN08] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 55(5):23:1–23:27, 2008. Announced in STOC 2005.

[ADK+22] Anders Aamand, Debarati Das, Evangelos Kipouridis, Jakob Bæk Tejs Knudsen, Peter M. R. Rasmussen, and Mikkel Thorup. No repetition: Fast and reliable sampling with highly concentrated hashing. *Proc. VLDB Endow.*, 15(13):3989–4001, 2022.

[BBC04] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Mach. Learn.*, 56(1-3):89–113, 2004.

[BBK+23] Ioana O. Bercea, Lorenzo Beretta, Jonas Klausen, Jakob Bæk Tejs Houen, and Mikkel Thorup. Locally uniform hashing. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 1440–1470. IEEE, 2023.

[Bro97] Andrei Z. Broder. On the resemblance and containment of documents. In Bruno Carpentieri, Alfredo De Santis, Ugo Vaccaro, and James A. Storer, editors, *Compression and Complexity of SEQUENCES 1997, Positano, Amalfitan Coast, Salerno, Italy, June 11-13, 1997, Proceedings*, pages 21–29. IEEE, 1997.

[DEFI06] Erik D. Demaine, Dotan Emanuel, Amos Fiat, and Nicole Immorlica. Correlation clustering in general weighted graphs. *Theor. Comput. Sci.*, 361(2-3):172–187, 2006.

[Die82] Paul F. Dietz. Maintaining order in a linked list. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, page 122–127, New York, NY, USA, 1982. Association for Computing Machinery.

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

[DLT07] Nick Duffield, Carsten Lund, and Mikkel Thorup. Priority sampling for estimation of arbitrary subset sums. *Journal of the ACM (JACM)*, 54(6):32–es, 2007.

[DW03] Martin Dietzfelbinger and Philipp Woelfel. Almost random graphs with simple hash functions. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '03, page 629–638, New York, NY, USA, 2003. Association for Computing Machinery.

[EG11] David Eppstein and Michael T Goodrich. Straggler identification in round-trip data streams via newton's identities and invertible bloom filters. *Knowledge and Data Engineering, IEEE Transactions on*, 23(2):297–306, 2011.

[FEFGM07] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédédric Meunier. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *In Analysis of Algorithms (AOFA)*, 2007.

[FKKT25] Nick Fischer, Evangelos Kipouridis, Jonas Klausen, and Mikkel Thorup. A faster algorithm for constrained correlation clustering. In Olaf Beyersdorff, Michal Pilipczuk, Elaine Pimentel, and Kim Thang Nguyen, editors, *42nd International Symposium on Theoretical Aspects of Computer Science, STACS 2025, March 4-7, 2025, Jena, Germany*, volume 327 of *LIPIcs*, pages 32:1–32:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

[FLOS24] Nils Fleischhacker, Kasper Green Larsen, Maciej Obremski, and Mark Simkin. Invertible bloom lookup tables with less memory and randomness. In *32nd Annual European Symposium on Algorithms (ESA 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.

[GM11] Michael T. Goodrich and Michael Mitzenmacher. Invertible bloom lookup tables. In *2011 49th Annual Allerton Conference on Communication, Control, and Computing, Allerton Park & Retreat Center, Monticello, IL, USA, 28-30 September, 2011*, pages 792–799, 2011.

[IKR81] Alon Itai, Alan G. Konheim, and Michael Rodeh. A sparse table implementation of priority queues. In Shimon Even and Oded Kariv, editors, *Automata, Languages and Programming*, pages 417–431, Berlin, Heidelberg, 1981. Springer Berlin Heidelberg.

[JS68] Kumar Jogdeo and Stephen M Samuels. Monotone convergence of binomial probabilities and a generalization of ramanujan's equation. *The Annals of Mathematical Statistics*, 39(4):1191–1195, 1968.

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

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

[PPR09] Anna Pagh, Rasmus Pagh, and Milan Ružić. Linear probing with constant independence. *SIAM Journal on Computing*, 39(3):1107–1120, 2009. See also STOC'07.

[PT10] Mihai Pătraşcu and Mikkel Thorup. On the k-independence required by linear probing and minwise independence. In *Proc. 37th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 715–726, 2010.

[PT12] Mihai Pătraşcu and Mikkel Thorup. The power of simple tabulation hashing. *J. ACM*, 59(3):14:1–14:50, 2012.

[SSS95] Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-Hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995. See also SODA'93.

[Tho13] Mikkel Thorup. Simple tabulation, fast expanders, double tabulation, and high independence. In *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 J. Comput.*, 41(2):293–331, 2012.

[vZHJW07] Anke van Zuylen, Rajneesh Hegde, Kamal Jain, and David P. Williamson. Deterministic pivoting algorithms for constrained ranking and clustering problems. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 405–414, USA, 2007. Society for Industrial and Applied Mathematics.

[vZW09] Anke van Zuylen and David P. Williamson. Deterministic pivoting algorithms for constrained ranking and clustering problems. *Math. Oper. Res.*, 34(3):594–620, 2009. Announced in SODA 2007.

[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.

[WRM16] Di Wang, Satish Rao, and Michael W. Mahoney. Unified acceleration method for packing and covering problems via diameter reduction. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 50:1–50:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 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.

Appendix A

Locally Uniform Hashing

Locally Uniform Hashing

Ioana O. Bercea[1], Lorenzo Beretta[2], Jonas Klausen[2], Jakob Bæk Tejs Houen[2], and Mikkel Thorup[2]

[1]IT University of Copenhagen , {iobe}@itu.dk
[2]University of Copenhagen, {beretta, jokl, jakn, mthorup}@di.ku.dk

Abstract

Hashing is a common technique used in data processing, with a strong impact on the time and resources spent on computation. Hashing also affects the applicability of theoretical results that often assume access to (unrealistic) uniform/fully-random hash functions. In this paper, we are concerned with designing hash functions that are practical and come with strong theoretical guarantees on their performance.

To this end, we present tornado tabulation hashing, which is simple, fast, and exhibits a certain full, local randomness property that provably makes diverse algorithms perform almost as if (abstract) fully-random hashing was used. For example, this includes classic linear probing, the widely used HyperLogLog algorithm of Flajolet, Fusy, Gandouet, Meunier [AOFA'97] for counting distinct elements, and the one-permutation hashing of Li, Owen, and Zhang [NIPS'12] for large-scale machine learning. We also provide a very efficient solution for the classical problem of obtaining fully-random hashing on a fixed (but unknown to the hash function) set of n keys using $O(n)$ space. As a consequence, we get more efficient implementations of the splitting trick of Dietzfelbinger and Rink [ICALP'09] and the succinct space uniform hashing of Pagh and Pagh [SICOMP'08].

Tornado tabulation hashing is based on a simple method to systematically break dependencies in tabulation-based hashing techniques.

1 Introduction

The generic goal of this paper is to create a practical hash function that provably makes important algorithms behave almost as if (unrealistic) fully random hashing was used. By practical, we mean both simple to implement and fast. Speed is important because hashing is a common inner loop for data processing. Suppose for example that we want to sketch a high-volume data stream such as the packets passing a high-end Internet router. If we are too slow, then we cannot handle the stream at all. Speed matters, also within constant factors.

The use of weak hash functions is dangerous, not only in theory but also in practice. A good example is the use of classic linear hashing. Linear hashing is 2-independent and Mitzenmacher and Vadhan [27] have proved that, for some applications, 2-independent hashing performs as well as fully random hashing if the input has enough entropy, and indeed this often works in practice. However, a dense set has only 1 bit of entropy per element, and [36, 30] have shown that with a linear hashing scheme, if the input is a dense set (or more generally, a dense subset of an arithmetic sequence), then linear probing becomes extremely unreliable and the expected probe length[1] increases from constant to $\Omega(\log n)$. This is also a practical problem because dense subsets may occur for many reasons. However, if the system is only tested on random inputs, then we may not discover the problem before deployment.

The issue becomes even bigger with, say, sampling and estimation where we typically just trust our estimates with no knowledge of the true value. We may never find out that our estimates are bad. With 2-independent hashing, we get the right variance, but not exponential concentration. Large errors can happen way too often, yet not often enough to show up in a few tests. This phenomenon is demonstrated on synthetic data in [2] and on real-world data in [1]. All this shows the danger of relying on weak hash functions without theoretical guarantees for all possible inputs, particularly for online systems where we cannot just change the hash function if the input is bad, or in situations with estimates whose quality cannot be verified. One could instead, perhaps, use hash functions based on cryptographic assumptions, but the hash function that we propose here is simple to implement, fast, and comes with strong unconditional guarantees.

In this paper, we introduce *tornado tabulation hashing*. A tornado tabulation hash function $h : \Sigma^c \to \mathcal{R}$ requires $O(c\,|\Sigma|)$ space and can be evaluated in $O(c)$ time using, say, $2c$ lookups in tables with $|\Sigma|$ entries plus some simple AC^0 operations (shifts, bit-wise xor, and assignments). As with other tabulation schemes, this is very fast when Σ is small enough to fit in fast cache, e.g., for 32-bit keys divided into $c = 4$ characters of 8 bits (namely, $|\Sigma| = 2^8$), the speed is similar to that of evaluating a degree-2 polynomial over a Mersenne prime field.

Tornado hashing has the strong property that if we hash a set of $|\Sigma|/2$ keys, then with high probability, the hash values are completely independent. For when we want to handle many more keys, e.g., say $|\Sigma|^3$ (as is often the case when Σ is small), tornado tabulation hashing offers a certain *local uniformity* that provably makes a diverse set of algorithms behave almost as if the hashing was fully random on all the keys. The definition of local uniformity is due to Dahlgaard, Knudsen, Rotenberg, and Thorup [10]. The definition is a bit complicated, but they demonstrate how it applies to the widely used HyperLogLog algorithm of the Flajolet, Fusy, Gandouet, Meunier [18] for counting distinct elements in data streams, and the One-Permutation Hashing of Li, Owen, and Zhang [26] used for fast set similarity. They conclude that the estimates obtained are only a factor

[1]The probe length is defined as the number of contiguous cells probed to answer a query of a linear probing hash table.

$1 + o(1)$ worse than if fully-random hashing was used. Interestingly, [10] proves this in a high-level black-box manner. Loosely speaking, the point is that the algorithm using locally uniform hashing behaves as well as the same algorithm using fully-random hashing on a slightly worse input.

As a new example, we will demonstrate this on linear probing. Knuth's original 1963 analysis of linear probing [25], which started the field of algorithms analysis, showed that with fully-random hashing and load $(1 - \varepsilon)$, the expected probe length is $(1 + 1/\varepsilon^2)/2$. From this, we conclude that tornado tabulation hashing yields expected probe length $(1 + o(1))(1 + 1/\varepsilon^2)/2$, and we get that without having to reconsider Knuth's analysis.

For contrast, consider the work on linear probing with k-independent hashing. Pagh, Pagh, Ružić [29] showed that 5-independence is enough to obtain a bound of $O(1/\varepsilon^{13/6})$ on the expected probe length. This was further improved to $O(1/\varepsilon^2)$ by Pătraşcu and Thorup [31], who achieved the optimal $O(1/\varepsilon^2)$. They matched Knuth's bound modulo some large constants hidden in the O-notation and needed a very different analysis.

In practice, the guarantee that we perform almost like fully-random hashing means that no set of input keys will lead to substantially different performance statistics. Thus, if we tested an online system on an arbitrary set of input keys, then we do not have to worry that future input keys will degrade the performance statistics, not even by a constant factor.

The definition of local uniformity is due to Dahlgaard, Knudsen, Rotenberg, and Thorup [10]. They did not name it as an independent property, but they described it as a property of their new hashing scheme: mixed tabulation hashing. However, the local uniformity of mixed tabulation assumes table size $|\Sigma| \to \infty$, but the speed of tabulation hashing relies on $|\Sigma|$ being small enough to fit in fast cache and all reported experiments use 8-bit characters (see [31, 2, 1, 11]). However, none of the bounds from [10] apply to 8 or even 16-bit characters, e.g., they assume $O(\log |\Sigma|)^c < |\Sigma|$. Our new scheme avoids the exponential dependence on c, and we get explicit error probability bounds that are meaningful, not just in theory, but also for practice with tables in fast cache.

For when we want full randomness on more keys than fit in fast cache, we could, as above, increase $|\Sigma|$ in all $O(c)$ lookup tables. In this paper, we show that it suffices to augment the in-cache tornado hashing with just 2 lookups in tables of size $2n$ to get full randomness on n keys with high probability. This would work perfectly inside a linear space algorithm assuming fully-random hashing, but it also leads to more efficient implementations of the spitting trick of Dietzfelbinger and Rink [13] and the succinct space uniform hashing of Pagh and Pagh [28].

In Section 1.1 we define our hash function, and in Section 1.2 we present our technical results, including the definition of local uniformity. In Section 1.3, we discuss more explicitly how our work compares to mixed tabulation and explain some of our techniques in comparison. In Section 1.4 we discuss several applications. In Section 1.5, we relate our work to previous work on achieving highly independent hash functions. Finally, in Section 1.6 we discuss how tornado tabulation can be employed to improve the so-called "splitting trick" and succinct uniform hashing.

1.1 Tornado tabulation hashing

Simple tabulation hashing. We first introduce our main building block, which is the simple tabulation hash function dating back to at least Zobrist [38] and Wegman and Carter [37]. Throughout the paper, we will consider keys to come from the universe Σ^c and hash values to be in $\mathcal{R} = [2^r]$. More concretely, we interpret a key x as the concatenation of c *characters* $x_1 \dots x_c$

from Σ. We then say that a function $h : \Sigma^c \longrightarrow \mathcal{R}$ is a *simple tabulation* hash function if

$$h(x) = T_1[x_1] \oplus \cdots \oplus T_c[x_c]$$

where, for each $i = 1 \ldots c$, $T_i : \Sigma \longrightarrow \mathcal{R}$ is a fully-random function stored as a table.

We think of c as a small constant, e.g., $c = 4$, for 32-bit keys divided into 8-bit characters, yet we will make the dependence on c explicit. We assume that both keys and hash values fit in a single word and that $|\Sigma| \geqslant 2^8$.

Tornado tabulation hashing. To define a tornado tabulation hash function h, we use several simple tabulation hash functions. A tornado tabulation function has a number d of *derived* characters. Think of d as, say, c or $2c$. It will later determine our error probability bounds. We will always assume $d = O(c)$ so that d characters from Σ can be represented in $O(1)$ words of memory (since a key from Σ^c fits in a single word).

For each $i = 0, \ldots, d$, we let $\widetilde{h}_i : \Sigma^{c+i-1} \longrightarrow \Sigma$ be a simple tabulation hash function. Given a key $x \in \Sigma^c$, we define its *derived key* $\widetilde{h}(x) \in \Sigma^{c+d}$ as $\widetilde{x} = \widetilde{x}_1 \cdots \widetilde{x}_{c+d}$, where

$$\widetilde{x}_i = \begin{cases} x_i & \text{if } i < c \\ x_c \oplus \widetilde{h}_0(\widetilde{x}_1 \cdots \widetilde{x}_{c-1}) & \text{if } i = c \\ \widetilde{h}_{i-c}\,(\widetilde{x}_1 \cdots \widetilde{x}_{i-1}) & \text{if } i > c. \end{cases} \tag{1}$$

We note that each of the d derived characters $\widetilde{x}_{c+1}, \ldots, \widetilde{x}_{c+d}$ is progressively defined by applying a simple tabulation hash function to all its preceding characters in the derived key $\widetilde{x}$. Hence, the name tornado tabulation. The step by which we obtain $\widetilde{x}_c$ corresponds to the twist from [32]. By Observation 1.1. in [32], $x_1 \ldots x_c \mapsto \widetilde{x}_1 \ldots \widetilde{x}_c$ is a permutation, so distinct keys have distinct derived keys. Finally, we have a simple tabulation hash function $\widehat{h} : \Sigma^{c+d} \longrightarrow \mathcal{R}$, that we apply to the derived key. The *tornado tabulation* hash function $h : \Sigma^c \longrightarrow \mathcal{R}$ is then defined as $h(x) = \widehat{h}(\widetilde{x})$.

Implementation. The simplicity of tornado tabulation is apparent from its C implementation below. In the code, we fold tornado's lookup tables together so we can implement them using $c + d$ character tables $\Sigma \to \Sigma^{d+1} \times \mathcal{R}$. Elements of $\Sigma^{d+1} \times \mathcal{R}$ are just represented as w-bits numbers. For memory alignment and ease of implementation, we want w to be a power of two such as 64 or 128.

We now present a C-code implementation of tornado tabulation for 32-bit keys, with $\Sigma = [2^8]$, $c = 4$, $d = 4$, and $\mathcal{R} = [2^{24}]$. Besides the key `x`, the function takes as input an array of $c + d$ tables of size $|\Sigma|$, all filled with independently drawn 64-bit values.

```
INT32 Tornado(INT32 x, INT64[8][256] H) {
        INT32 i; INT64 h=0; INT8 c;
        for (i=0;i<3;i++) {
                c=x;
                x>>=8;
                h^=H[i][c];}
        h^=x;
        for (i=3;i<8;i++) {
                c=h;
                h>>=8;
                h^=H[i][c];}
        return ((INT32) h);}
```

Speed. As we can see in the above implementation, tornado hashing uses $c + d$ lookups and $O(c + d)$ simple AC^0 operations. The speed of tabulation hashing depends on the tables fitting in fast cache which means that Σ should not be too big. In the above code, we used $\Sigma = [2^8]$, as in all previously reported experiments with tabulation hashing. (see [31, 2, 1, 11]).

The speed of tabulation schemes is incomparable to that of polynomial methods using small space but multiplication. Indeed, the ratio between the cost of cache lookups and multiplication depends on the architecture. In line with previous experiments, we found our tornado implementation for 32-bit keys to be as fast as a degree-2 polynomial over a Mersenne prime $(2^{89} - 1)$ field.

We note that our implementation for the random table `H` only needs a pointer to an area filled with "fixed" random bits, and it could conceivably be made much faster if we instead of cache had access to random bits stored in simple read-only memory (ROM or EPROM).

1.2 Theoretical Results

The main aim of our paper is to prove that, with high probability (whp), a tornado tabulation hash function is fully random on some set X of keys. The challenge is to characterize for which kinds of sets we can show such bounds.

Full randomness for fixed keys. We begin with a simpler result that follows directly from our main technical theorem. In this case, the set X of keys is fixed.

Theorem 1. *Let $h : \Sigma^c \to \mathcal{R}$ be a random tornado tabulation hash function with d derived characters. For any fixed $X \subseteq \Sigma^c$, if $|X| \leqslant |\Sigma|/2$, then h is fully random on X with probability at least*

$$1 - 7|X|^3(3/|\Sigma|)^{d+1} - 1/2^{|\Sigma|/2}\,.$$

With $c, d = O(1)$, Theorem 1 gives an $O(|\Sigma|)$ space hash function that can be evaluated in constant time and, with high probability, will be fully random for any fixed set X of at most $|\Sigma|/2$ keys. This is asymptotically tight as we need $|X|$ random hash values to get this randomness.

The random process behind the error probability that we get will be made clear in the next paragraph. We note here that, since $|X| \leqslant |\Sigma|/2$, we have that $7|X|^3(3/|\Sigma|)^{d+1} \leqslant 24(3/|\Sigma|)^{d-2}$. With $|\Sigma| \geqslant 2^8$, the bound is below $1/300$ for $d = 4$, and decreases rapidly for larger d. For $c \geqslant 4$, if we set $d = 2c$, we get an error probability below $1/u$ where $u = |\Sigma|^c$ is the size of the universe. We can get error probability $1/u^\gamma$ for any constant γ with $d = O(c)$, justifying this assumption on d.

Linear independence. The general structure of our results is to identify some error event such that (1) if the event does not occur, then the hash function will be fully random on X, and (2) the error event itself happens with low probability. The error event that we consider in Theorem 1 is inherent to all tabulation-based hashing schemes. Namely, consider some set Y of keys from some universe Σ^b. We say that Y is *linearly independent* if and only if, for every subset $Y' \subseteq Y$, there exists a character position $i \in \{1, \ldots, b\}$ such that some character appears an odd number of times in position i among the keys in Y'. A useful connection between this notion and tabulation-based hashing was shown by Thorup and Zhang [36], who proved that a set of keys is linearly independent if and only if simple tabulation hashing is fully random on these keys:

Lemma 2 (Simple tabulation on linearly independent keys)**.** *Given a set of keys $Y \subseteq \Sigma^b$ and a simple tabulation hash function $h : \Sigma^b \to \mathcal{R}$, the following are equivalent:*

(i) Y is linearly independent

(ii) h is fully random on Y (i.e., $h|_Y$ is distributed uniformly over $\mathcal{R}^Y$).[2]

To prove Theorem 1, we employ Lemma 2 with sets of derived keys. Namely, given a set of keys $X \subseteq \Sigma^c$, we consider the error event that the set $\widetilde{X} = \left\{\widetilde{h}(x) \mid x \in X\right\}$ of its derived keys is linearly dependent. We then show that this happens with probability at most $7|X|^3(3/|\Sigma|)^{d+1} + 1/2^{|\Sigma|/2}$. If this doesn't happen, then the derived keys are linearly independent, and, by Lemma 2, we get that the tornado tabulation hash function $h = \widehat{h} \circ \widetilde{h}$ is fully random on X since it applies the simple tabulation hash function $\widehat{h}$ to the derived keys $\widetilde{X}$. We note that the general idea of creating linearly independent lookups to create fully-random hashing goes back at least to [33]. The point of this paper is to do it in a really efficient way.

Query and selected keys. Our main result, Theorem 5, is a more general version of Theorem 1. Specifically, while Theorem 1 holds for any fixed set of keys, it requires that $|X| \leqslant |\Sigma|/2$. For a fast implementation, we want $|\Sigma|$ to be small enough to fit in fast cache, e.g., $|\Sigma| = 2^8$, but in most applications, we want to hash a set S of keys that is much larger, e.g., $|S| \sim |\Sigma|^3$. Moreover, we might be interested in showing full randomness for subsets X of S that are not known in advance: consider, for instance, the set X of all the keys in S that hash near to $h(q)$ for some fixed key $q \in S$. In this case, Theorem 1 would not help us, since the set X depends on $h(q)$ hence on h.

To model this kind of scenario, we consider a set of *query keys* $Q \subseteq \Sigma^c$ and define a set of *selected keys* $X \subseteq \Sigma^c$. Whether a key x is selected or not depends only on x, its own hash value $h(x)$, and the hash values of the query keys $h|_Q$ (namely, conditioning on $h(x)$ and $h|_Q$ makes $x \in X$ and h independent). In Theorem 5, we will show that, if the selected keys are few enough, then $h|_X$ is fully random with high probability.

Formally, we have a selector function $f : \Sigma^c \times \mathcal{R} \times \mathcal{R}^Q \longrightarrow \{0,1\}$ and we define the set of selected keys as

$$X^{f,h} = \{x \in \Sigma^c \mid f(x, h(x), h|_Q) = 1\}\,.$$

We make the special requirement that f should always select all the query keys $q \in Q$, that is, $f(q,\cdot,\cdot) = 1$ regardless of the two last arguments. We then define

$$\mu^f := \sum_{x \in \Sigma^c} p_x^f \quad \text{with} \quad p_x^f := \max_{\varphi \in \mathcal{R}^Q} \Pr_{r \sim \mathcal{U}(\mathcal{R})}\left[f(x, r, \varphi) = 1\right]\,. \tag{2}$$

Here the maximum is taken among all possible assignments of hash values to query keys $\varphi : Q \to \mathcal{R}$ and r is distributed uniformly over $\mathcal{R}$. Trivially we have that

Observation 3. If $h : \Sigma^c \to \mathcal{R}$ is fully random then $\mathrm{E}[|X^{f,h}|] \leqslant \mu^f$.

When f and h are understood, we may omit these superscripts. It is important that X depends on both f and h while μ only depends on the selector f. We now have the following main technical theorem:

Theorem 4. *Let $h = \widehat{h} \circ \widetilde{h} : \Sigma^c \to \mathcal{R}$ be a random tornado tabulation hash function with d derived characters and f as described above. If $\mu^f \leqslant \Sigma/2$ then the derived selected keys $\widetilde{h}(X^{f,h})$ are linearly dependent with probability at most $\mathsf{DependenceProb}(\mu^f, d, \Sigma)$, where*

$$\mathsf{DependenceProb}(\mu, d, \Sigma) := 7\mu^3(3/|\Sigma|)^{d+1} + 1/2^{|\Sigma|/2}\,.$$

[2] In general, we employ the notation $h|_S$ to denote the function h restricted to the keys in some set S.

Note that Theorem 4 only bounds the probability of the error event. Similarly as in Theorem 1, we would like to then argue that, if the error event does not happen, we could apply Lemma 2 to claim that the final hash values via the simple tabulation function $\widehat{h}$ are fully random. The challenge, however, is the presence of an inherent dependency in how the keys are selected to begin with, namely that $h = \widehat{h} \circ \widetilde{h}$ is already used to select the keys in $X^{f,h}$. In other words, by the time we want to apply $\widehat{h}$ to the derived selected keys $\widetilde{h}(X^{f,h})$, we have already used some information about $\widehat{h}$ in selecting them in $X^{f,h}$.

Local uniformity. Nevertheless, there is a general type of selector functions for which we can employ Theorem 4 in conjunction with Lemma 2 to claim full randomness. Namely, we consider selector functions that partition the bit representation of the final hash values into *s selection bits* and *t free bits* so that $\mathcal{R} = \mathcal{R}_s \times \mathcal{R}_t = [2^s] \times [2^t]$. Given a key $x \in \Sigma^c$, we then denote by $h^{(s)}(x) \in \mathcal{R}_s$ and $h^{(t)}(x) \in \mathcal{R}_t$ the selection and free bits of $h(x)$ respectively. We then say that a selector function f is an *s-selector* if, for all $x \in \Sigma^c$, the output of $f(x, h(x), h|_Q)$ only depends on the selection bits of the hash function, i.e., $f(x, h(x), h|_Q) = f(x, h^{(s)}(x), h^{(s)}|_Q)$.

We now crucially exploit the fact that the output bits of a simple tabulation hash function are completely independent. Formally, we split the simple tabulation $\widehat{h}$ into two independent simple tabulation functions: $\widehat{h}^{(s)}$ producing the selection bits and $\widehat{h}^{(t)}$ producing the free bits. We then apply Theorem 4 to $h^{(s)} = \widehat{h}^{(s)} \circ \widetilde{h}$ to conclude that the set of selected derived keys $\widetilde{h}(X^{f,h^{(s)}})$ is linearly independent with high probability. Assuming this, we then apply Lemma 2 to conclude that $\widehat{h}^{(t)}$ is fully random on $\widetilde{h}(X^{f,\widehat{h}^{(s)}})$, hence that $h^{(t)} = \widehat{h}^{(t)} \circ \widetilde{h}$ is fully random on $X^{f,h^{(s)}}$.

Theorem 5. *Let $h : \Sigma^c \to \mathcal{R}$ be a tornado tabulation hash function with d derived characters and f be an s-selector as described above. If $\mu^f \leqslant \Sigma/2$, then $h^{(t)}$ is fully random on $X^{f,h^{(s)}}$ with probability at least*

$$1 - \mathsf{DependenceProb}(\mu^f, d, \Sigma) \ .$$

While the concept of an s-selector function might seem a bit cryptic, we note that it intuitively captures the notion of locality that linear probing and other applications depend on. Namely, the effect of the (high order[3]) selector bits is to specify a dyadic interval[4] of a given length such that all the keys with hash values falling in that interval are possibly selected (with this selection further depending, perhaps, on the query keys Q, or on other specific selector bits, leading to more refined dyadic intervals). Theorem 5 then says that the (low order) free bits of these selected keys will be fully-random with high probability. In other words, the distribution inside such a neighborhood is indistinguishable from what we would witness if we had used a fully-random hash function.

As mentioned earlier, the concept of local uniformity stems from [10], except that they did not consider query keys. Also, they didn't name the concept. They demonstrated its power in different streaming algorithms. For those applications, it is important that *the selection bits are not known to the algorithm.* They are only set in the analysis based on the concrete input to demonstrate good performance on this input. The problem in [10] is that their error probability bounds only apply when the alphabet is so large that the tables do not fit in fast cache. We will describe this issue closer in Section 1.3. In Section 1.4 we will sketch the use of local uniformity on linear probing where the locality is relative to a query key.

[3]Thinking about selector bits as higher order bits helps our intuition. However, they do not have to be higher-order bits necessarily. More generally, any representation of $\mathcal{R}$ as a product $\mathcal{R}_s \times \mathcal{R}_t$ would do the job.

[4]Recall that a dyadic interval is an interval of the form $[j2^i, (j+1)2^i)$, where i, j are integers.

Upper tail Chernoff bounds. Theorem 5 is only concerned with the distribution of the free bits of the selected keys, but to employ it in our applications, we often require that the number of selected keys is not much larger than μ^f with high probability (see Sections 1.4 and 1.6). We show that this size can be bounded from above with the usual Chernoff bound when the set of derived selected keys is linearly independent.

Lemma 6. *Let $h = \widehat{h} \circ \widetilde{h} : \Sigma^c \to \mathcal{R}$ be a random tornado tabulation hash function with d derived characters, query keys Q and selector function f. Let $\mathcal{I}_{X^{f,h}}$ denote the event that the set of derived selected key $\widetilde{h}(X^{f,h})$ is linearly independent. Then, for any $\delta > 0$, the set $X^{f,h}$ of selected keys satisfies the following:*

$$\Pr\left[\left|X^{f,h}\right| \geqslant (1+\delta)\cdot\mu^f \wedge \mathcal{I}_{X^{f,h}}\right] \leqslant \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^{\mu^f} .$$

For a nice direct application, consider hash tables with chaining, or throwing n keys into n bins using tornado hashing. We can select a given bin, or the bin of a given query key. In either case we have $\mu^f = 1$ and then Lemma 6 together with Theorem 4 says that the probability of getting k keys is bounded by

$$e^{k-1}/k^k + 7(3/|\Sigma|)^{d+1} + 1/2^{|\Sigma|/2}. \tag{3}$$

If k is not too large, the first term dominates.

Lower bound. Finally, we show that our upper bound for the error probability in Theorem 4 is tight within a constant factor.

Theorem 7. *Let $h = \widehat{h} \circ \widetilde{h} : \Sigma^c \to \mathcal{R}$ be a random tornado tabulation hash function with d derived characters. There exists a selector function f with $\mu^f \leqslant \Sigma/2$ such that the derived selected keys $\widetilde{h}(X^{f,h})$ are linearly dependent with probability at least $\Omega((3/|\Sigma|)^{d-2})$.*

Full randomness for larger sets of selected keys. As mentioned earlier, for a fast implementation of tabulation hashing, we pick the alphabet Σ small enough for the tables to fit in fast cache. A common choice is 8-bit characters. However, we only get full randomness for (selected) sets of (expected) size at most $|\Sigma|/2$ (c.f. Theorems 5 and 1).

To handle larger sets, we prove that it suffices to only increase the alphabet of the last two derived characters; meaning that only two lookups have to use larger tables. This is close to best possible in that we trivially need at least one lookup in a table at least as big as the set we want full randomness over. More precisely, if we use alphabet Ψ for the last two derived characters, then our size bound increases to $|\Psi|/2$. The error probability bound of Theorems 5 becomes

$$14(\mu^f)^3(3/|\Psi|)^2(3/|\Sigma|)^{d-1} + 1/2^{|\Sigma|/2} .$$

With the above mix of alphabets, we have tornado hashing running in fast cache except for the last two lookups that could dominate the overall cost, both in time and space. Because they dominate, we will consider a slight variant, where we do not store derived characters in any of the two large tables. Essentially this means that we change the definition of the last derived character from $\widetilde{x}_{c+d} = \widetilde{h}_d(\widetilde{x}_1 \cdots \widetilde{x}_{c+d-1})$ to $\widetilde{x}_{c+d} = \widetilde{h}_d(\widetilde{x}_1 \cdots \widetilde{x}_{c+d-2})$. This is going to cost us a factor two in the error probability, but in our implementation, we will now have $c+d-2$ lookups in tables $\Sigma \to \Sigma^{d-1} \times \Psi^2 \times R$ (where the values are represented as a single w-bit numbers), and 2 lookups in tables $\Psi \to R$. We shall refer to this scheme as *tornado-mix*. Corresponding to Theorem 5, we get

Theorem 8. *Let $h = \widehat{h} \circ \widetilde{h} : \Sigma^c \to \mathcal{R}$ be a random tornado-mix tabulation hash function with d derived characters, the last two from Ψ, and an s-selector function f. If $\mu = \mu^f \leqslant |\Psi|/2$ then $h^{(t)}$ is fully random on $X^{f,h}$ with probability at least*

$$1 - 14\mu^3(3/|\Psi|)^2(3/|\Sigma|)^{d-1} - 1/2^{|\Sigma|/2}\ .$$

An interesting case is when we want linear space uniform hashing for a fixed set X, in which case $\mu = |X|$ above. This leads to a much better implementation of the uniform hashing fo Pagh and Pagh [28]. The main part of their paper was to show a linear space implementation, and this would suffice for all but the most succinct space algorithms. They used highly independent hashing [33, 35] as a subroutine, but this subroutine alone is orders of magnitude slower than our simple implementation (see, e.g., [1]). They combined this with a general reduction from succinct space to linear space, for which we now have a really efficient construction.

1.3 Techniques and relation to mixed tabulation

In spirit, our results are very related to the results on mixed tabulation [10]. For now, we only consider the case of a single alphabet Σ. Indeed, tornado and mixed tabulation are very similar to implement. Both deal with c-character keys from some alphabet Σ, produce a derived key with $c+d$ characters, and then apply a top simple tabulation to the resulting derived keys. Both schemes can be implemented with $c + d$ lookups. The difference is in how the two schemes compute the derived keys. For ease of presentation, let $\widetilde{h}_i : \Sigma^* \to \Sigma$, that is, $\widetilde{h}_i$ adjusts to the number of characters in the input. Now for mixed tabulation, we define the derived key $\widetilde{x}_1 \cdots \widetilde{x}_{c+d}$ by

$$\widetilde{x}_i = \begin{cases} x_i & \text{if } i \leqslant c \\ \widetilde{h}_{i-c}\left(\widetilde{x}_1 \ldots \widetilde{x}_c\right) & \text{if } i > c. \end{cases}$$

The analysis from [10] did not consider query keys, but ignoring this issue, their analysis works in the limit $|\Sigma| \to \infty$. For example, the analysis from [10] requires that [5]

$$(\log|\Sigma|)^c \leqslant |\Sigma|/2.$$

This is true for $c = O(1)$ and $|\Sigma| \to \infty$, but simply false for realistic parameters. Assuming the above condition to be satisfied, if we consider scenarios with non-constant c and d, the error probability from [10] becomes

$$(O(cd)^c/|\Sigma|)^{\lfloor d/2 \rfloor - 1} + 1/2^{\Omega(|\Sigma|)}.$$

Now, even if we replace $O(cd)$ with cd, the error probability is not below 1 even with 16-bit characters and $c = 4$. In contrast, practical tabulation schemes normally use 8-bit characters for efficiency, and our explicit bound of $7\mu^3(3/|\Sigma|)^{d+1} + 1/2^{|\Sigma|/2}$ from Theorem 4 works fine even in this case, implying that our theory actually applies to practice.

The reason that mixed tabulation has the problematic exponential dependency on c is that for a set of linearly dependent keys, it uses a clever encoding of each of the c characters in some of the keys. With tornado, the only encoding we use is that if we have a zero set of keys, then each key is the xor of the other keys in the zero set, and this is independent of c. [6]

[5] While using their Lemma 3, if t goes up to s in case D.

[6] A set is linearly dependent if it contains a subset that is a zero set. See Section 2 for the precise definition.

To describe the advantage of tornado tabulation over mixed tabulation, it is easier to first compare with what we call *simple tornado* where the derived key $\widetilde{x}_1 \cdots \widetilde{x}_{c+d}$ is defined by

$$\widetilde{x}_i = \begin{cases} x_i & \text{if } i \leqslant c \\ \widetilde{h}_{i-c}\left(\widetilde{x}_1 \ldots \widetilde{x}_{i-1}\right) & \text{if } i > c. \end{cases}$$

The implementation difference is that with simple tornado, each derived character uses all preceding characters while mixed tabulation uses only the original characters. This difference gives tornado a big advantage when it comes to breaking up linear dependence in the input keys. Recall that a set $Y \subset \Sigma^{c+d}$ of derived keys is linearly independent if and only if, for every subset $Y' \subseteq Y$, there exists a character position $i \in \{1, \ldots, c+d\}$ such that some character appears an odd number of times in position i among the keys in Y'. Intuitively, the strategy is to argue that whatever linear dependence exists in the input key set to begin with (essentially, in the first c characters of the derived keys) will, whp, be broken down by their d derived characters (and hence, disappear in the derived keys). In this context, the way we compute the derived characters becomes crucial: in mixed tabulation, each derived character is computed independently of the other derived characters. Thus, whether a derived character breaks a dependency or not is independent of what other derived characters do. In contrast, we make the derived characters in tornado tabulation depend on all previously computed derived characters such that, if we know that some derived character does not break a dependency, then we also know that none of its previously computed derived characters have broken it either. Or, in other words, each successive tornado-derived character benefits from the dependencies already broken by previously computed derived characters, i.e., the benefits compound each time we compute a new derived character.

This structural dependence between tornado-derived characters turns out to be very powerful in breaking linear dependencies among the input keys and indeed, leads to a much cleaner analysis. The most important benefit, however, is that tornado tabulation has a much lower error probability. For simple tornado, we get an error probability of

$$7(\mu^f)^3(3/|\Sigma|)^d + 1/2^{|\Sigma|/2}\ ,$$

which essentially gains a factor $(3/|\Sigma|)$ for each derived character. It turns out that we gain an extra factor $(3/|\Sigma|)$ if we twist the last original character as we did in the original tornado definition (1), and then we get the bound from Theorem 5, the point being that this twisting does not increase the number of lookups.

One might wonder if twisting more characters would help further, that is, setting $\widetilde{x}_i = x_i \oplus \widetilde{h}_i(\widetilde{x}_1 \cdots x_{i-1})$ for $i = 2, \ldots, c$, but it doesn't. The point is that the bad key set from our lower bound in Theorem 7 is of the form $[0]^{c-2} \times [2] \times A$ for some $A \subseteq \Sigma$, and then it is only the last character where twisting makes a difference.

As a last point, recall tornado-mix which was designed to deal with larger sets. There it only costs us a factor 2 when we let the last derived character $\widetilde{x}_{c+d}$ depend only on $\widetilde{x}_1 \ldots \widetilde{x}_{c+d-2}$ and not on $\widetilde{x}_{c+d-1}$. This is essentially like switching to mixed tabulation on the last two derived keys, hence the name *tornado-mix*. This only works for the last two derived characters that can play a symmetric role.

The exponential dependence on c. We note that having an exponential dependence on c is symptomatic for almost all prior work on tabulation hashing, starting from the original work in [31]. Above, we discussed how tornado tabulation hashing avoided such exponential dependence on

c in the error probability from mixed tabulation. The dependence on c was particularly destructive for mixed tabulation because it pushed the error probability above 1 for the relevant parameters.

Concentration bounds. Tornado hashing inherits the strongest concentration bounds known for mixed tabulation [22]. The reason is that they only require one derived character and tornado tabulation can be seen as applying mixed tabulation with one character to a derived tornado key with one less character. Unlike our Lemma 6, which essentially only applies for expected values $\mu \leqslant |\Sigma|/2$, the concentration bounds we get from [22] work for unbounded μ and for both the upper and lower tail. They fall exponentially in μ/K_c where K_c is exponential in c, but this still yields strong bounds when μ is large. Inheriting the strongest concentration bound for mixed tabulation implies that tornado tabulation can replace mixed tabulation in all the applications from [10] while improving on the issue of local uniformity.

1.4 The power of locally uniform hashing

We now describe, on a high level, how the notion of locally uniform hashing captures a type of randomness that is sufficient for many applications to work almost as if they employed full randomness. For the discussion, we will assume the parameters from Theorem 8 with tornado-mix. The main observation is that, for many algorithms, the performance measure (i.e., its distribution) depends, whp, only on the behavior of keys that hash inside a local neighborhood defined via some selected bits of the hash value (the selection may depend both on the algorithm and on the concrete input since the selection is only used for analysis). Moreover, the set of keys $X^{f,h}$ that land in that selected neighborhood has expected size $\leqslant |\Psi|/2$. For any such neighborhood, our results imply that the keys in $X^{f,h}$ have fully random free bits whp.

To understand the role of the free bits, it is helpful to think of hashing a key x as a two-stage process: the selector bits of $h(x)$ tell us whether x hashes in the desired neighborhood or not, while the remaining free bits of $h(x)$ determine how x hashes once it is *inside* the neighborhood. This suggests a general coupling, where we let both fully-random hashing and tornado tabulation hashing first choose the select bits and then the free bits. Since both are fully random on the free bits (for us with high probability), the only difference is in the selection, but here concentration bounds imply that we select almost the same number of keys as fully-random hashing.

In [10], this approach was demonstrated for the HyperLogLog algorithm of Flajolet, Fusy, Gandouet, Meunier [18] for counting distinct elements in data streams, and the One-Permutation Hashing of Li, Owen, and Zhang [26] used for fast set similarity in large scale machine learning. In [10] they implemented the local uniformity with mixed tabulation, but with tornado hashing we get a realistic implementation. This also includes later application of mixed tabulation, e.g., the dynamic load balancing from [3]. Below, as a new example, we illustrate how local uniformity makes classic linear probing perform almost as if fully random hashing was used.

We briefly recall the basic version of linear probing. We employ an array T of length m and hash keys to array entries using a tornado tabulation hash function $h : \Sigma^c \to [m]$. Thus, in Theorem 8, we have $\mathcal{R} = [m] = [2^r]$. Upon insertion of a key x, we first check if entry $T[h(q)]$ is empty. If it is, we insert the key in $T[h(q)]$. Otherwise, we scan positions in T sequentially starting with $T[h(q) + 1]$ until we find an empty position and store x in this next available free entry. To search for x, we inspect array entries sequentially starting with $T[h(q)]$ until we find x or we find an empty array entry, concluding that the key is not in the array. The general concept that dominates the performance of linear probing is the *run* of q, which is defined as the longest

interval $I_q \subseteq [m]$ of full (consecutive) array entries that $h(q)$ is part of. The run affects the probe length (the length of the interval from $h(q)$ till the end of I_q) and other monotone measures such as insertion and deletion time (i.e., monotonically increasing in the number of keys). Thus, any corresponding analysis depends only on the set X_q of keys that fall in the interval I_q, and it is there that we are interested in showing full randomness.

The first step is to argue that the behavior of I_q is local in nature, in that it is affected only by the keys that hash in a fixed length interval around $h(q)$. To that end, we show that, whp, the length of the run is no bigger than a specific $\Delta = 2^\ell$. This Δ implies locality: consider the interval $J_q \subseteq [m]$, centered at $h(q)$, which extends Δ entries in each direction, i.e., $J_q = \{j \in [m] \mid |j - h(q)| \leqslant \Delta\}$. Then, whp, any run that starts outside J_q is guaranteed to end by the time we start the run of q. In other words, whp, the behavior of I_q depends only on the set X_q of keys that hash inside the fixed length neighborhood J_q (and specifically where in J_q the keys hash). A similar argument can be made for other variants of linear probing, such as linear probing with tombstones [6], where the run/neighborhood must also take into account the tombstones that have been inserted.

At this point, it is worthwhile pointing out that the set X_q is no longer a fixed set of keys but rather a random variable that depends on the realization of $h(q)$. We cannot just apply Theorem 1. Nevertheless, in the second step, we argue that X_q can be captured by our notion of selector functions. For this, we cover J_q with three dyadic intervals, each of length Δ: one including $h(q)$ and the corresponding ones to the left and to the right. Since each interval is dyadic, it is determined only by the leftmost $r - \ell$ bits of the hash value: for example, an element $x \in \Sigma^c$ hashes into the same dyadic interval as q iff the leftmost $r - \ell$ bits of $h(x)$ match those of $h(q)$. We can then design a selector function that returns a 1 if and only if x is in the input set and the leftmost $r - \ell$ bits of $h(x)$ (its selector bits) hash it into any of the three specific dyadic intervals we care about. Such a selector function is guaranteed to select all the keys in X_q. By setting Δ appropriately, we get that the expected size of the selected set is at most $\Sigma/2$, and can thus apply Theorem 5. We get that inside the intervals, keys from $X^{f,h}$ hash (based on their free bits) in a fully random fashion.

Finally, what is left to argue is that the number of keys hashing inside each of the three dyadic intervals is not much bigger than what we would get if we used a fully-random hash function for the entire set (one in which the selector bits are also fully random). For this, we employ Lemma 6, and can conclude that we perform almost as well with fully-random hashing. In particular, from Knuth's [25] analysis of linear probing with fully-random hashing, we conclude that with load $(1 - \varepsilon)$, the expected probe length with tornado hashing is $(1 + o(1))(1 + 1/\varepsilon^2)/2$.

The above type of analysis could be applied to other variants of linear probing, e.g., including lazy deletions and tombstones as in the recent work of Bender, Kuszmaul, and Kuszmaul [6]. We would need to argue that the run, including tombstones, remains within the selected neighborhood and that we have concentration both on live keys and tombstones. However, since the analysis from [6] is already based on simple hash functions, we would not get new bounds from knowing that tornado hashing performs almost as well as fully-random hashing.

1.5 Relation to high independence

The k-independence approach to hashing [37] is to construct hash functions that map every set of k keys independently and uniformly at random. This is much stronger than getting fully random hashing for a given set and, not surprisingly, the results for highly independent hashing [9, 33, 35] are weaker. The strongest high independence result from [9] says that we get $|\Sigma|^{1-\varepsilon}$-independent hashing in $O((c/\varepsilon)\log(c/\varepsilon))$ time, but the independence is much less than our $|\Sigma|/2$. Experiments

from [1] found that even the simpler non-recursive highly independent hashing from [35] was more than an order of magnitude slower than mixed tabulation which is similar to our tornado tabulation.

We also note that the whole idea of using derived characters to get linear independence between keys came from attempts to get higher k-independence Indeed, [15, 23, 36] derive their characters deterministically, guaranteeing that we get k-independence. The construction for $k = 5$ is efficient but for larger k, the best deterministic construction is that from [36] using $d = (k-1)(c-1)+1$ derived characters, which for larger k is much worse than the randomized constructions.

1.6 Relation to the splitting trick and to succinct uniform hashing

We will now describe how tornado-mix provides a much more efficient implementation of the succinct uniform hashing by Pagh and Pagh [28] and the splitting trick of Dietzfelbinger and Rink [13]. This further illustrates how our work provides a component of wide applicability within hashing.

Succinct uniform hashing. The main contribution in [28] is to obtain uniform hashing in linear space. The succinct construction is then obtained through a general reduction from the linear-space case. They achieved uniform hashing in linear space using the highly independent hashing of Siegel [33] as a subroutine in combination with other ideas. Now, thanks to Theorem 8, we know that tornado-mix offers an extremely simple and efficient alternative to implement that. Indeed, if n is the size of the set we want linear space uniform hashing on, then setting $|\Psi|$ to the power of two just above $2n$ is sufficient to ensure the degree of independence that we need. Moreover, we can set $|\Sigma| \sim \sqrt{\Psi}$ and c so that we obtain (i) $c|\Sigma| = o(|\Psi|)$ and (ii) setting $d = 2c+1$ the probability bound in Theorem 8 becomes $1 - O(1/u)$ where u is the size of the universe. The two previous conditions ensure respectively that tornado-mix uses linear space and that it is, whp, fully random.

The splitting trick. We now explain how our tornado-mix tabulation hashing provides a very simple and efficient implementation of the splitting trick of Dietzfelbinger and Rink [13] which is a popular method for simulating full-randomness in the context of data structures, most notably various dictionary constructions [14, 16, 19, 5]. This is an especially relevant application because it usually requires hashing keys into a range of size $\Theta(n)$ and, in this context, employing the uniform hashing of Pagh and Pagh [28] would be too costly, i.e., (compared to the size of the overall data structure). This trick was also used to obtain a simpler and exponentially faster implementation of succinct uniform hashing [28]. We note that the splitting idea had also been used earlier works of Dietzfelbinger and Meyer auf der Heide [12] and Dietzfelbinger and Woelfel [15].

The idea is to first split the input set S of size n into $n^{1-\delta}$ subsets $S_1, \ldots, S_m$, for some $\delta \in (0,1)$. The splitting is done through a hash function $h_1 : \Sigma^c \to [n^{1-\delta}]$ such that S_i is defined as all the keys in S that hash to the same value, i.e., $S_i = \{x \in S \mid h_1(x) = i\}$. In many applications, we want the splitting to be balanced, that is, we need a joint upper bound s on all set sizes with s close to the expected size n^{δ}. On top of this, we need a shared hash function $h_2 : \Sigma^c \to \mathcal{R}$ which with high probability is fully random on each set S_i and we ensure this with a hash function that w.h.p. is fully random on any set of size at most s. The problem is then solved separately for each subset (e.g., building $n^{1-\delta}$ dictionaries, each responsible for just one subset S_i).

The above splitting trick can be easily done with tornado-mix hashing from Theorem 8. The dominant cost for larger s is two lookups in tables of size at most $4s$. We let h_1 be the select bits (with the strong concentration from Lemma 6) and h_2 as the remaining free bits. Getting both for the price of one is nice, but the most important thing is that we get an efficient implementation of the shared hash function h_2. Specifically, we compare this with how h_1 and h_2 were implemented

in [13] to get uniform hashing. For the splitter h_1, instead of using Siegel's [33] highly independent hashing, [13, §3.1] uses that if h_1 has sufficiently high, but constant, independence, then it offers good concentration (though not as good as ours from Lemma 6).

The bigger issue is in the implementation of the shared h_2 from [13, §3.2]. For this, they first employ a hash function $f : [s] \to [s^{1+\varepsilon}]$ such that, whp, f has at most $k = O(1)$ keys that get the same hash value. For small ε and k, this forces a high independence of f. Next, for every $i \in [s^{1+\varepsilon}]$, they use a k-independent hash function $g_i : \Sigma^c \to \mathcal{R}$, and finally, h_2 is implemented as $g_{f(x)}(x)$. The space to implement h_2 is dominated by the $O(s^{1+\varepsilon})$ space to store the $s^{1+\varepsilon}$ k-independent hash functions g_i, and time-wise, we have to run several sufficiently independent hash functions. This should be compared with our tornado-mix that only uses $O(s)$ space and runs in time corresponding to a few multiplications (in tests with 32-bit keys).

1.7 Paper Organization

The remainder of our paper is structured as follows. In Section 2, we introduce some notation, a more general notion of generalized keys and Lemma 9 (the corresponding version of Lemma 2 for them). Our main technical result, Theorem 4, is proved in three steps: we first define obstructions, the main combinatorial objects we study, in Section 3. In Section 4 we present a simplified analysis of Theorem 4 that achieves a weaker error probability. We then show in Section 5 a tighter analysis that finally achieves the desired bound. The Chernoff bounds for the upper tail are proved in Section 6. The details for the linear probing analysis can be found in Section 7. Finally, the lower bound from Theorem 7 is proved in Section 8.

2 Preliminaries

Notation. We use the notation $n!! = n(n-2)\cdots$. More precisely, $n!! = 1$ for $n \in \{0, 1\}$, while $n!! = n(n-2)!!$ for $n > 1$. For odd n, this is exactly the number of perfect matchings of $n+1$ nodes. We use the notations

$$n^{\underline{k}} = n(n-1)\cdots(n+1-k) = n!/(n-k)!$$
$$n^{\underline{\underline{k}}} = n(n-2)\cdots(n+2(1-k)) = n!!/(n-2k)!$$

Here $n^{\underline{\underline{k}}}$ appears to be non-standard, though it will be very useful in this paper in connection with something we will call greedy matchings. We note that

$$n^{\underline{\underline{k}}} \leqslant n^{\underline{k}} \leqslant n^k.$$

Position characters and generalized keys. We employ a simple generalization of keys going back to Pătraşcu and Thorup [31]. Namely, a *position character* is an element of $\{1 \dots b\} \times \Sigma$, e.g., where $b = c$ or $c + d$. Under this definition a key $x \in \Sigma^b$ can be viewed as a set of b position characters $(1, x_1) \dots (b, x_b)$, and, in general, we consider *generalized keys* that may be arbitrary subsets of $\{1 \dots b\} \times \Sigma$. A natural example of a generalized key is the symmetric difference $x \triangle y$ of two (regular) keys. We then have that

$$h(x) = h(y) \iff h(x \triangle y) = 0.$$

These symmetric differences will play an important role in our constructions, and we shall refer to $x \triangle y$ as a *diff-key*. A diff-key is thus a generalized key where we have zero or two characters in each position. For a generalized key x, we can then define

$$x[i] = \{(i, a) \in x\}, \quad x[< i] = \{(j, a) \in x \mid j < i\} \quad \text{and} \quad x[\leqslant i] = \{(j, a) \in x \mid j \leqslant i\}.$$

For example, if we have a regular key $x = x_1, \ldots, x_c$, then $x[i] = x_i$ and $x[\leqslant i] = x_1 \ldots, x_i$. Also note that $(x \triangle y)[\leqslant i] = x[\leqslant i] \triangle y[\leqslant i]$. We shall also apply this indexing to sets X of generalized keys by applying it to each key individually, e.g.,

$$X[< i] = \{x[< i] \mid x \in X\}.$$

Generalized keys and linear independence. A generalized key x can also be interpreted as a $(|\Sigma| \cdot b)$-dimensional vector over $\mathbb{F}_2$, where the only entries set to 1 are those indexed by position characters in x. The generalized key domain is denoted by $\mathbb{F}_2^{\{1\ldots b\}\times\Sigma}$. Now, if we have a simple tabulation hash function $h : \Sigma^b \to \mathcal{R}$ using character tables $T_1, \ldots, T_b$, then h can be lifted to hash any generalized key $x \in \mathbb{F}_2^{\{1\ldots b\}\times\Sigma}$ by

$$h(x) = \bigoplus_{(i,a)\in x} T_i[a].$$

Thus h provides a mapping $\mathbb{F}_2^{\{1\ldots b\}\times\Sigma}$ to $\mathcal{R}$.

As for (regular) keys, for a set Y of generalized keys, we can then define $\triangle Y$ to be the set of position characters that appear an odd number of times across the keys in Y. We then say that Y is a *zero-set* if $\triangle Y = \varnothing$. We also say that Y is *linearly dependent* if it contains a subset which is a zero-set, and *linearly independent* otherwise. It is apparent then that a set of generalized keys is linearly independent if and only if the set of their vector representations is linearly independent. Indeed, the proof of Thorup and Zhang [36] for Lemma 9 works quite directly in this generality. Thus we have

Lemma 9 (Simple tabulation on linearly independent generalised keys)**.** *Given a set of generalized keys $Y \subseteq \left(\mathbb{F}_2^{\{1\ldots c\}\times\Sigma}\right)$ and a simple tabulation hash function $h : \Sigma^c \to \mathcal{R}$, the following are equivalent:*

(i) Y is linearly independent

(ii) h is fully random on Y (i.e., $h|_Y$ is distributed uniformly over $\mathcal{R}^Y$).

3 Obstructions with simple tornado tabulation

We prove Theorem 4 by first considering a simpler version of tornado tabulation hashing, which we call *simple tornado hashing*, where we do not change the last character of the (original) key. Formally, for a key $x = x_1 \cdots x_c$, its corresponding derived key $\widetilde{x} = \widetilde{x}_1 \ldots \widetilde{x}_{c+d}$ is computed as

$$\widetilde{x}_i = \begin{cases} x_i & \text{if } i = 1, \ldots, c \\ \widetilde{h}_{i-c}\left(\widetilde{x}_1 \ldots \widetilde{x}_{i-1}\right) & \text{otherwise.} \end{cases}$$

Note that, in the original tornado hashing, we had $\widetilde{x}_c = x_c \oplus \widetilde{h}_0(\widetilde{x}_1 \dots \widetilde{x}_{c-1})$. Removing this extra step is thus equivalent to fixing $\widetilde{h}_0(\cdot) = 0$. While this step comes at almost no cost in the code, it allows us to gain a factor of $3/|\Sigma|$ in the overall error probability. See Section 5.3 for details. For the simple tornado hashing, we will prove a slightly weaker probability bound.

For ease of notation, for every key x, we use $\widetilde{x}$ to denote the corresponding derived key $\widetilde{h}(x)$; and likewise for any set of keys. We also define $X = X^{f,h}$ to be the set of selected keys and $\widetilde{X} = \widetilde{h}(X)$. We want to argue that the derived selected keys $\widetilde{X}$ are linearly independent with high probability. To prove this, we assume that $\widetilde{X}$ is linearly dependent and hence, contains some zero-set $\widetilde{Z}$. From $\widetilde{Z}$ we construct a certain type of "obstruction" that we show is unlikely to occur.

3.1 Levels and matchings

We first define some necessary concepts. We use the notion of *level* i to refer to position $c + i$ in the derived keys. Let $M \subseteq \binom{|\Sigma|^c}{2}$ be a (partial) matching on the keys Σ^c.[7] We say that M is an *i-matching* if for all $\{x, y\} \in M$, it holds that $\widetilde{x}[c + i] = \widetilde{y}[c + i]$, namely if every pair of keys in M matches on level i. Our obstruction will, among other things, contain an i-matching M_i for each level i.

Recall that a diff-key $x \triangle y$ is the symmetric difference of two keys x and y in terms of their position characters. We then say that M is an *i-zero, i-dependent, or i-independent* matching if

$$\text{DiffKeys}(M, i) = \{(\widetilde{x} \triangle\, \widetilde{y})[\leqslant c + i] \mid \{x, y\} \in M\}$$

is a zero-set, linearly dependent, or linearly independent, respectively. In other words, for each pair $\{x, y\}$ in the matching, we consider the diff-key corresponding to the first $c + i$ characters of their derived keys. We then ask if this set of diff-keys now forms a zero-set or contains one as a subset, by looking at their (collective) symmetric difference. We employ this notion to derive the probability that the function $\widetilde{h}$ satisfies a certain matching as such:

Lemma 10. *Let M be a partial matching on Σ^c. Conditioning on M being $(i-1)$-independent, M is an i-matching with probability $1/|\Sigma|^{|M|}$.*

Proof. First, we notice that the event "M is $(i-1)$-independent" only depends on $\widetilde{h}_j$ for $j < i$. Then, by Lemma 9, when we apply the simple tabulation hash function $\widetilde{h}_i : \Sigma^{c+i} \to \Sigma$ to the linearly independent generalized key set $\text{DiffKeys}(M, i-1)$, the resulting hash values $\widetilde{h}(z)[c + i]$ for $z \in \text{DiffKeys}(M, i-1)$ are independent and uniform over Σ. Hence, so are the resulting derived characters $\widetilde{h}(z)[c + i]$ for $z \in \text{DiffKeys}(M, i)$. The probability that they are all 0 is therefore $1/|\Sigma|^{|M|}$. □

Similarly, as for matchings, we say that a set of keys $Z \subseteq \Sigma^c$ is *i-zero, i-dependent, or i-independent* if $\widetilde{Z}[\leqslant c + i]$ is a zero-set, linearly dependent, or linearly independent, respectively. We note the following relations:

Observation 11. Let M be a partial matching on Σ^c and $Z = \bigcup M$. Then M is an i-zero matching iff Z is an i-zero set. Furthermore, if M is i-dependent then Z is also i-dependent (but not vice versa).

[7] Here, we mean the graph-theoretic definition of a matching as a set of edges with disjoint endpoints. In our case, the vertices of the graph are keys in Σ^c, and the edges of the matching are represented as $\{x, y\} \in M$.

We will also make use of the following observation repeatedly:

Observation 12. If Z is an i-zero set, then there is a perfect j-matching on Z for every level $j \leqslant i$.

3.2 Constructing an obstruction

In this section, we show that whenever the set of selected derived keys $\widetilde{X}$ is linearly dependent, it gives rise to a certain *obstruction*. We now show how to construct such an obstruction.

Since $\widetilde{X}$ is linearly dependent, there must be some d-zero set $Z \subseteq X$. We are going to pick a d-zero set that (1) minimizes the number of elements contained that are not in the query set Q and, subject to (1), (2) minimizes the number of elements from Q contained. In particular, Z is contained in Q if $\widetilde{Q}$ is not linearly independent.

If Z is not contained in Q, we let x^* be any element from $Z \backslash Q$. Then $Q \cup Z \backslash \{x^*\}$ must be linearly independent since any strict subset Z' would contradict (1). If Z is contained in Q, then we let x^* be an arbitrary element of Z.

The top two levels. By Observation 12, we have a perfect d-matching M_d^* and a perfect $(d-1)$-matching M_{d-1}^* on Z (we also have perfect matchings on other levels, but they will be treated later). These two perfect matchings partition Z into alternating cycles.

We will now traverse these cycles in an arbitrary order except that we traverse the cycle containing x^* last. For all but the last cycle, we start in an arbitrary vertex and its cycle starting with the edge from M_{d-1}^*. When we get to the last cycle, we start at the M_d^* neighbor of x^* and follow the cycle from the M_{d-1}^* neighbor of this key. This ensures that x^* will be the very last vertex visited. The result is a traversal sequence $x_1, \ldots, x_{|Z|}$ of the vertices in Z ending in $x_{|Z|} = x^*$. Note that M_{d-1}^* contains the pairs $\{x_1, x_2\}, \{x_3, x_4\}, \ldots$. For M_d^* it is a bit more complicated, since its pairs may be used to complete a cycle (and hence are not visible in the traversal).

We now define $W = \{x_1 \ldots x_w\}$ to be the shortest prefix of $x_1 \ldots x_{|Z|}$ such that M_{d-1}^* restricted to the keys in W is a $(d-1)$-dependent matching, i.e., we go through the pairs $\{x_1, x_2\}, \{x_3, x_4\}, \ldots$ until the set of their diff-keys (up to level $d-1$) contains a zero-set. Note that such a $W \subseteq Z$ always exists because M_{d-1}^* itself is a $(d-1)$-zero matching. Also note that $W \backslash \{x_w\} \subseteq Z \backslash \{x^*\}$. Let $e_{d-1} := \{x_{w-1}, x_w\}$ be the last pair in the prefix, and as $e_{d-1} \in M_{d-1}^*$, we get that w is even. We then define M_{d-1} to be the restriction of M_{d-1}^* to the keys in W and M_d to be the restriction of M_d^* to the keys in $W \setminus \{x_w\}$. Note that M_{d-1} is a perfect matching on W while M_d is a maximal matching on $W \backslash \{x_w\}$. Since M_{d-1} is $(d-1)$-dependent, we can define a submatching $L_{d-1} \subseteq M_{d-1}$ such that L_{d-1} is a $(d-1)$-zero matching (this corresponds exactly to the subset of $\mathrm{DiffKeys}(M_{d-1}, d-1)$ that is a zero-set). By construction, $e_{d-1} \in L_{d-1}$. Finally, we set $Z_{d-1} = \bigcup L_{d-1}$ and notice that Z_{d-1} is an $(d-1)$-zero key set (by Observation 11).

A special total order. We now define a special new total order $\preceq$ on Σ^c that we use in order to index the keys in W and describe matchings on levels $< d-1$. Here x_w has a special role and we place it in a special position; namely at the end. More precisely, we have the natural $\leqslant$-order on Σ^c, i.e, in which keys are viewed as numbers $< |\Sigma|^c$. We define $\preceq$ to be exactly as $\leqslant$ except that we set x_w to be the largest element. Moreover, we extend the total order $\preceq$ to disjoint edges in a matching M: given $\{x_1, x_2\}, \{y_1, y_2\} \in M$, we define $\{x_1, x_2\} \preceq \{y_1, y_2\}$ if and only if $\min x_i \prec \min_i y_i$.

Lower levels. Now for $i = d-2 \dots 0$, we do the following: from level $i+1$, we have an $(i+1)$-zero set Z_{i+1}. By Observation 12, there exists a perfect i-matching M_i^* over Z_{i+1}. Notice that M_i^* is an i-zero matching. We define M_i as the shortest prefix (according to $\preceq$) of M_i^* that is i-dependent. Denote by e_i the $\preceq$-maximum edge in M_i. Define the submatching $L_i \subseteq M_i$ such that L_i is an i-zero set. By construction, $e_i \in L_i$. Finally, we set $Z_i = \bigcup L_i$ and notice that Z_i is an i-zero set.

3.3 Characterizing an obstruction

Our main proof strategy will be to show that an obstruction is unlikely to happen, implying that our selected derived keys $\widetilde{X}$ are unlikely to be linearly dependent. To get to that point, we first characterize such an obstruction in a way that is independent of how it was derived in Section 3.2. With this classification in hand, we will then be able to make a union bound over all possible obstructions.

Corresponding to the two top levels, our obstruction consists of the following objects:

- A set of keys $W \subseteq \Sigma^c$ of some size $w = |W|$.
- A special key $x_w \in W$, that we put last when we define $\preceq$.
- A maximal matching M_d on $W \backslash \{x_w\}$.
- A perfect matching M_{d-1} on W, where e_{d-1} is the only edge in M_{d-1} incident to x_w.
- A submatching $L_{d-1} \subseteq M_{d-1}$, which includes e_{d-1}, and its support $Z_{d-1} = \bigcup L_{d-1}$.

Note that the above objects do not include the whole selected set $X^{f,h}$, the d-zero set Z, or the perfect matchings M_{d-1}^*, M_d^* that we used in our construction of the obstruction. Also, note that thus far, the objects have not mentioned any relation to the hash function h.

To describe the lower levels, we need to define greedy matchings. We say a matching M on a set Y is *greedy* with respect to the total order $\preceq$ on Y if either: (i) $M = \varnothing$ or; (ii) the key $\min_{\preceq} Y$ is incident to some $e \in M$ and $M \backslash \{e\}$ is greedy on $Y \backslash e$. Assuming that $|Y|$ is even, we note that M is greedy if and only if it is a prefix of some $\preceq$-ordered perfect matching M^* on Y.

For the lower levels $i = d-2, \dots, 1$, we then have the following objects:

- A greedy matching M_i on Z_{i+1}. Denote with e_i the $\preceq$-maximum edge in M_i.
- A submatching $L_i \subseteq M_i$ which includes e_i, and its support $Z_i = \bigcup L_i$.

Note that the above objects describe all possible obstructions that we might construct. Moreover, any obstruction can be uniquely described as the tuple of objects

$$(W, x_w, M_d, M_{d-1}, e_{d-1}, L_{d-1}, Z_{d-1}, \dots M_1, e_1, L_1, Z_1) \ .$$

3.4 Confirming an obstruction

In order for a given obstruction to actually occur, the tornado tabulation hash function $h = \widehat{h} \circ \widetilde{h}$ must satisfy the following conditions with respect to it:

- The keys in W are all selected, that is, $W \subseteq X^{f,h}$.

- Either $W \subseteq Q$ or $Q \cup W \backslash \{x_w\}$ is d-independent.
- For $i = 1, \ldots, d$, M_i is an i-matching.
- M_d is $(d-1)$-independent.

For $i = 1, \ldots, d-1$,

- $M_i \backslash \{e_i\}$ is $(i-1)$-independent.
- The submatching L_i is i-zero (implying that $Z_i = \bigcup L_i$ is an i-zero set).

When a hash function h satisfies the above conditions, we say that it *confirms* an obstruction. We now need to argue two things. We need (1) to verify that our construction of an obstruction from Section 3.2 satisfies these conditions on h, and (2) that, given a fixed obstruction, the probability that a random h confirms the obstruction is very small. This is captured by the two lemmas below.

Lemma 13. *The obstruction constructed in Section 3.2 satisfies the conditions on $h = \widehat{h} \circ \widetilde{h}$.*

Proof. Since $W \backslash \{x_w\} \subseteq Z \backslash \{x^*\}$, the choice of Z implies that either $W \subseteq Q$ or $Q \cup W \backslash \{x_w\}$ is d-independent.

The matchings M_i were all submatchings of perfect i-matchings. We also picked M_i is minimally i-dependent, but for an i-matching M_i this also implies that M_i is minimally $(i-1)$-dependent, and therefore $M_i \backslash \{e_i\}$ is $(i-1)$-independent. Finally, we constructed L_i and Z_i to be i-zero. □

Lemma 14. *Given the objects of an obstruction, the probability that $h = \widehat{h} \circ \widetilde{h}$ confirms the obstruction is at most*

$$\left(\prod_{x \in W \backslash \{x_w\}} p_x \right) \Big/ |\Sigma|^{w-2+\sum_{i=1}^{d-2}(|M_i|-1)}.$$

Proof. For simplicity, we group the conditions above in the following events: we define the event $\mathcal{C}_S$ to be the event that $W \backslash \{x_w\} \subseteq X^{f,h}$ given that the set $W \backslash \{x_w\}$ is d-independent. Then, for each $i \in \{1, \ldots, d-1\}$, we define $\mathcal{C}^{(i)}$ to be the event that $M_i \backslash e_i$ is an i-matching conditioned on the fact that it is $(i-1)$-independent. Finally, we define the last event $\mathcal{C}^{(d)}$ to be the event that M_d is a d-matching conditioned on the fact that it is $(d-1)$-independent. It is then sufficient to show that:

$$\Pr\left(\mathcal{C}_S \wedge \bigwedge_{i=1}^{d} \mathcal{C}^{(i)} \right) \leqslant \left(\prod_{x \in W \backslash \{x_w\}} p_x \right) \Big/ |\Sigma|^{w-2+\sum_{i=1}^{d-2}(|M_i|-1)}.$$

We proceed from the bottom up, in the following sense. For every $i \in \{1, \ldots, d-1\}$, the randomness of the event $\mathcal{C}^{(i)}$ conditioned on $\bigwedge_{j=1}^{i-1} \mathcal{C}^{(j)}$ depends solely on $\widetilde{h}_i$. We invoke Lemma 10 for $M_i \backslash e_i$ and get that

$$\Pr\left(\mathcal{C}^{(i)} \;\middle|\; \bigwedge_{j=1}^{i-1} \mathcal{C}^{(j)} \right) \leqslant 1/|\Sigma|^{|M_i|-1} \ .$$

When it comes to level d, from Lemma 10 applied to M_d, we get that the probability of $\mathcal{C}^{(d)}$ conditioned on $\bigwedge_{j=1}^{d-1} \mathcal{C}^{(j)}$ is at most $1/|\Sigma|^{|M_d|}$. We now notice that $|M_{d-1}| + |M_d| = w - 1$ by construction, and so:

$$\Pr\left(\bigwedge_{j=1}^{d} \mathcal{C}^{(j)}\right) \leqslant 1/|\Sigma|^{w-2+\sum_{i=1}^{d-2}(|M_i|-1)} .$$

Finally, we know that either $W \subseteq Q$ or $Q \cup W\backslash\{x_w\}$ is d-independent. If $W \subseteq Q$, then $p_x = 1$ for all $x \in W$, and therefore, trivially,

$$\Pr\left(\mathcal{C}_S \;\middle|\; \bigwedge_{j=1}^{d} \mathcal{C}^{(j)}\right) \leqslant \prod_{x \in W\backslash\{x_w\}} p_x = 1 .$$

Otherwise $Q \cup W\backslash\{x_w\}$ is d-independent. This means that the derived keys $\widetilde{h}(Q \cup W\backslash\{x_w\})$ are linearly independent. We can then apply Lemma 9 to this set of derived keys and the final simple tabulation hash function $\widehat{h}$. We get that the final hash values $h(Q \cup W\backslash\{x_w\})$ are chosen independently and uniformly at random. For any one element x, by definition, $\Pr\left(x \in X^{f,h}\right) \leqslant p_x$ and so:

$$\Pr\left(\mathcal{C}_S \;\middle|\; \bigwedge_{j=1}^{d} \mathcal{C}^{(j)}\right) \leqslant \prod_{x \in W\backslash\{x_w\}} p_x .$$

This completes the claim. □

4 Simplified analysis

In this section, we present a simplified analysis showing that, under the hypotheses of our theorem, $\widetilde{h}(X^{f,h})$ is linearly dependent with probability at most $\Theta(\mu^3(17/|\Sigma|)^d) + 2^{-|\Sigma|/8}$. Later, we will replace $(17/|\Sigma|)^d$ with $(3/|\Sigma|)^d$, which essentially matches the growth in our lower bound.

For now we use a fixed limit $w_0 = |\Sigma|/2^{5/2}$ on the set size $w = |W|$. With this limited set size, we will derive the $\Theta(\mu^3(17/|\Sigma|)^d)$ bound. The $2^{-|\Sigma|/8}$ bound will stem for sets of bigger size and will be derived in a quite different way.

Our goal is to study the probability that there exists a combinatorial obstruction agreeing with a random tornado hash function h; if not, $\widetilde{h}(X^{f,h})$ is linearly independent. To do this, we consider a union bound over all combinatorial obstructions as such:

$$\sum_{W,x_w,M_d,M_{d-1},e_{d-1},L_{d-1},Z_{d-1},\ldots M_1,e_1,L_1,Z_1} \Pr_h[h \text{ confirms the obstruction}] \tag{4}$$

The above sum is informally written in that we assume that each element respects the previous elements of the obstruction, e.g., for $i < d-2$, M_i is a greedy matching over Z_{i+1}. Likewise, in the probability term, it is understood that h is supposed to confirm the obstruction whose combinatorial description is $(W, x_w, M_d, M_{d-1}, e_{d-1}, L_{d-1}, Z_{d-1}, \ldots M_1, e_1, L_1, Z_1)$.

Using Lemma 14, we bound (4) by

$$\sum_{W,x_w,M_d,M_{d-1},e_{d-1},L_{d-1},Z_{d-1},\ldots M_1,e_1,L_1,Z_1} \left(\prod_{x\in W\backslash\{x_w\}} p_x\right) \Big/ |\Sigma|^{w-2+\sum_{i=1}^{d-2}(|M_i|-1)}$$

$$\leqslant \left(\sum_{W,x_w,M_d,M_{d-1},e_{d-1},L_{d-1},Z_{d-1}} \left(\prod_{x\in W\backslash\{x_w\}} p_x\right) |\Sigma|^{2-w} 2^{w/4-1}\right) \tag{5}$$

$$\times \prod_{i=1}^{d-2} \max_{Z_{i+1}} \left(\sum_{M_i,e_i,L_i,Z_i} |\Sigma|^{1-|M_i|} \Big/ 2^{(|Z_{i+1}|-|Z_i|)/4}\right). \tag{6}$$

Above , $W, x_w, M_d, M_{d-1}, e_{d-1}, L_{d-1}, Z_{d-1}$ are all the elements from the top two levels and we refer to (5) as the "top" factor. We refer to Equation (6) as the "bottom" factor which is the product of "level" factors.

For each lower level $i \leqslant d-2$, the elements M_i, e_i, L_i, Z_i are only limited by Z_{i+1}, so for a uniform bound, we just consider the maximizing choice of Z_{i+1}. For this to be meaningful, we divided the level factor (6) by $2^{(|Z_{i+1}|-|Z_i|)/4}$ and multiplied (5) by $2^{w/4-1} \geqslant \prod_{i=1}^{d-2} 2^{(|Z_{i+1}|-|Z_i|)/4}$. This inequality follows because $|Z_{d-1}| \leqslant w$ and $|Z_1| \geqslant 4$. The exponential decrease in $|Z_{i+1}|$ helps ensuring that the maximizing choice of Z_{i+1} has bounded size (and is not infinite). We note here that our bound $|W| \leqslant w_0 = |\Sigma|/2^{5/2}$ is only needed when bounding the level factors, where it implies that $|Z_{i+1}| \leqslant w_0$.

4.1 The top two levels and special indexing for matchings and zero sets

We now wish to bound the top factor (5) from the two top levels. Below, we will first consider w fixed. Later we will sum over all relevant values of w.

We want to specify the w keys in W using the fact that the keys from $W\backslash\{x_w\}$ hash independently. Thus, we claim that we only have to specify the set $V = W\backslash\{x_w\}$, getting x_w for free. By construction, we have x_w in the zero-set Z_{d-1}, so

$$x_w = \bigtriangleup(Z_{d-1}\backslash\{x_w\}). \tag{7}$$

To benefit from this zero-set equality, we need the special ordering $\preceq$ from the construction. It uses the standard ordering $\leqslant$ of V since all these keys are known, and then it just places the yet unknown key x_w last. The special ordering yields and indexing $x_1 \prec x_2 \prec \cdots \prec x_w$ (this is not the order in which we traversed the cycles in the construction except that x_w is last in W in both cases). Formally, we can now specify all the M_i, L_i, Z_i over the index set $\{1,\ldots,w\}$. For $i < w$, we directly identify x_i as the ith element in V. This way we identify all $x_i \in Z_{d-1}\backslash\{x_w\}$, and then we can finally compute the special last key x_w; meaning that we completely resolve the correspondance between indices and keys in W. As a result, we can bound (5) by

$$\sum_{V:|V|=w-1} \left(\prod_{x\in V} p_x\right) \times \sum_{M_d,M_{d-1},e_{d-1},L_{d-1},Z_{d-1}} |\Sigma|^{2-w} 2^{w/4-1}.$$

For the first part, with $v = w-1$, we have

$$\sum_{V:|V|=v} \left(\prod_{x\in V} p_x\right) = \frac{1}{v!} \sum_{(x_1,\ldots,x_v)} \prod_{i=1}^{v} p_{x_i} \leqslant \frac{1}{v!} \prod_{i=1}^{v} (\sum_x p_x) = \frac{\mu^v}{v!}. \tag{8}$$

For the second part, we note M_{d-1} and M_d can both be chosen in $(w-1)!!$ ways and we know that e_{d-1} is the edge in M_{d-1} incident to w. Since the submatching L_{d-1} of M_{d-1} contains the known e_{d-1}, it can be chosen in at most $2^{|M_{d-1}|-1} = 2^{w/2-1}$ ways. Putting this together, we bound (5) by

$$\frac{\mu^{w-1}}{(w-1)!}((w-1)!!)^2\, 2^{w/2-1}|\Sigma|^{-w+2}2^{w/4-1} = \frac{\mu^{w-1}}{|\Sigma|^{w-2}} \cdot \frac{(w-1)!!}{(w-2)!!} \cdot 2^{3w/4-2}$$
$$\leqslant \frac{\mu^3}{|\Sigma|^2}2^{4-w} \cdot \frac{(w-1)!!}{(w-2)!!} \cdot 2^{3w/4-2} = \frac{\mu^3}{|\Sigma|^2} \cdot \frac{(w-1)!!}{(w-2)!!} \cdot 2^{2-w/4} \tag{9}$$

Above we got to the second line using our assumption that $\mu \leqslant |\Sigma|/2$. This covers our top factor (5), including specifying the set Z_{d-1} that we need for lower levels.

Union bound over all relevant sizes. We will now sum our bound (9) for a fixed set size w over all relevant set sizes $w \leqslant w_0$, that is, $w = 4, 6, \ldots, w_0$. The factor that depends on w is

$$f(w) = \frac{(w-1)!!}{(w-2)!!}/2^{w/4}.$$

We note that $f(w+2) = f(w)\frac{w+1}{\sqrt{2}w}$ and $\frac{w+1}{\sqrt{2}w} < 4/5$ for $w \geqslant 8$, so

$$\sum_{\text{Even } w=4}^{w_0} f(w) < f(4) + f(6) + 5f(8) < 4.15.$$

Thus we bound the top factor (5) over all sets W of size up to w_0 by

$$\sum_{\text{Even } w=4}^{w_0} \left(\frac{\mu^3}{|\Sigma|^2} \cdot \frac{(w-1)!!}{(w-2)!!} \cdot 2^{2-w/4} \right) < 16.6(\mu^3/|\Sigma|^2). \tag{10}$$

4.2 Lower levels with greedy matchings

We now focus on a lower level $i \leqslant d-2$ where we will bound the level factor

$$\max_{Z_{i+1}} \left(\sum_{M_i, e_i, L_i, Z_i} |\Sigma|^{1-|M_i|} \Big/ 2^{(|Z_{i+1}|-|Z_i|)/4} \right). \tag{11}$$

We want a bound of $O(1/|\Sigma|)$, implying a bound of $O(1/|\Sigma|)^{d-2}$ for all the lower levels in (6).

All that matters for our bounds is the cardinalities of the different sets, and we set $m_i = |M_i|$ and $z_i = |Z_i|$. For now we assume that $z_{i+1} \leqslant w_0$ and m_i are given.

Lemma 15. *With a given linear ordering over a set S of size n, the number of greedy matchings of size k over S is $(n-1)^{\underline{k}}$.*

Proof. We specify the edges one at the time. For greedy matchings, when we pick the jth edge, the first end-point is the smallest free point in S and then there are $n-2j+1$ choices for its match, so the number of possible greedy matchings of size k is $(n-1)^{\underline{k}}$. □

Recall that e_i denotes the last edge in our greedy matching M_i and that $e_i \in L_i$. In our case, we are only going to specify $M_i' = M_i \backslash \{e_i\}$ and $L_i' = L_i \backslash \{e_i\}$. The point is that if we know M_i' and L_i', then we can compute e_i. More precisely, we know that e_i is the next greedy edge to be added to M_i', so we know its first end-point x. We also know that $Z_i = \bigcup L_i$ is a zero-set, so the other end-point can be computed as key

$$y = \bigtriangleup(\{x\} \cup \bigcup L_i') \tag{12}$$

Above we note that even though the keys in x and L_i' are only specified as indices, we know how to translate between keys and indices, so we can compute the key y and then translate it back to an index.

By Lemma 15, we have $(z_{i+1}-1)^{\underline{\underline{m_i-1}}}$ choices for M_i', and then there are 2^{m_i-1} possible submatchings $L_i' \subset M_i'$. The number of combinations for M_i, e_i, L_i is thus bounded by

$$(z_{i+1}-1)^{\underline{\underline{m_i-1}}} \cdot 2^{m_i-1}.$$

We want to multiply this by

$$|\Sigma|^{1-m_i}/2^{(z_{i+1}-z_i)/4}.$$

Here $z_i = 2|L_i| \leqslant 2m_i$, so we get a bound of

$$(z_{i+1}-1)^{\underline{\underline{m_i-1}}} \cdot (2^{3/2}/|\Sigma|)^{m_i-1}/2^{z_{i+1}/4-1/2} \leqslant 2^{1/2}(2^{3/2}z_{i+1}/|\Sigma|)^{m_i-1}/2^{z_{i+1}/4}. \tag{13}$$

We now note that

$$(2^{3/2}z_{i+1}/|\Sigma|) \leqslant 1/2.$$

since $z_{i+1} \leqslant w \leqslant w_0 = |\Sigma|/2^{5/2}$. Having this factor bounded below 1 is critical because it implies that the term decreases as m_i grows.

Still keeping z_{i+1} fixed, we will now sum over all possible values of m_i. Since M_i contains a zero-matching L_i of size at least 2, that is 2 edges covering 4 keys, we have that $m_i \geqslant 2$. Moreover, the terms of the summation are halving, hence they sum to at most twice the initial bound, so we get

$$\sum_{m_i \geqslant 2} 2^{1/2}(2^{3/2}z_{i+1}/|\Sigma|)^{m_i-1}/2^{z_{i+1}/4} \leqslant 2^{3/2}(2^{3/2}z_{i+1}/|\Sigma|)/2^{z_{i+1}/4}. \tag{14}$$

The maximizing real value of z_{i+1} is $4/\ln 2 = 5.77$, but z_{i+1} also has to be an even number, that is, either 4 or 6, and 6 yeilds the maximum bound leading to the overall bound of $16.98/|\Sigma|$ for (6). Together with our bound (10), we get a total bound of

$$16.6(\mu^3/|\Sigma|^2) \cdot (17/|\Sigma|)^{d-2}. \tag{15}$$

4.3 Large set sizes

We now consider sets W of sizes $w > w_0$. In this case, we will only consider the two top levels of the obstructions. Recall that we have a cycle traversal $x_1, \ldots, x_w$. If $w > w_0$, we only use the prefix $x_1, \ldots, x_{w_0}$, where we require that w_0 is even. The two matchings M_d and M_{d-1} are reduced accordingly. Then M_{d-1} is a perfect matching on $W_0 = \{x_1, \ldots, x_{w_0}\}$ while M_d is maximal excluding x_{w_0}. Each of these matchings can be chosen in $(w_0-1)!!$ ways.

This time we know that we have full independence. We know that $\widetilde{W}_0$ is a strict subset of the original minimimal zero set $\widetilde{Z}$ of derived keys, so the keys in W_0 all hash independently. Therefore the probability that they are all selected is bounded by $\prod_{x\in W_0} p_x$.

Also, since we terminate the traversal early, we know that M_{d-1} is $(d-2)$-linearly independent, and M_d must be $d-2$-linearly independent, so the probability that these two matchings are satisfied is $1/|\Sigma|^{|M_{d-1}|+|M_d|} \leqslant 1/|\Sigma|^{w_0-1}$.

The total bound for all $w > w_0 = |\Sigma|/2^{5/2}$ is now

$$\sum_{W_0:\,|W_0|=w_0,M_d,M_{d-1}} \left(\prod_{x\in W_0} p_x\right)/|\Sigma|^{w_0-1} \leqslant \frac{\mu^{w_0}}{w_0!}((w_0-1)!!)^2/|\Sigma|^{w_0-1} \leqslant w_0|\Sigma|/2^{w_0} \tag{16}$$

$$\leqslant 1/2^{|\Sigma|/8}. \tag{17}$$

The last step holds easily for $|\Sigma| \geqslant 256$. We note that (16) holds for any choice of w_0, limiting the probability of any obstruction with $|W| < w$.

5 Tighter analysis

We will now tighten the analysis to prove that

Theorem 5. *Let $h : \Sigma^c \to \mathcal{R}$ be a tornado tabulation hash function with d derived characters and f be an s-selector as described above. If $\mu^f \leqslant \Sigma/2$, then $h^{(t)}$ is fully random on $X^{f,h^{(s)}}$ with probability at least*

$$1 - \mathit{DependenceProb}(\mu^f, d, \Sigma)\ .$$

5.1 Bottom analysis revisited

We first tigthen the analysis of the bottom factor (6) so as to get a bound of $O((3/\Sigma)^{d-2})$ and, together with our top bound (10), obtain an overall bound of $O(\mu^3(3/\Sigma)^d)$ matching our lower bound within a constant factor. We are still using our assumption that $w \leqslant w_0 \leqslant |\Sigma|/2^{5/2}$ implying that $z_{i+1} \leqslant |\Sigma|/2^{5/2}$.

First we look at a single level i. For a tighter analysis of the level factor (11), we partition into cases depending on z_{i+1} and to some degree on $z_i = 2m_i$. Recall that z_{i+1} is given from the level above.

If $z_{i+1} \leqslant 6$, then we must have $z_i = z_{i+1}$, since we cannot split into two zero sets each of size at least 4. This implies that the factor $2^{|Z_{i+1}|-|Z_i|}$ is just one. Also, in this case, $M_i = L_i$ must be a perfect matching on $Z_{i+1} = Z_i$, and such a perfect matching can be in $(z_{i+1}-1)!!$ ways. Thus, for given $z_{i+1} \leqslant 6$, we bound the level factor by

$$(z_{i+1}-1)!!/|\Sigma|^{z_{i+1}/2-1} \leqslant 3/|\Sigma|. \tag{18}$$

with equality for $z_{i+1} = 4$. As a result, if $z_{d-1} \leqslant 6$, then we have already achieved a bound of $(3/|\Sigma|)^{d-2}$ for the bottom factor.

Split levels. We now consider $z_{i+1} \geqslant 8$. Now it is possible that Z_{i+1} can split into two sets so that $z_i < z_{i+1}$. For a given m_i and z_{i+1}, we already had the bound (13)

$$2^{1/2}(2^{3/2}z_{i+1}/|\Sigma|)^{m_i-1}/2^{z_{i+1}/4}.$$

Since $(2^{3/2}z_{i+1}/|\Sigma|) \leqslant 1/2$, the worst case is achieved when $m_i = 2$, but here we can do a bit better. In (13) we had a factor 2^{m_i-1} to specify the subset L_i of M_i containing e_i, but since L_i should have size at least 4, we have $L_i = M_i$. Thus, for $m_i = 2$ and given z_{i+1}, we improve (13) to

$$2(z_{i+1}/|\Sigma|)/2^{z_{i+1}/4}.$$

For $z_{i+1} \geqslant 8$, this is maximized with $z_{i+1} = 8$. Thus for $m_i = 2$ and $z_{i+1} \geqslant 8$, the level factor (11) is bounded by

$$(2 \cdot 8/|\Sigma|)/2^2 = 4/|\Sigma|.$$

Now, for $m_i \geqslant 3$, we just use the bound (13). As in (14), we get

$$\sum_{m_i \geqslant 3} 2^{1/2}(2^{3/2}z_{i+1}/|\Sigma|)^{m_i-1}/2^{z_{i+1}/4} \leqslant 2 \cdot 2^{1/2}(2^{3/2}z_{i+1}/|\Sigma|)^2/2^{z_{i+1}/4}$$

Over the reals, this is maximized with $z_{i+1} = 2 \cdot 4/\ln 2 \approx 11.52$, but we want the maximizing even z_{i+1} which is 12, and then we get a bound of $1.6/|\Sigma|$.

We now want to bound the whole bottom factor in the case where $z_{d-1} \geqslant 8$. Let j be the lowest level with $z_{j+1} \geqslant 8$. For all lower levels, if any, the level factor is bounded by $(3/|\Sigma|)$. Also, for all higher levels $i > j$, we have $2m_i = z_i \geqslant 8$, hence $m_i \geqslant 4$, so the level factor for higher levels is bounded by our last $1.6/|\Sigma|$. The worst case is the level j, where we could get any m_s (note that if $s = 1$, we have no guarantee that $m_i \leqslant 2$), hence we have to add the bound $4/|\Sigma|$ for $m_s = 2$ with the bound $1.6|\Sigma|$ for $m_s \geqslant 3$, for a total bound of $5.6/|\Sigma|$.

Thus, for a given j, the bottom factor is bounded by

$$(3/|\Sigma|)^{j-1}(5.6/|\Sigma|)(1.6/|\Sigma|)^{d-2-j} = (3/|\Sigma|)^{d-2} * (5.6/3) * (1.6/3)^{d-2-j}.$$

Summing, this over $j = 1, \ldots, d-2 \geqslant 1$, we get a bound of at most

$$(3/|\Sigma|)^{d-2}(5.6/3)(1/(1-1.6/3)-1) < 4(3/|\Sigma|)^{d-2}\ . \tag{19}$$

This is our bound for the whole bottom factor when we maximized with $z_{d-1} \geqslant 8$. Since it is larger than our bound of $3/|\Sigma|)^{d-2}$ when we maximized with $z_{d-1} \leqslant 6$, we conclude that it is also our bound if we maximize over any value of z_{d-1}. In combination with our top factor, $16.6(\mu^3/|\Sigma|^2)$ from (10), we get a combined bound of

$$16.6(\mu/|\Sigma|^2)4(3/|\Sigma|)^{d-2} \leqslant 7\mu^3(3/|\Sigma|)^d. \tag{20}$$

5.2 Increasing the maximization range

We will now show how to deal with larger sets up to size $w_0^+ = 0.63|\Sigma|$. The level factor with a fixed z_{i+1} and m_i has the following tight version from (13)

$$\begin{aligned}&(z_{i+1}-1)^{\underline{\underline{m_i-1}}} \cdot (2^{3/2}/|\Sigma|)^{m_i-1}/2^{z_{i+1}/4-1/2}\\ &= 2^{1/4}f(m_i-1, z_{i+1}-1) \text{ where } f(x,y) = y^{\underline{\underline{x}}} \cdot (2^{3/2}/|\Sigma|)^x/2^{y/4}.\end{aligned}$$

We want to find the sum over relevant m_i with the maximizing z_i. However, we already considered all even $z_{i+1} \leqslant |\Sigma|/2^{5/2}$, so it suffices to consider $z_{i+1} \in (|\Sigma|/2^{5/2}, |\Sigma|/2]$. Also, for a given z_{i+1}, we have to sum over all $m_i \in [2, z_{i+1}/2]$. Thus we want to bound

$$\max_{\text{odd } y \in [|\Sigma|/2^{5/2}, |\Sigma|/2)} \sum_{x=1}^{\lfloor y/2 \rfloor} f(x,y).$$

Note that $f(x+1,y) = f(x,y) \cdot (y-2x)(2^{3/2}/|\Sigma|)$. Hence $f(x+1,y) < f(x,y) \iff y-2x < |\Sigma|/2^{3/2}$.

Assume that $y \geqslant |\Sigma|/2^{3/2}$ and consider the smallest integer x_y such that $y - 2x_y < |\Sigma|/2^{3/2}$. For a given value of y this x_y maximizes $f(x_y, y)$.

To bound $f(x_y, y)$, we note that

$$y^{\underline{x}} < (y-x+1)^x$$

We have $y < w_0^+ = 0.63|\Sigma|$ and $y - 2x_y < |\Sigma|/2^{3/2}$, so

$$y - x_y + 1 \leqslant (0.63 + 1/2^{3/2})|\Sigma|/2 + 1 \leqslant |\Sigma|/2.$$

Therefore

$$f(x_y, y) \leqslant y^{\underline{x_y}} \cdot (2^{3/2}/|\Sigma|)^{x_y}/2^{y/4} \leqslant ((y - x_y + 1)2^{3/2}/|\Sigma|)^{x_y}/2^{y/4} \leqslant 1/2^{(y-2x_y)/4}.$$

We also have $y - 2(x_y - 1) \geqslant |\Sigma|/2^{3/2}$, so we get

$$f(x_y, y) \leqslant 1/2^{(|\Sigma|/2^{3/2}-2)/4} = 1/2^{|\Sigma|/2^{7/2}-1/2}.$$

For $|\Sigma| \geqslant 256$, this is below $0.015/|\Sigma|^2$, so even if we sum over all $x \leqslant y/2 \leqslant |\Sigma|/2$, we get a bound below $0.008/|\Sigma|$, that is,

$$\max_{y \in [|\Sigma|/2^{3/2}, w_0^+]} \sum_{x=1}^{\lfloor y/2 \rfloor} f(x,y) \leqslant 0.08/|\Sigma|.$$

Now consider $y < |\Sigma|/2^{3/2}$. Then $f(x,y)$ is decreasing in x starting $f(0,y) = 1/2^{y/4}$. Summing over all $x \leqslant y/2$, we get $(y/2)/2^{y/4}$. This expression is maximized with $y = 4/\ln 2$, so for $y \geqslant |\Sigma|/4$, we get a maximum of $(|\Sigma|/8)/2^{|\Sigma|/16}$. Now $2^{|\Sigma|/16} \geqslant |\Sigma|^2$ for $|\Sigma| \geqslant 256$, so we end up with

$$\max_{y \in [|\Sigma|/4|, |\Sigma|/2^{3/2}]} \sum_{x=1}^{\lfloor y/2 \rfloor} f(x,y) \leqslant 1/(8|\Sigma|).$$

Finally we consider $y \in [|\Sigma|/2^{5/2}, |\Sigma|/4]$. Then $f(x+1,y) \leqslant f(x,y)(y\,2^{3/2}/|\Sigma|) \leqslant f(x,y)/2^{1/2}$, so

$$\sum_{x=0}^{\infty} f(x,y) \leqslant f(0,y)/(1-2^{-1/2}) \leqslant 1/((1-2^{-1/2})2^{y/4}).$$

With $y \geqslant |\Sigma|/2^{5/2}$, this is maximized with $y = |\Sigma|/2^{5/2}$, so we get the bound

$$\max_{y \in [|\Sigma|/2^{5/2}, |\Sigma|/4]} \sum_{x=0}^{\infty} f(x,y) \leqslant 1/((1-2^{-1/2})2^{(|\Sigma|/2^{9/2})}) \leqslant 0.344/|\Sigma|.$$

Putting all our bounds together, we conclude that

$$\max_{\text{odd } y \in [|\Sigma|/2^{5/2}, w_0^+)} \sum_{x=1}^{\lfloor y/2 \rfloor} f(x,y) \leqslant 0.344/|\Sigma|.$$

Our bound for the level factor with $z_{i+1} < (|\Sigma|/2^{5/2}, |\Sigma|/2]$ is $2^{1/2}$ times bigger, but this is still much smaller than all the bounds from Section 5.1 based on smaller z_{i+1}. Thus we conclude that the analysis from Section 5.1 is valid even if allow z_{i+1} to go the whole way up to $w_0^+ = 0.63|\Sigma|$. Therefore (20) bounds the probability of any obstruction with set size $|W| \leqslant w_0^+$.

However, for the probability of obstructions with sets sizes $|W| > w_0^+$, we can apply (16), concluding that they happen with probability bounded by

$$w_0^+ |\Sigma|/2^{w_0^+} = 0.63|\Sigma|^2/2^{0.63|\Sigma|} < 1/2^{|\Sigma|/2}.$$

The last step used $|\Sigma| \geqslant 256$. Together with (20) we have thus proved that the probability of any obstruction is bounded by

$$7\mu^3(3/|\Sigma|)^d + 1/2^{|\Sigma|/2}. \tag{21}$$

This is for simple tornado hashing without the twist.

5.3 Tornado hashing including the twist

We will now return to the original tornado hashing with the twisted character

$$\widetilde{x}[c] = x[c] \oplus \widetilde{h}_0(x[< c]).$$

This twist does not increase the number of lookups: it is still $c + d$ with c input characters and d derived characters, so the speed should be almost the same, but we will gain a factor $3/|\Sigma|$ in the probability, like getting an extra derived character for free.

The obstruction is constructed exactly as before except that we continue down to level 0 rather than stopping at level 1. All definitions for level i are now just applied for $i = 0$ as well. However, we need to reconsider some parts of the analysis. First, we need to prove that Lemma 10 also holds for $i = 0$:

Lemma 10 *Let M be a partial matching on Σ^c. Conditioning on M being $(i-1)$-independent, M is an i-matching with probability $1/|\Sigma|^{|M|}$.*

Proof. Since M is (-1)-independent, we know that

$$\text{DiffKeys}(M, -1) = \{(\widetilde{x} \mathbin{\triangle} \widetilde{y})[< c] \mid \{x, y\} \in M\}$$

is linearly independent. We note here that $(\widetilde{x} \mathbin{\triangle} \widetilde{y})[< c] = (x \mathbin{\triangle} y)[< c]$. We want to know the probability that M is a 0-matching, that is, the probability that $\tilde{x}[c] = \tilde{y}[c]$ for each $\{x, y\} \in M$. For $i > 0$, we had

$$\tilde{x}[c+i] = \tilde{y}[c+i] \iff \widetilde{h}_i((\widetilde{x} \mathbin{\triangle} \widetilde{y})[< c+i]) = 0.$$

However, for $i = 0$, we have

$$\tilde{x}[c] = \tilde{y}[c] \iff \tilde{h}_0((\widetilde{x} \mathbin{\triangle} \widetilde{y})[< c]) = x[c] \oplus y[c].$$

However, Lemma 9 states that the simple tabulation hash function $\tilde{h}_0$ is fully random on the linearly indepedent DiffKeys$(M, -1)$, so we still conclude that M is a 0-matching with probability $1/\Sigma^{|M|}$. □

There is one more thing to consider. We have generally used that if Z is an i-zero set, that is, if $\widetilde{Z}[\leqslant c+i]$ is a zero-set, then Z is also a zero-set. However, this may no longer be the case. All we know is that $\widetilde{Z}[\leqslant c]$ is a zero-set. This also implies that $Z[< c] = \widetilde{Z}[< c]$ is a zero-set. However, we claim that

$$\bigoplus Z = 0 \tag{22}$$

Here $\bigoplus Z$ is xoring that keys in Z viewed as bit-strings. Note that $\triangle Z = \varnothing$ implies $\bigoplus Z = 0$. Since $Z[< c]$ is a zero set, we know that $\{\widetilde{h}_0(x[< c]) \mid x \in Z\}$ is a zero set. We also know that $\widetilde{Z}[c]$ is a zero set and it is equal to $\{\widetilde{h}_0(x[< c]) \oplus x[c] \mid x \in Z\}$. Thus we have

$$\bigoplus\{\widetilde{h}_0(x[< c]) \oplus x[c] \mid x \in Z\} = 0 = \bigoplus\{\widetilde{h}_0(x[< c]) \mid x \in Z\}$$

implying $\bigoplus\{x[c] \mid x \in Z\} = 0$. Together with $Z[< c]$ being a zero set, this settles (22). As a result, for the coding key coding in (7) and (12), we just have to replace $\triangle$ with $\bigoplus$.

No other changes are needed. Level 0 gives exactly the same level factor (11) as the levels $i > 0$, so it is like getting an extra level for free. Therefore with tornado hashing we improve the bottom factor for simple tornado hashing (19) to $4(3/|\Sigma|)^{d-1}$ and the probability of any small obstruction from (20) to $7\mu^3(3/|\Sigma|)^{d+1}$. The probability of any obstruction is thus improved from (21) to

$$7\mu^3(3/|\Sigma|)^{d+1} + 1/2^{|\Sigma|/2}. \tag{23}$$

5.4 Full randomness for large sets of selected keys

While implementing tabulation hashing we often want to set the character size so that all tables fit in cache. Indeed, this is the main reason why tabulation-based hashing schemes are extremely fast in practice. However, restricting the size of Σ constrains how many keys we can expect to be hashed fully randomly: in particular, Theorem 5 states that a set of characters Σ allows for up to $|\Sigma|/2$ selected keys to hash fully randomly with high probability. Most experiments on tabulation-based hashing [1, 2], though, use 8-bit characters (namely $|\Sigma| = 2^8$). This implies that whenever the selected keys are $s \leqslant 2^7$ then they hash uniformly at random with high probability. However, we may want local full randomness for s keys, where $s \gg 2^7$. A trade-off between memory usage and number of keys hashed uniformly is of course unavoidable, however we can improve over the one in Theorem 5 with a clever observation.

More precisely, Theorem 5 states that given a set $X^{f,h}$ of query-selected keys with $\mu^f \leqslant |\Sigma|/2$ their derived keys are linearly dependent with probability at most $\mathsf{DependenceProb}(\mu, d, \Sigma) = 7\mu^3(3/|\Sigma|)^{d+1} + 1/2^{|\Sigma|/2}$ while tornado is using $O(c|\Sigma|)$ words of memory and exactly $c + d$ lookups in tables of size $|\Sigma|$. Recall that μ^f here is the expected size of $X^{f,h}$ for a fully random h, when the queries are chosen by an adaptive adversary. If we want to handle larger sets of selected keys with $\mu^f \gg |\Sigma|/2$ using Theorem 5, then we need to use $O(c + d)$ larger tables of size roughly $2\mu^f$. The clever observation we make here is that, in order to obtain a meaningful probability bound, it is not necessary at all to employ $O(c + d)$ of such larger tables. Indeed, we just need the tables in the top two levels to have size $|\Psi| \geqslant 2\mu^f$ to obtain that the derived keys are linearly dependent with probability at most

$$14|X|^3(3/|\Psi|)^2(3/|\Sigma|)^{d-1} + 1/2^{|\Sigma|/2}$$

losing a factor two with respect to Theorem 5. We name this variant of tornado tabulation tornado-mix tabulation, because the last two derived characters can be evaluated in parallel as in mixed tabulation hashing [10].

Formal definition of tornado-mix. For each $i = 0, \ldots, d-2$ we let $\widetilde{h}_i : \Sigma^{c+i-1} \longrightarrow \Sigma$ be a simple tabulation hash function. For $i = d-1, d$ we let $\widetilde{h}_i : \Sigma^{c+d-2} \longrightarrow \Psi$ be a simple tabulation hash function. Given a key $x \in \Sigma^c$, we define its *derived key* $\widetilde{x} \in \Sigma^{c+d-2} \times \Psi^2$ as $\widetilde{x} = \widetilde{x}_1 \cdots \widetilde{x}_{c+d}$, where

$$\widetilde{x}_i = \begin{cases} x_i & \text{if } i = 1, \ldots, c-1 \\ x_c \oplus \widetilde{h}_0(\widetilde{x}_1 \ldots \widetilde{x}_{c-1}) & \text{if } i = c \\ \widetilde{h}_{i-c}(\widetilde{x}_1 \ldots \widetilde{x}_{i-1}) & \text{if } i = c+1, \ldots, c+d-2. \\ \widetilde{h}_{i-c}(\widetilde{x}_1 \ldots \widetilde{x}_{c+d-2}) & \text{if } i = c+d-1, c+d. \end{cases}$$

Finally, we have a simple tabulation hash function $\widehat{h} : \Sigma^{c+d-2} \times \Psi^2 \longrightarrow \mathcal{R}$, that we apply to the derived key. The *tornado-mix tabulation* hash function $h : \Sigma^c \longrightarrow \mathcal{R}$ is then defined as $h(x) = \widehat{h}(\widetilde{x})$.

Cache efficiency. It is worth to notice that such construction allows us to store in cache all $|\Sigma|$-sized tables while the two $|\Psi|$-sized tables might overflow cache. However, these larger tables are accessed only once while evaluating tornado and they can be accessed in parallel.

Local uniformity theorem. Now we are ready to state an analogous of Theorem 4 for larger sets of selected keys. In what follows we use f, μ^f and $X^{f,h}$ as defined in Section 1.2.

Theorem 16. *Let $h = \widehat{h} \circ \widetilde{h} : \Sigma^c \to \mathcal{R}$ be a random tornado-mix tabulation hash function with d derived characters, the last two from Ψ, and select function f. If $\mu = \mu^f \leqslant |\Psi|/2$ then the derived selected keys $\widetilde{h}(X^{f,h})$ are linearly dependent with probability at most*

$$14\mu^3(3/|\Psi|)^2(3/|\Sigma|)^{d-1} + 1/2^{|\Sigma|/2} \,.$$

Proof. This proof works exactly as the proof of Theorem 4, except for a few slight differences. We limit ourselves to listing such small differences. In Section 3.2 we define Z as the smallest d-zero set among those d-zero sets minimizing the number of elements not in Q. This definition implies that there exists $x^* \in Z$ such that $Z \backslash \{x^*\}$ is d-independent. Moreover, either $Z \subseteq Q$ or we can choose $x^* \in Z \backslash Q$. In the original proof, we considered the alternating-cycle structure induced by the top level matchings M^*_{d-1} and M^*_d and traversed these cycles leaving x^* last. This ensured that the edges from M^*_d were always "surprising" in the sense that the probability of any such edge being realised by our random choice of h was $1/|\Sigma|$, even after conditioning on all previously discovered edges (these events were, indeed, independent). This observation allowed us to bound the probability of our obstruction being realised by h. In fact, we wanted our obstruction to be constituted by edges which realisations were independent, exluding the last edge. This is exactly what we did in Section 3.2, where all edges but the last edge $e_{d-1} \in M^*_{d-1}$ were realised by h independently.

In the current scenario, it is not obvious that the realisations of traversed edges from M^*_d are all independent and the first edge introducing a dependence belongs to M^*_{d-1}. Here, instead, we

traverse the alternating-cycle graph until we find a prefix $W = \{x_1 \dots x_w\}$ such that either (i) M^*_{d-1} restricted to W or (ii) M^*_d restricted to W is $(d-2)$-dependent. Case (i) is identical to the case already analysed, but in case (ii) we need some small changes, essentially swapping the roles of M_d and M_{d-1}.

To understand the interplay between the two cases, note that every time we add a vertex x_w, we add an edge to $M^*_{d-1}|_W$ if w is odd, and to $M^*_d|_W$ if w is even. Case (i) can only happen if w is even while Case (ii) can only happen if w is odd. If we get to $x_w = x^*$, then we have have $W = Z$ and then we are in case (i) since Z is even. Thus, if we end in case (ii), then $x^* \notin W$.

We now consider the slightly reduced traversal sequence where we simply drop the first vertex in the last cycle considered in the traversal. The point is that this vertex was not matched in $M^*_d|W$, but if we skip it then $M^*_d|W$ is a perfect $d-2$-dependent matching while $M^*_{d-1}|W$ is a maximal matching. Since we did not have x^* in W, we have $W \subseteq Z \backslash \{x^*\}$ implying full independence of the hash values over W. Thus, using this reduced traversal sequence, we have swapped the roles of the two top levels. Since we now have two cases, our union based probability bounds are doubled (increasing the leading constant from 7 to 14).

The rest of the analysis is unchanged except that two top levels use the different alphabet Ψ. This means Ψ replaces Σ in our bound (9) for the two top levels. This implies that our overall bound is multiplied by $|\Sigma|^2 / |\Psi|^2$. Combined with the doubling, we get an overall probability bound of

$$14|X|^3(3/|\Psi|)^2(3/|\Sigma|)^{d-1} + 1/2^{|\Sigma|/2} .$$

□

From Theorem 16, using Lemma 2, we derive the following analogous of Theorem 5 which was already stated in the introduction. Here we use the same notation as in Theorem 5, where $h^{(s)}$ are the selection bits and $h^{(t)}$ are the free bits.

Theorem 8. *Let $h = \widehat{h} \circ \widetilde{h} : \Sigma^c \to \mathcal{R}$ be a random tornado-mix tabulation hash function with d derived characters, the last two from Ψ, and an s-selector function f. If $\mu = \mu^f \leqslant |\Psi|/2$ then $h^{(t)}$ is fully random on $X^{f,h}$ with probability at least*

$$1 - 14\mu^3(3/|\Psi|)^2(3/|\Sigma|)^{d-1} - 1/2^{|\Sigma|/2} .$$

6 Upper Tail Chernoff

In this section, we show a Chernoff-style bound on the number of the selected keys $X^{f,h}$.

Lemma 6. *Let $h = \widehat{h} \circ \widetilde{h} : \Sigma^c \to \mathcal{R}$ be a random tornado tabulation hash function with d derived characters, query keys Q and selector function f. Let $\mathcal{I}_{X^{f,h}}$ denote the event that the set of derived selected key $\widetilde{h}(X^{f,h})$ is linearly independent. Then, for any $\delta > 0$, the set $X^{f,h}$ of selected keys satisfies the following:*

$$\Pr\left[\left|X^{f,h}\right| \geqslant (1+\delta)\cdot\mu^f \wedge \mathcal{I}_{X^{f,h}}\right] \leqslant \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^{\mu^f} .$$

Proof. We follow the proof of the Chernoff bound for the upper tail. Mainly, we let $\mathcal{J}_x$ denote the indicator random variable for whether the key x gets selected in $X^{f,h}$. Then $\left|X^{f,h}\right| = \sum_{x\in\Sigma^c} \mathcal{J}_x$.

For simplicity, we let $a = 1 + \delta$. Then for any $s > 0$:

$$\begin{aligned}
\Pr\Big[\Big|X^{f,h}\Big| \geqslant a \cdot \mu^f \wedge \mathcal{I}_{X^{f,h}}\Big] &= \Pr\Big[\Big|X^{f,h}\Big| \cdot [\mathcal{I}_{X^{f,h}}] \geqslant a \cdot \mu^f\Big] \\
&= \Pr\Big[e^{s|X^{f,h}|\cdot[\mathcal{I}_{X^{f,h}}]} \geqslant e^{sa\cdot\mu^f}\Big] \\
&\leqslant \frac{\mathrm{E}\Big[M_{|X^{f,h}|\cdot[\mathcal{I}_{X^{f,h}}]}(s)\Big]}{e^{sa\cdot\mu^f}},
\end{aligned}$$

where $M_{|X^{f,h}|\cdot[\mathcal{I}_{X^{f,h}}]}(s)$ is the moment generating function of the random variable $(\big|X^{f,h}\big| \cdot [\mathcal{I}_{X^{f,h}}])$, and is equal to:

$$M_{|X^{f,h}|\cdot[\mathcal{I}_{X^{f,h}}]}(s) = \mathrm{E}\Big[e^{s\cdot|X^{f,h}|\cdot[\mathcal{I}_{X^{f,h}}]}\Big] = \sum_{i=0}^{\infty} \frac{s^i}{i!} \cdot \mathrm{E}\left[\Big|X^{f,h}\Big|^i \cdot [\mathcal{I}_{X^{f,h}}]\right].$$

Since $\big|X^{f,h}\big| = \sum_{x\in\Sigma^c} \mathcal{J}_x$, each moment $\mathrm{E}\Big[\big|X^{f,h}\big|^i \cdot [\mathcal{I}_{X^{f,h}}]\Big]$ can be written as the sum of expectations of the form $\mathrm{E}[\prod_{x\in S} \mathcal{J}_x \cdot [\mathcal{I}_{X^{f,h}}]]$, where $S \subseteq \Sigma^c$ is a fixed subset of size $\leqslant i$. In general, the moment generating function can be written as a sum, with positive coefficients, of terms of the form $\mathrm{E}[\prod_{x\in S} \mathcal{J}_x \cdot [\mathcal{I}_{X^{f,h}}]]$ for some fixed subset S. Moreover, each such term amounts to

$$\mathrm{E}\left[\prod_{x\in S} \mathcal{J}_x \cdot [\mathcal{I}_{X^{f,h}}]\right] = \Pr\Big[S \subseteq X^{f,h} \wedge \mathcal{I}_{X^{f,h}}\Big].$$

We now condition on the hash values of the query keys $h|_Q$. For any set $S \subseteq \Sigma^c$, we let $\mathcal{I}_{S\cup Q}$ denote the event that the derived keys in $\widetilde{h}(S \cup Q)$ are linearly independent. Note that $\mathcal{I}_{X^{f,h}}$ being true implies that $\mathcal{I}_{S\cup Q}$ is true. Moreover, since $S \cup Q$ is a fixed (deterministic) set, the event $\mathcal{I}_{S\cup Q}$ only depends on the randomness of $\widetilde{h}$. We get the following:

$$\begin{aligned}
\Pr\Big[S \subseteq X^{f,h} \wedge \mathcal{I}_{X^{f,h}} \;\Big|\; h|_Q\Big] &\leqslant \Pr\Big[S \subseteq X^{f,h} \wedge \mathcal{I}_{S\cup Q} \;\Big|\; h|_Q\Big] \\
&= E\Big[\,[\mathcal{I}_{S\cup Q}] \cdot \Pr\left(S \subseteq X^{f,h} \;\middle|\; \widetilde{h}, h|_Q\right) \;\Big|\; h|_Q\Big] \\
&= E\Big[\prod_{x\in S} {\mathcal{J}^*}_x \;\Big|\; h|_Q\Big],
\end{aligned}$$

where $\{{\mathcal{J}^*}_x\}_{x\in X}$ denotes the indicator random variables for choosing to select the keys in S independently and uniformly at random when we fix the hash values of the query keys. This last step is due to the fact that the event if $\mathcal{I}_{S\cup Q}$ is true then, then $(h|_S, h|_Q)$ is fully random and it has the same distribution as $(h^*|_S, h|_Q)$, where h^* is a fully-random hash function. If $\mathcal{I}_{S\cup Q}$ is false, then the entire expression is 0. We can thus continue the proof of Chernoff's as if $\big|X^{f,h}\big|$ were a sum of independent random variables. The claim follows by noticing that, when $\mathcal{I}_{X^{f,h}}$ is true, we have that $E\left[X^{f,h}\right] \leqslant \mu^f$.

□

6.1 Upper Tail Chernoff for larger μ^f

We now consider the case in which we have a selector function for which $\mu^f > |\Sigma|/2$. In this case, even though we cannot guarantee that the set of derived selected keys is linearly independent whp, we show that, whp, its size still cannot be much larger than μ^f. This particular case will be useful in the analysis of linear probing from Section 7.

Lemma 17. *Let $h = \widehat{h} \circ \widetilde{h} : \Sigma^c \to \mathcal{R}$ be a random tornado tabulation hash function with d derived characters, query key Q with $|Q| < |\Sigma|/2$, and selector function f such that $\mu^f > |\Sigma|/2$. Then, for any $\delta > 0$, the set of selected derived keys $X^{f,h}$ satisfies the following:*

$$\Pr\left[\left|X^{f,h}\right| \geqslant (1+\delta)\cdot\mu^f\right] \leqslant 4\cdot\left(\frac{e^{\delta_0}}{(1+\delta_0)^{1+\delta_0}}\right)^{|\Sigma|/2} + 4\cdot \mathsf{DependenceProb}(|\Sigma|/2, d, \Sigma)\ ,$$

where

$$\delta_0 = \frac{\mu^f}{\mu^f - |Q|}\cdot\frac{|\Sigma|/2 - |Q|}{|\Sigma|/2}\cdot\delta \geqslant \left(1 - \frac{|Q|}{|\Sigma|/2}\right)\cdot\delta\ .$$

Proof. We modify the given selector function f to get another selector function f_p with the same set of query keys Q but with a much smaller μ^{f_p}. Selection according to f_p is done such that, once f selects a key in $\Sigma^c\backslash Q$, f_p further sub-selects it with some probability p. This sub-selection is done independently for every selected key. It follows that, for all $x \in \Sigma^c\backslash Q$, $p_x^{f_p} = p_x^f \cdot p$. Taking into account query keys, we also get that $\mu^{f_p} = (\mu^f - |Q|)\cdot p + |Q| = p\mu^f + (1-p)|Q|$.

Moreover, one can show that, as long as $\mu^{f_p} \leqslant |\Sigma|/2$, all our results about linear independence also hold for such sub-sampled select function. In particular, the only aspect of the proof of Theorem 4 that depends on the probabilities $p_x^{f_p}$ is the proof of Lemma 14. There, we invoke Lemma 9 to get an upper bound on the probability that the set $W\backslash\{x_w\}$ is selected, given that it is d-independent. Notice that, if, additionally, we sub-sample elements from $W\backslash\{x_w\}$ each independently with probability p, we obtain the same bounds as if we initially selected elements with probability $p_x^{f_p}$. Therefore, when $\mu^f > |\Sigma|/2$, we can pick any $p \leqslant (|\Sigma|/2 - |Q|)/(\mu^f - |Q|)$ to get that the set of derived keys for the sampled selection $\widetilde{X}^{f_p,h}$ is indeed linearly independent with probability at least $1 - \mathsf{DependenceProb}(\mu^{f_p}, d, \Sigma)$. We use $I_{X^{f_p,h}}$ to denote this event.

The next step is to notice that, conditioned on $\left|X^{f,h}\right| - |Q|$, the distribution of $\left|X^{f_p,h}\right| - |Q|$ is exactly the binomial distribution $B(\left|X^{f,h}\right| - |Q|, p)$. Then, for $p > 1/(\left|X^{f,h}\right| - |Q|)$, we have from [20] that

$$\Pr\left[\left|X^{f_p,h}\right| \geqslant \mathrm{E}\left[\left|X^{f_p,h}\right| \mid \left|X^{f,h}\right|\right] \mid \left|X^{f,h}\right|\right] > 1/4\ .$$

Therefore, for any $t > 0$:

$$\Pr\left[\left|X^{f_p,h}\right| \geqslant p\cdot t + (1-p)|Q| \;\middle|\; \left|X^{f,h}\right| \geqslant t\right] > 1/4\ .$$

We now use this to derive an upper bound on $\Pr\left(\left|X^{f,h}\right| \geqslant t\right)$ as such:

$$\begin{aligned}\Pr\left(\left|X^{f,h}\right| \geqslant t\right) &< 4 \cdot \Pr\left(\left|X^{f_p,h}\right| \geqslant p \cdot t + (1-p)\,|Q| \;\middle|\; \left|X^{f,h}\right| \geqslant t\right) \cdot \Pr\left(\left|X^{f,h}\right| \geqslant t\right) \\ &\leqslant 4 \cdot \Pr\left(\left|X^{f_p,h}\right| \geqslant p \cdot t + (1-p)\,|Q|\right) \\ &\leqslant 4 \cdot \Pr\left(\left|X^{f_p,h}\right| \geqslant p \cdot t + (1-p)\,|Q| \wedge I_{X^{f_p,h}}\right) + \\ &+ 4 \cdot \mathsf{DependenceProb}(\mu^{f_p}, d, |\Sigma|)\ .\end{aligned}$$

We now plug in $t = (1+\delta)\cdot\mu^f$, and get that $p \cdot t + (1-p)\,|Q| \geqslant (1+\delta_0)\cdot\mu^{f_p}$ for

$$\delta_0 \leqslant \frac{p\mu^f}{\mu^{f_p}} \cdot \delta\ .$$

We then invoke Lemma 6 to get that

$$\Pr\left(\left|X^{f_p,h}\right| \geqslant (1+\delta_0)\cdot\mu^{f_p} \wedge I_{X^{f_p,h}}\right) \leqslant \left(\frac{e_0^\delta}{(1+\delta_0)^{1+\delta_0}}\right)^{\mu^{f_p}}\ .$$

Finally, we instantiate $p = (|\Sigma|/2 - |Q|)/(\mu^f - |Q|)$ such that $\mu^{f_p} = |\Sigma|/2$. and notice that indeed, $p > 1/(\left|X^{f,h}\right| - |Q|)$ when $\left|X^{f,h}\right| \geqslant (1+\delta)\cdot\mu^f$ and $|Q| < |\Sigma|/2$. We notice that, in this case,

$$\delta_0 = \frac{\mu^f}{\mu^f - |Q|} \cdot \frac{|\Sigma|/2 - |Q|}{|\Sigma|/2} \cdot \delta \geqslant \left(1 - \frac{|Q|}{|\Sigma|/2}\right) \cdot \delta\ .$$

The argument follows.

□

7 Linear Probing with Tornado Tabulation

In this section we show how to formally apply our framework to obtain results on linear probing with tornado tabulation. We present the following main result comparing the performance of linear probing with tornado tabulation to that of linear probing using fully random hashing on a slightly larger keyset.

Theorem 18. *Let $S, S^* \subseteq \Sigma^c$ be sets of keys of size n and $n^* = (1+15\sqrt{\log(1/\delta)/|\Sigma|})n$, respectively, for some $\delta \in (0, 1/6)$. Let T, T^* be arrays of size m, a power of two. Now consider inserting the keys in S (S^*) into T (T^*) with linear probing using tornado tabulation (fully random hashing). Let X and X^* be the number of comparisons performed when inserting a new key x in each of T and T^* (i.e., $x \notin S \cup S^*$). Given the restrictions listed below there exists an event $\mathcal{E}$ with $\Pr(\mathcal{E}) \geqslant 1 - (1/|\Sigma| + 6\delta + 61\log n \cdot \mathsf{DependenceProb}(|\Sigma|/2, d, \Sigma))$ such that, conditioned on $\mathcal{E}$, X is stochastically dominated by X^*.*
Restrictions:

- $n/m \leqslant 4/5$
- $|\Sigma| \geqslant 2^{16}$

- $|\Sigma| \geqslant 30 \cdot \log n$
- $\sqrt{\log(1/\delta)/|\Sigma|} \leqslant 1/18$

From Theorem 18, it follows that linear probing using tornado tabulation achieves the same expected number of comparison as in the fully random setting, a proof is given in Section 7.4.

Corollary 19. *Setting $\delta = \Theta(1/|\Sigma|)$, $d \geqslant 5$, $\log n \leqslant o(|\Sigma|)$ in Theorem 18 we have*

$$\mathrm{E}[X] \leqslant \mathrm{E}[X^*] + o(1)\,.$$

The result of Corollary 19 is to be contrasted with previous work on practical implementations of linear probing. While Knuth's analysis serves as evidence of linear probing's efficieny in terms of the number of comparisons performed, the advantage of linear probing (over other hash table implementations) is that each sequential memory access is much faster than the random memory access we do for the first probe at $T[h(x)]$. How much faster depends on the computer system as does the cost of increasing the memory to reduce the load. Some experimental studies [7, 21, 34] have found linear probing to be the fastest hash table organization for moderate load factors (30-70%). If the load is higher, we could double the table size.

However, using experimental benchmarks to decide the hash table organization is only meaningful if the experiments are representative of the key sets on which linear probing would be employed. Since fully random hashing cannot be efficiently implemented, we might resort to weaker hash functions for which there could be inputs leading to much worse performance than in the benchmarks. The sensitivity to a bad choice of the hash function led [21] to advice *against* linear probing for general-purpose use. Indeed, Mitzenmacher and Vadhan [27] have proved that 2-independent hashing performs as well as fully random hashing if the input has enough entropy. However, [36, 30] have shown that with the standard 2-independent linear hashing scheme, if the input is a dense set (or more generally, a dense subset of an arithmetic sequence), then linear probing becomes extremely unreliable and the expected probe length increases from Knuth's $\frac{1+1/\varepsilon^2}{2}$ to $\Omega(\log n)$, while the best known upper bound in this case is $n^{o(1)}$ [24].[8]

In a breakthrough result, Pagh, Pagh and Ružić [29] showed that if we use 5-independent hashing and the load gap $\varepsilon = \Omega(1)$, then the expected probe length is constant. Pătraşcu and Thorup [31] generalized this to an $O(1/\varepsilon^2)$ bound for arbitrary ε, and showed that this also holds for simple tabulation hashing. However, in both cases, the analysis hides unspecified large constants in the O-notation. Thus, with these hashing schemes, there could still be inputs for which linear probing performs, say, 10 times worse in expectation, and then we would be better off using chaining.

Our result is of a very different nature. We show that whp, for any given query key x, the probe length corresponding to $h(x)$ when we use tornado tabulation hashing is stochastically dominated by the probe length in a linear probing table that hashes slightly more keys but uses fully random hashing. In particular, this implies that whp, the expected probe length with tornado tabulation hashing is only a factor $1 + o(1)$ away from Knuth's $\frac{1+1/\varepsilon^2}{2}$. We get this result without having to revisit Knuth's analysis from [25], but simply because we know that we are almost as good as fully random hashing, in a local sense that is sufficient for bounding the probe length (see Section 7.1).

As a further consequence of our results, we get that any benchmarking with random keys that we do in order to set system parameters will be representative for all possible sets of input keys.

[8] We note that if we only know that the hash function is 2-independent, then the lower bound for the expected probe length is $\Omega(\sqrt{n})$ and this is tight. [36, 30]

Moreover, the fact that tornado tabulation hashing only needs locality to perform almost as well as fully random hashing means that our arguments also work for other variants of linear probing. For instance, ones where the maintenance of the hash table prioritizes the keys depending on when they were inserted, as first suggested in [4]. Examples of this include Robin hood hashing where keys with the lowest hash value come first [8] or time-reversed linear probing [34] where the latest arrival comes first. In all these cases, tornado tabulation hashing performs almost as well as with fully-random hashing.

7.1 Proof of Theorem 18

We let $\alpha = n/m$ denote the fill of the hash table and $\varepsilon = 1 - \alpha$. The basic combinatorial measure that we study and employ is the *run length*: If cells $T[a], T[a+1], \ldots, T[b-1]$ are all occupied by elements from S but both $T[a-1]$ and $T[b]$ are freem these $b-a$ cells are called a run of length $b-a$. Let $R(x, S)$ be the length of the run intersecting $T[h(x)]$ and note that $R(x, S) + 1$ is an upper bound on the number of comparisons needed to insert some element y into the table when y hashes to the same location as x.

Let Δ be the largest power of two such that $3\alpha\Delta + 1 \leqslant |\Sigma|/2$. The following lemma, proven in Section 7.3, gives an upper bound on the probability that $h(x)$ intersects a long run.

Lemma 20.

$$\Pr[R(x, S) \geqslant \Delta] \leqslant \frac{1}{|\Sigma|} + 60 \log n \cdot \mathsf{DependenceProb}(|\Sigma|/2, d, \Sigma)\,.$$

Let $\mathcal{A}$ be the event $(R(x, S) \geqslant \Delta)$. Assuming $\mathcal{A}$, there exists at least one unoccupied cell in table T between $T[h(x)-\Delta]$ and $T[h(x)]$ and likewise between $T[h(x)]$ and $T[h(x)+\Delta]$. Hence the insertion of x only depends on the distribution of the much smaller key-set $\big\{s \in S \,\big|\, |h(s) - h(x)| \leqslant \Delta\big\}$.

The second step of our proof bounds the probability that tornado tabulation behaves like a fully random hash function when restricted to this small set of keys. As Theorem 5 doesn't apply for arbitrary intervals we will instead cover the necessary interval with three dyadic intervals. Recall that a dyadic interval is an interval of the form $[j2^i, (j+1)2^i)$, where i, j are integers. In the following we will exclusively consider a number of dyadic intervals all of length Δ. Let I_C denote the dyadic interval that contains $h(x)$, and similarly let I_R and I_L denote the dyadic intervals to the left and right, respectively, of I_C. We further let X_C be the set of keys in S that hash into the interval I_C, i.e., $X_C = \{x \in S \mid h(x) \in I_C\}$, and similarly, X_R and X_L are the pre-image of h in I_R and I_L, respectively. Given $\mathcal{A}$, the distribution of X is completely determined by the distribution of the keys in $X_L \cup X_C \cup X_R$ and $h(x)$.

The expected size of each preimage is $\alpha\Delta$ and our choice of Δ thus allows us to apply Theorem 5 to all three intervals at once. Let $\mathcal{B}$ be the event that the keys hashing into these intervals are distributed independently:

Corollary 21. *With probability at least* $1 - \mathsf{DependenceProb}(|\Sigma|/2, d, \Sigma)$, $\tilde{h}(X_R \cup X_C \cup X_L \cup \{x\})$ *is linearly independent, such that* h *hashes the keys in* $X_R \cup X_C \cup X_L \cup \{x\}$ *independently and uniformly in their respective intervals.*

We now define the analogous terms in the fully random setting. We let I_C^* denote the dyadic interval in T^* that contains $h^*(x)$ and I_R^* and I_L^* the right and left neighboring dyadic intervals. Similarly, we let X_C^*, X_R^* and X_L^* denote their preimages under h^*. The following lemma compares the two experiments in terms of the sizes of these preimages, and is proven in Section 7.2.

Lemma 22. *Let $\mathcal{C}$ be the event $|X_L| \leqslant |X_L^*| \wedge |X_C| \leqslant |X_C^*| \wedge |X_R| \leqslant |X_R^*|$, then*

$$\Pr\big[\mathcal{B} \wedge \bar{\mathcal{C}}\big] \leqslant 6\delta$$

Let $\mathcal{C}$ be the event that each of the preimages X_i contain at most as many elements as the corresponding preimage X_i^*. Finally, define $\mathcal{E} = \mathcal{A} \cap \mathcal{B} \cap \mathcal{C}$. We will now present a coupling $\tilde{X}$ of X which, when conditioned on $\mathcal{E}$, satisfies $\tilde{X} \leqslant X^*$.

For every realization of $|X_L|$, $|X_C|$ and $|X_R|$, we consider the following random process: starting from an emtpy table of size m and using the fully random h^*, insert the first $|X_L|$ elements from X_L^*, then the first $|X_R|$ elements from X_R^* and finally, the first $|X_C|$ elements from X_C^* (we do not insert any more elements after this). Note that, conditioned on $\mathcal{C}$, we have that it is possible to choose such elements (i.e., $|X_R| \leqslant |X_R^*|$ etc.). Now let $\tilde{X}$ denote the number of comparisons performed when inserting x into the table at this point in time.

We now have that X (defined for the tornado tabulation) is identically distributed as $\tilde{X}$ (defined for a fully random hash function). This is because event $\mathcal{A}$ implies that the distribution of X only depends on X_R, X_C and X_L. Event $\mathcal{B}$ further implies that, on these intervals, h behaves like a fully random hash function. Now note that $\tilde{X} \leqslant X^*$, since we can continue the random process and add the remaining keys in S^* and this can only increase the number of comparisons required to insert x (i.e., "more is worse").

Left is to compute the total probability that any of our required events fail:

$$\begin{aligned}
\Pr\big[\bar{\mathcal{E}}\big] &= \Pr\big[\bar{\mathcal{A}} \vee \bar{\mathcal{B}} \vee \bar{\mathcal{C}}\big] \\
&= \Pr\big[(\bar{\mathcal{A}} \vee \bar{\mathcal{C}}) \wedge \mathcal{B}\big] + \Pr\big[\bar{\mathcal{B}}\big] \\
&\leqslant \Pr\big[\bar{\mathcal{A}}\big] + \Pr\big[\mathcal{B} \wedge \bar{\mathcal{C}}\big] + \Pr\big[\bar{\mathcal{B}}\big] \\
&\leqslant 1/|\Sigma| + 6\delta + 61 \cdot \log n \cdot \mathsf{DependenceProb}(|\Sigma|/2, d, \Sigma)\,.
\end{aligned}$$

This concludes the proof.

7.2 Proof of Lemma 22

As Δ is chosen to be the largest power of two such that $3\alpha\Delta \leqslant |\Sigma|/2$ we get $\frac{|\Sigma|}{12\alpha} \leqslant \Delta$. Let t be a constant, to be decided later. For each $i \in \{L, C, R\}$ let $\mathcal{E}_i$ be the event $(|X_i| \leqslant t)$ and $\mathcal{E}_i^*$ be the event $(t \leqslant |X_i^*|)$.

$$\begin{aligned}
\Pr\big[\mathcal{B} \wedge \bar{\mathcal{C}}\big] &= \Pr[\mathcal{B} \wedge \exists i \in \{L, C, R\} : |X_i| > |X_i^*|] \\
&\leqslant \Pr\big[\mathcal{B} \wedge \exists i \in \{L, C, R\} : \bar{\mathcal{E}}_i \vee \bar{\mathcal{E}}_i^*\big] \\
&\leqslant \sum_{i \in \{L,C,R\}} (\Pr[\mathcal{B} \wedge |X_i| > t] + \Pr[\mathcal{B} \wedge |X_i^*| < t]) \\
&\leqslant \sum_{i \in \{L,C,R\}} (\Pr[\mathcal{B} \wedge |X_i| > t] + \Pr[|X_i^*| < t])
\end{aligned}$$

Let $\mu = \mathrm{E}[|X_i|] = \Delta\alpha$, $k = \sqrt{3\log(1/\delta)/\mu}$ and $t = (1+k)\mu$. Applying the tail-bound of

Lemma 6 we have

$$\begin{aligned}\Pr[\mathcal{B} \wedge |X_i| > t] &= \Pr[\mathcal{B} \wedge |X_i| > (1+k)\mu] \\ &\leqslant \exp\left(-k^2\mu/3\right) \\ &= \delta\,.\end{aligned}$$

As $\mu = \alpha\Delta \geqslant |\Sigma|/12$ we have

$$\begin{aligned}k &= \sqrt{3\log(1/\delta)/(\alpha\Delta)} \\ &\leqslant \sqrt{36\log(1/\delta)/|\Sigma|} \\ &\leqslant 1/3\,.\end{aligned}$$

As $n^* = (1+15\sqrt{\log(1/\delta)/|\Sigma|})n \geqslant (1+2.5k)n$, $\mu^* = \mathrm{E}[|X_i^*|] \geqslant (1+2.5k)\mu$. Next, let $k^* = k\cdot\sqrt{2/3}$. Then

$$\begin{aligned}\mu^* \cdot (1-k^*) &\geqslant (1+2.5k)\cdot\mu\cdot(1-k^*) \\ &= (1+2.5k)\cdot\mu\cdot(1-\sqrt{2/3}\cdot k) \\ &\geqslant \mu\cdot(1+k) \\ &= t\,.\end{aligned}$$

Hence

$$\begin{aligned}\Pr[|X_i^*| < t] &\leqslant \Pr[|X_i^*| < (1-k^*)\mu^*] \\ &\leqslant \exp(-(k^*)^2\mu^*/2) \\ &= \exp(-k^2\mu^*/3) \\ &\leqslant \exp(-k^2\mu/3) \\ &= \delta\,.\end{aligned}$$

Summing over the six cases we see $\Pr[\mathcal{B} \wedge \exists i \in \{L, C, R\} : |X_i| > |X_i^*|] \leqslant 6\delta$.

7.3 Proof of Lemma 20

Our proof relies on the simple observation that if $T[a]$ through $T[b]$ are all occupied and the run *starts* in $T[a]$ (i.e. $T[a-1]$ is free which excludes the possibility of prior positions spilling over), then the preimage $h^{-1}([a,b]) = \{s \in S \mid h(x) \in [a,b]\}$ must have size at least $|b-a|$. It must also be the case that either (1) the preimage $h^{-1}([a+1,b])$ has size at least $(b-a)\cdot(1-\gamma)$ or (2) the preimage $h^{-1}([a,a])$ is of size at least $(b-a)\cdot\gamma$, for any parameter $\gamma \geqslant 0$.

We can generalize this to consider a run starting in any position $T[b]$ within some interval $b \in [a,c]$ which continues through $T[d]$, then either (1) $\left|h^{-1}([c,d])\right| \geqslant (d-c)\cdot(1-\gamma)$ or (2) $\left|h^{-1}([a,c])\right| \geqslant (d-c)\cdot\gamma$. We will refer to $[a,c]$ as the start-interval and to $[c,d]$ as the long interval.

Our strategy is to make both of these events unlikely by balancing the size of the start-interval with the number of keys needed to fill up the long interval. Larger difference $(d-c)$ allows for a larger start-interval. With a collection of roughly $\log_{1+\varepsilon/(6\alpha)} m$ such start-intervals we cover all possible starting poisitions before $T[h(x)-\Delta]$, ruling out the possibility that a run starting before

$T[h(x) - \Delta]$ reaches $T[h(x)]$. The same strategy applied once more rules out the possibility that the run intersecting $T[h(x)]$ will continue through $T[h(x) + \Delta]$.

For our proof we set $\gamma = \varepsilon/3$ where $\varepsilon = 1 - n/m$ is the fill gap of T. The first pair of intervals we consider is the long interval $I_0 = [h(x) - \Delta, h(x)]$ and the start-interval I'_0 of size $\Delta\varepsilon/(6\alpha)$ preceding I_0. Next follows the long interval $I_1 = I'_0 \cup I_0$ with start-interval I'_1 of length $|I_1|\,\varepsilon/(6\alpha)$ preceding it, and so forth. Let $\Delta_i = |I_i|$ and $\Delta'_i = |I'_i|$. Observe how $\Delta_i = \Delta \cdot (1 + \varepsilon/(6\alpha))^i$, bounding the number of needed interval-pairs at $\log_{1+\varepsilon/(6\alpha)}(m/\Delta)$.

Depending on the length of the interval being inspected we can apply either Lemma 6 or Lemma 17 to bound the probability that the preimage exceeds the given threshold. Taking the maximum of the two bounds simplifies the analysis. Letting X be the size of a preimage, $\mu = \mathrm{E}[X]$ and $\delta \leqslant 1$ we obtain

$$\Pr[X \geqslant (1+\delta)\mu] \leqslant \mathsf{DependenceProb}(|\Sigma|/2, d, \Sigma) + 4 \cdot \exp\left(-\delta^2 \cdot \min\left\{|\Sigma|/2, \mu\right\}/3\right) .$$

Let X_i be the size of the preimage of the long interval I_i of length Δ_i with $\mu_i = \alpha\Delta_i$. Then

$$\begin{aligned}
\Pr[X_i \geqslant (1-\varepsilon/3)\Delta_i] &= \Pr\left[X_i \geqslant \left(1 + \frac{2\varepsilon}{3\alpha}\right)\alpha\Delta_i\right] \\
&\leqslant \Pr\left[X_i \geqslant \left(1 + \frac{2\varepsilon}{3}\right)\mu_i\right] \\
&\leqslant \mathsf{DependenceProb}(|\Sigma|/2, d, \Sigma) + 4 \cdot \exp\left(-\left(\frac{2\varepsilon}{3}\right)^2 \cdot \frac{\min\left\{\mu_i, |\Sigma|/2\right\}}{3}\right)
\end{aligned}$$

Notice how the probability is non-increasing for increasing sizes of the intervals. Thus we bound each of the probabilites for a long interval exceeding its threshold by the probability obtained for I_0 with $\mu_0 = \alpha\Delta \leqslant |\Sigma|/2$.

For start-interval I'_i of length Δ'_i with $\mu'_i = \alpha\Delta'_i = \Delta_i \cdot \varepsilon/6$ we observe the same pattern

$$\begin{aligned}
\Pr\left[X'_i \geqslant \varepsilon/3\Delta_i\right] &= \Pr\left[X'_i \geqslant 2\mu'_i\right] \\
&\leqslant \mathsf{DependenceProb}(|\Sigma|/2, d, \Sigma) + 4 \cdot \exp\left(-\frac{\max\left\{\mu'_i, |\Sigma|/2\right\}}{3}\right) ,
\end{aligned}$$

where we can bound the probability that each start-interval is too large by the probability obtained for I'_0 with $\mu'_0 = \Delta\varepsilon/6$.

The probability that *any* of our intervals is too large is thus at most

$$2\log_{1+\varepsilon/(6\alpha)} m \cdot \left(\mathsf{DependenceProb}(|\Sigma|/2, d, \Sigma) + 4\exp\left(-4/27 \cdot \varepsilon^2\alpha\Delta\right)\right)$$

where use that $4/9 \cdot \varepsilon^2\alpha\Delta \leqslant \varepsilon\Delta/9 \leqslant \varepsilon\Delta/6$ as $\varepsilon + \alpha = 1$, hence the probability obtained for I_0 is larger than that for I'_0.

Let us rewrite this expression in terms of n and $|\Sigma|$, our main parameters.

$$\begin{aligned}
\log_{1+\varepsilon/(6\alpha)} m &= \log_{1+\varepsilon/(6\alpha)} n + \log_{1+\varepsilon/(6\alpha)}(1/\alpha) \\
&\leqslant \log n \cdot \log_{1+\varepsilon/(6\alpha)}(2) + 6 \\
&\leqslant \log n \cdot 6\alpha/\varepsilon + 6 \\
&\leqslant 30 \cdot \log n ,
\end{aligned}$$

using $\varepsilon \geqslant 1/5$. As $\alpha\Delta \geqslant |\Sigma|/12$, we get

$$\begin{aligned} 4\exp\left(-4/27 \cdot \varepsilon^2 \alpha\Delta\right) &\leqslant 4\exp\left(-4/27 \cdot \varepsilon^2 |\Sigma|/12\right) \\ &\leqslant 4\exp\left(-1/2025 \cdot |\Sigma|\right) \\ &\leqslant \frac{1}{2|\Sigma|^2} \end{aligned}$$

as $|\Sigma| \geqslant 2^{16}$. Assuming $30 \cdot \log n \leqslant |\Sigma|$ the total error-probability becomes

$$1/(2|\Sigma|) + 30 \cdot \log n \cdot \mathsf{DependenceProb}(|\Sigma|/2, d, \Sigma)\,.$$

Repeating the process once more to ensure that the run at $T[h(x)]$ doesn't continue past $T[h(x) + \Delta]$ doubles the error-probability and proves the lemma.

7.4 Proof of Corollary 19

To bound the expected number of comparisons we rely on the following lemma from [31] which gives strong concentration bounds for the runlength when applied to our simple tabulation $\widehat{h}$.

Lemma 23 (Corollary 3.2 in [31])**.** *For any $\gamma = O(1)$ and $\ell \leqslant n^{1/(3(c+d))}/\alpha$,*

$$\Pr[R(x,S) \geqslant \ell] \leqslant \begin{cases} 2e^{-\Omega(\ell\varepsilon^2)} + (\ell/m)^\gamma & \text{if } \alpha \geqslant 1/2 \\ \alpha^{\Omega(\ell)} + (\ell/m)^\gamma & \text{if } \alpha \leqslant 1/2 \end{cases}$$

where the constants hidden in O and Ω are functions of $c+d$, the size of the derived keys on which we apply simple tabulation.

In particular this implies that, for some $\ell = \Theta\left((\log n)/\varepsilon^2\right)$, we have that $R(q,S) \geqslant \ell$ with probability at most $1/n^{10}$. Let $\mathcal{E}$ be the event of stochastic dominance, as given by Theorem 18, and $\mathcal{A}$ the event $(R(x,S) \leqslant \ell)$. Then

$$\mathrm{E}[X] = \mathrm{E}[X \mid \mathcal{E}] \cdot \Pr[\mathcal{E}] + \mathrm{E}\big[X \mid \bar{\mathcal{E}} \wedge \mathcal{A}\big] \cdot \Pr\big[\bar{\mathcal{E}} \wedge \mathcal{A}\big] + \mathrm{E}\big[X \mid \bar{\mathcal{E}} \wedge \bar{\mathcal{A}}\big] \cdot \Pr\big[\bar{\mathcal{E}} \wedge \bar{\mathcal{A}}\big]\,.$$

First, observe

$$\begin{aligned} \mathrm{E}[X \mid \mathcal{E}] \cdot \Pr[\mathcal{E}] &= \sum_{i=1} \Pr[X \geqslant i \mid \mathcal{E}] \cdot \Pr[\mathcal{E}] \\ &\leqslant \sum_{i=1} \Pr[X^* \geqslant i \mid \mathcal{E}] \cdot \Pr[\mathcal{E}] \\ &= \sum_{i=1} \Pr[X^* \geqslant i \wedge \mathcal{E}] \\ &\leqslant \sum_{i=1} \Pr[X^* \geqslant i] \\ &= \mathrm{E}[X^*]\,. \end{aligned}$$

With $\delta = 1/|\Sigma|$ and $d \geqslant 5$, $\Pr\big[\bar{\mathcal{E}}\big] \leqslant 9/|\Sigma|$. Assuming $\mathcal{A}$, the next open cell of T is at most ℓ positions away,

$$\begin{aligned} \mathrm{E}\big[X \mid \bar{\mathcal{E}} \wedge \mathcal{A}\big] \cdot \Pr\big[\bar{\mathcal{E}} \wedge \mathcal{A}\big] &\leqslant \ell \cdot \Pr\big[\bar{\mathcal{E}}\big] \\ &\leqslant \Theta\left(\frac{\log n}{\varepsilon^2}\right) \cdot \frac{9}{|\Sigma|} \\ &\leqslant o(1)\,. \end{aligned}$$

Finally, no more than n comparisons will ever be necessary. Hence

$$\begin{aligned}\mathrm{E}\big[X \mid \bar{\mathcal{E}} \wedge \bar{\mathcal{A}}\big] \cdot \Pr\big[\bar{\mathcal{E}} \wedge \bar{\mathcal{A}}\big] &\leqslant n \cdot \Pr\big[\bar{\mathcal{A}}\big] \\ &\leqslant 1/n^9 \,.\end{aligned}$$

This gives the desired bound on $\mathrm{E}[X]$,

$$\mathrm{E}[X] \leqslant \mathrm{E}[X^*] + o(1) + 1/n^9 \,.$$

8 Lower Bound for Tornado Tabulation

In this section, we show that the probability obtained in Theorem 4 is tight up to constant factors. Specifically, we will prove the following:

Theorem 7. *Let $h = \widehat{h} \circ \widetilde{h} : \Sigma^c \to \mathcal{R}$ be a random tornado tabulation hash function with d derived characters. There exists a selector function f with $\mu^f \leqslant \Sigma/2$ such that the derived selected keys $\widetilde{h}(X^{f,h})$ are linearly dependent with probability at least $\Omega((3/|\Sigma|)^{d-2})$.*

Our strategy will mimic that in the proof of Theorem 4 and show that the set of derived selected keys will contain a zero-set with probability at least $\Theta((\mu^f)^3(3/|\Sigma|)^{d+1})$. We begin by establishing some initial general bounds. In the following, we define $\widetilde{h}' : \Sigma^c \to \Sigma^{c+d}$ to map keys in Σ^c to *simple* derived keys in Σ^{c+d} by applying the same functions as $\widetilde{h}$ except with $\widetilde{h}_0(\cdot) = 0$, i.e., for all $i > 1$, $\widetilde{h}'_{c+i} = \widetilde{h}_{c+i}$.

Definition 1. We say that a zero-set $Y \subseteq \Sigma^c$ *survives* d rounds of tornado tabulation if the set $\widetilde{h}'(Y) \subseteq \Sigma^{c+d}$ of its simple derived keys is also a zero-set.

We focus on zero-sets of size 4 and lower bound the probability that they survive successive rounds of tornado tabulation. We first define some terminology necessary to describe how each new derived character in $\widetilde{x}'$ behaves. Specifically, let $Y = \{x_1, x_2, x_3, x_4\}$ be a zero-set, for some fixed ordering of its keys. We distinguish between four types of positions $i \in \{1, \ldots, c\}$ as such: (1) position i is of Type A iff $x_1[i] = x_2[i]$ and $x_3[i] = x_4[i]$, (2) it is of Type B iff $x_1[i] = x_3[i]$, $x_2[i] = x_4[i]$, (3) it is of Type C iff $x_1[i] = x_4[i]$ and $x_2[i] = x_3[i]$ and, (4) it is of Type D iff $x_1[i] = x_2[i] = x_3[i] = x_4[i]$. We now prove that"

Lemma 24. *Let $Y \subseteq \Sigma^c$ be a zero-set with $|Y| = 4$. Then, for any $c \geqslant 2$, Y survives one round of tornado tabulation with probability $(3 - 2/|\Sigma|)/|\Sigma|$.*

Proof. Since the original keys in Y already form a zero-set, the set of simple derived keys $\widetilde{h}'(Y)$ is a zero-set iff the set of simple derived characters $\widetilde{h}'_{c+1}(Y)$ is a zero-set. Moreover, the cases in which $\widetilde{h}'_{c+1}(Y)$ is a zero-set can be classified based on the type of position $c+1$. Specifically, let $\mathcal{A}_{c+1}$ denote the event that position $c+1$ is of Type A, i.e., $\widetilde{h}'_{c+1}(x_1) = \widetilde{h}'_{c+1}(x_2)$ and $\widetilde{h}'_{c+1}(x_3) = \widetilde{h}'_{c+1}(x_4)$, and similarly for $\mathcal{B}_{c+1}$, $\mathcal{C}_{c+1}$, and $\mathcal{D}_{c+1}$. Then

$$\begin{aligned}\Pr\left(Y \text{ survives one round}\right) &= \Pr\left(\widetilde{h}'_{c+1}(Y) \text{ is a zero-set }\right) \\ &= \Pr\left(\mathcal{A}_{c+1} \vee \mathcal{B}_{c+1} \vee \mathcal{C}_{c+1}\right) \\ &= \Pr\left(\mathcal{A}_{c+1}\right) + \Pr\left(\mathcal{B}_{c+1}\right) + \Pr\left(\mathcal{C}_{c+1}\right) - \Pr\left(\mathcal{A}_{c+1} \wedge \mathcal{B}_{c+1}\right) - \\ &\Pr\left(\mathcal{A}_{c+1} \wedge \mathcal{C}_{c+1}\right) - \Pr\left(\mathcal{B}_{c+1} \wedge \mathcal{C}_{c+1}\right) + \Pr\left(\mathcal{A}_{c+1} \wedge \mathcal{B}_{c+1} \wedge \mathcal{C}_{c+1}\right) \\ &= 3\Pr\left(\mathcal{A}_{c+1}\right) - 2\Pr\left(\mathcal{D}_{c+1}\right) \,,\end{aligned}$$

where the last equality follows from the fact that the events $\mathcal{A}_{c+1}$, $\mathcal{B}_{c+1}$ and $\mathcal{C}_{c+1}$ are equivalent up to a permutation of the elements in Y, and the fact that the conjunction of any pair of events in $\mathcal{A}_{c+1}$, $\mathcal{B}_{c+1}$ and $\mathcal{C}_{c+1}$ implies $\mathcal{D}_{c+1}$, and vice-versa.

We now bound $\Pr(\mathcal{A}_{c+1})$ and $\Pr(\mathcal{D}_{c+1})$. Recall that, by definition, the simple derived character $\widetilde{h}'_{c+1}(x)$ is the output of a simple tabulation hash function applied to the key x. Specifically, for each $i \in \{1, \ldots, c\}$, let $T_i : \Sigma \to \Sigma$ denote a fully random function. Then

$$\widetilde{h}'_{c+1}(x) = T_1(x[1]) \oplus \ldots \oplus T_c(x[c]) \ .$$

Let I_A, I_B, I_C and I_D partition the set of positions in the original keys $\{1, \ldots, c\}$ based on their type, i.e., I_A consists of positions that are of Type A but not Type D, similarly for I_B and I_C, and finally I_D denotes the positions of Type D. We then define $T_A(x) = \oplus_{i \in I_A} T_i[x[i]]$ and similarly $T_B(x)$, $T_C(x)$, and $T_D(x)$. When $\mathcal{A}_{c+1}$ happens, we have that $\widetilde{h}'_{c+1}(x_1) = \widetilde{h}'_{c+1}(x_2)$, which is equivalent to

$$T_B(x_1) \oplus T_C(x_1) = T_B(x_2) \oplus T_C(x_2) \ ,$$

since $T_A(x_1) = T_A(x_2)$ and $T_D(x_1) = T_D(x_2)$ by definition. Similarly, $\widetilde{h}'_{c+1}(x_3) = \widetilde{h}'_{c+1}(x_4)$, is equivalent to

$$T_B(x_3) \oplus T_C(x_3) = T_B(x_4) \oplus T_C(x_4) \ .$$

Note that, by definition, $x_1[i] = x_3[i]$ and $x_2[i] = x_4[i]$ for all $i \in I_B$, and hence $T_B(x_1) = T_B(x_3)$ and $T_B(x_2) = T_B(x_4)$. Similarly, $T_C(x_1) = T_C(x_4)$ and $T_C(x_2) = T_C(x_3)$. Therefore, both equalities are equivalent to

$$T_B(x_1) \oplus T_C(x_1) \oplus T_B(x_2) \oplus T_C(x_2) = 0 \ .$$

Given that $x_1[i] \neq x_2[i]$ for all $i \in I_B \cup I_C$ and the T_i's are independent, we have that

$$\Pr[T_B(x_1) \oplus T_C(x_1) \oplus T_B(x_2) \oplus T_C(x_2) = 0] = 1/|\Sigma| \ .$$

In order to bound $\Pr(\mathcal{D}_{c+1})$, we first note that

$$\Pr(\mathcal{D}_{c+1}) = \Pr(\mathcal{A}_{c+1} \wedge \mathcal{B}_{c+1}) = \Pr(\mathcal{A}_{c+1}) \cdot \Pr(\mathcal{B}_{c+1} \mid \mathcal{A}_{c+1}) = 1/|\Sigma| \cdot \Pr(\mathcal{B}_{c+1} \mid \mathcal{A}_{c+1}) \ .$$

A similar argument as before shows that event $\mathcal{B}_{c+1}$ is equivalent to

$$T_A(x_1) \oplus T_C(x_1) \oplus T_A(x_3) \oplus T_C(x_3) = 0 \ .$$

Note, in particular, that the event $\mathcal{B}_{c+1}$ depends on positions in I_A and I_C, while the event $\mathcal{A}_{c+1}$ depends on positions in I_B and I_C. Moreover, it cannot be that both I_A and I_B are empty, since then we would not have a zero-set of size 4 (i.e., we would get that $x_1 = x_4$ and $x_2 = x_3$). Therefore, $I_B \cup I_C \neq I_A \cup I_C$ and the two events $\mathcal{A}_{c+1}$ and $\mathcal{B}_{c+1}$ are independent, and so $\Pr(\mathcal{D}_{c+1}) = 1/|\Sigma|^2$. The claim follows. □

As a corollary, we get the following:

Corollary 25. *For any $c \geqslant 2$, a zero-set $Y \subseteq \Sigma^c$ with $|Y| = 4$ survives d rounds of tornado tabulation with probability $((3 - 2/|\Sigma|)/|\Sigma|)^d$.*

Proof. We prove the claim by induction on d and note that the case in which $d = 1$ is covered in Lemma 24. Now assume that the statement is true for $d-1$. Recall that $\widetilde{x}'$ denotes the simple derived key and that $\widetilde{x}'[\leqslant c+d-1]$ denotes the first $c+d-1$ characters of $\widetilde{x}'$. By extension, let $\widetilde{Y}'$ denote the set of simple derived keys of Y and similarly, $\widetilde{Y}'[\leqslant c+d-1] = \{\widetilde{x}'[\leqslant c+d-1] \mid x \in Y\}$ and $\widetilde{Y}'[c+d] = \{\widetilde{x}'[c+d] \mid x \in Y\}$. Finally, let $\mathcal{E}_{d-1}$ and $\mathcal{E}_d$ denote the events that the set $\widetilde{Y}'$ and $\widetilde{Y}'[\leqslant c+d-1]$, respectively, are zero-sets. Then:

$$\begin{aligned}
\Pr\left(\mathcal{E}_d\right) &= \Pr\left(\mathcal{E}_{d-1} \text{ and } \widetilde{Y}'[c+d] \text{ is a zero-set}\right) \\
&= \Pr\left(\mathcal{E}_{d-1}\right) \cdot \Pr\left(\widetilde{Y}'[c+d] \text{ is a zero-set } \mid \mathcal{E}_{d-1}\right) \\
&= \left(\left(3 - 2/\left|\Sigma\right|\right)/\left|\Sigma\right|\right)^{d-1} \cdot \Pr\left(\widetilde{Y}'[c+d] \text{ is a zero-set } \mid \mathcal{E}_{d-1}\right) \ ,
\end{aligned}$$

where the last equality holds by the inductive hypothesis. To finish things up, we note that the event $\widetilde{Y}'[c+d]$ conditioned on $\mathcal{E}_{d-1}$ is equivalent to the set $\widetilde{Y}'[\leqslant c+d-1]$ surviving one round of tabulation hashing. Hence, it happens with probability $\left(3 - 2/\left|\Sigma\right|\right)/\left|\Sigma\right|$ and the claim follows. □

8.1 Proof of Theorem 7

The hard instance. Consider the set of keys $S = \{0,1\} \times \Sigma$ and note that $\widetilde{h}_0$ on this set induces a permutation of the characters in Σ. Specifically, every key of the form $0c$ for some $c \in \Sigma$ will be mapped to the element $0c'$, where $c' = \widetilde{h}_0(0) \oplus c$ and the mapping $c \to \widetilde{h}_0(0) \oplus c$ is a permutation. Similarly for keys $1c$. Therefore, we can assume without loss of generality that $\widetilde{h}_0(\cdot) = 0$ and get that:

$$\Pr\left(\widetilde{h}(X^{f,h}) \text{ is linearly dep.}\right) = \Pr\left(\exists \text{ a four-set } Y \subseteq X^{f,h} \text{ that survives } d \text{ rounds of tornado tab.}\right).$$

We then define the selector function to select a key x if $x \in S$ and the two leftmost output characters of $h(x)$ are both 0. Note then than that the probability that an $x \in S$ gets selected to $X^{f,h}$ is $1/4$ and hence, $\mu^f = |\Sigma/2|$. We now focus on zero-sets from S of size 4 and, for any $c_1, c_2 \in \Sigma$ with $c_1 < c_2$, we denote the zero-set of size four $\{0c_1, 1c_1, 0c_2, 1c_2\}$ by $Y(c_1, c_2)$. We then let $\mathcal{Y} = \{Y(c_1,c_2) \mid c_1 < c_2 \in \Sigma\}$ and let $\mathcal{E}_i(Y)$ denote the event that a zero-set Y survives i rounds of tornado tabulation. We lower bound the probability that $\widetilde{h}(X^{f,h})$ is linearly dependent by only focusing on zero-sets in $\mathcal{Y}$:

$$\begin{aligned}
\Pr\left(\widetilde{h}(X^{f,h}) \text{ is linearly dep.}\right) &\geqslant \Pr\left(\exists Y \in \mathcal{Y} \text{ s.t. } \mathcal{E}_d(Y) \wedge Y \subseteq X^{f,h}\right) \\
&\geqslant \Pr\left(\exists Y \in \mathcal{Y} \text{ s.t. } \mathcal{E}_d(Y) \wedge Y \subseteq X^{f,h}\right) \\
&\geqslant \Pr\left(\exists Y \in \mathcal{Y} \text{ s.t. } \mathcal{E}_d(Y)\right) \cdot \Pr\left(\text{a fixed } Y \in \mathcal{Y}, Y \subseteq X^{f,h} \;\middle|\; \mathcal{E}_d(Y)\right) \\
&\geqslant 1/4^3 \cdot \Pr\left(\exists Y \in \mathcal{Y} \text{ s.t. } \mathcal{E}_d(Y)\right) \ ,
\end{aligned}$$

where the last two inequalities above are due to the fact that the probability that some fixed set $Y \in \mathcal{Y}$ gets selected in $X^{f,h}$ given that it survived d rounds of tabulation hashing is the same across all sets in $\mathcal{Y}$ and, furthermore, it is exactly $1/4^3$.

Surviving zero-sets. We now employ Corollary 25 to lower bound the probability that some zero-set in $\mathcal{Y}$ survives d rounds of tornado tabulation. Note that we already have that the expected number of zero-sets in $\mathcal{Y}$ that survive d rounds of tornado tabulation is

$$\Theta(|\Sigma|^2 \cdot ((3 - 2/|\Sigma|)/|\Sigma|)^d) = \Omega((3/|\Sigma|)^{d-2}) ,$$

which exhibits the desired dependency on d. The challenge with turning this expectation into a probability is that the events of sets in $\mathcal{Y}$ surviving a round are not independent. To address this, we decompose the event of some set Y surviving d rounds of tornado tabulation into the event that some set survives the first two rounds of tornado tabulation and the event that this set also survives the remaining $d - 2$ rounds:

$$\begin{aligned}
\Pr\left(\widetilde{h}(X^{f,h}) \text{ is linearly dep.}\right) &\geqslant 1/4^3 \cdot \Pr\left(\exists Y \in \mathcal{Y} \text{ s.t. } \mathcal{E}_d(Y)\right) \\
&\geqslant 1/4^3 \cdot \Pr\left(\exists Y \in \mathcal{Y} \text{ s.t. } \mathcal{E}_2(Y)\right) \cdot \Pr\left(\mathcal{E}_{d-2}(\widetilde{h}(Y)[\leqslant c+2]) \;\middle|\; \mathcal{E}_2(Y)\right) \\
&= 1/4^3 \cdot ((3 - 2/|\Sigma|)/|\Sigma|)^{d-2} \cdot \Pr\left(\exists Y \in \mathcal{Y} \text{ s.t. } \mathcal{E}_2(Y)\right) .
\end{aligned}$$

To finish the argument and prove the main claim, we show the following:

Lemma 26. *With constant probability, at least one set in $\mathcal{Y}$ survives the first two rounds of tornado tabulation.*

Proof. The proof proceeds in two stages: first, we will argue that, with constant probability, $\Theta(|\Sigma|)$ of the sets in $\mathcal{Y}$ survive the first round of tornado tabulation. These sets will have a specific structure that guarantees that they then survive the second round of tornado tabulation independently.

Let $T_1^{(1)}, T_2^{(1)} : \Sigma \to \Sigma$ be the two fully random hash functions involved in computing the first derived character, and let $\mathcal{C}_1$ denote the event that $T_1^{(1)}(0) \neq T_1^{(1)}(1)$. Note that $\mathcal{C}_1$ happens with probability $1 - 1/|\Sigma|$. Conditioned on $\mathcal{C}_1$, all the sets in $\mathcal{Y}$ that survive have position 3 of Type B or C. For any $Y(c_1, c_2) \in \mathcal{Y}$, position 3 is of Type B if $T_2^{(1)}(c_1) = T_2^{(1)}(c_2)$ and of Type C if $T_2^{(1)}(c_2) = T_1^{(1)}(0) \oplus T_1^{(1)}(1) \oplus T_2^{(1)}(c_1)$. These events are mutually exclusive and each occurs with probability $1/|\Sigma|$.

We now show that, with constant probability, at least $\Theta(|\Sigma)|$ of sets in $\mathcal{Y}$ will survive and futher, have position 3 be of Type B. We model this as a balls-into-bins game in which there is a bin for each character $\alpha \in \Sigma$ and the characters in Σ hash into bins using $T_2^{(1)}$. We let N_α denote the number of characters $c \in \Sigma$ with $T_2^{(1)}(c_1) = \alpha$, i.e., the occupancy of the bin for α. We are interested in events in which $N_\alpha \geqslant 2$, because this implies that there exists at least one set $Y(c_1, c_2) \in \mathcal{Y}$ where $T_2^{(1)}(c_1) = T_2^{(1)}(c_2) = \alpha$. The probability that this occurs is:

$$\Pr(N_\alpha \geqslant 2) = 1 - \Pr(N_\alpha = 0) - \Pr(N_\alpha = 1) = 1 - (1 - 1/|\Sigma|)^{|\Sigma|} - (1 - 1/|\Sigma|)^{|\Sigma|-1} ,$$

since each character hashes independently and uniformly into the bins. Now, for each $\alpha \in \Sigma$, define I_α to be the indicator random variable for whether $N_\alpha \geqslant 2$ and let $I = \sum_{\alpha \in \Sigma} I_\alpha$. We will argue that $I = \Theta(\Sigma)$ with constant probability. First note that the random variables $\{I_\alpha\}_{\alpha \in \Sigma}$ are negatively associated since bin occupancies are negatively associated [17]. Let $\mu = \mathrm{E}(I)$ and note that $\mu \geqslant |\Sigma|/4$ for $|\Sigma| \geqslant 2$. As such, we can apply Chernoff's bound and get that:

$$\Pr\left(I \leqslant |\Sigma|/8\right) \leqslant \Pr\left(I \leqslant (1-1/2)\cdot\mu\right) \leqslant e^{-\mu/8} \leqslant 1/2\ ,$$

where the last inequality holds when $|\Sigma| \geqslant 23$.

Now let $\mathcal{Y}' \subset \mathcal{Y}$ denote the set of zero-sets constructed as such: we partition the bins into subsets of the form $\{\alpha, \alpha'\}$ where $\alpha \oplus \alpha' = T_1^{(1)}(0) \oplus T_1^{(1)}(1)$. The event that $I \geqslant |\Sigma|/8$ implies that there are at least $|\Sigma|/16$ such subsets where at least one bin, say α, has $N_\alpha \geqslant 2$. Now let $c_1 < c_2$ be two such characters that hash into the bin and add the zero-set $Y(c_1, c_2)$ to $\mathcal{Y}'$. Note that every subset of bins contributes at most one zero-set to $\mathcal{Y}'$. Furthermore, the sets in $\mathcal{Y}'$ have the following properties:

- for any two distinct sets $Y(c_1, c_2), Y(c_3, c_4) \in \mathcal{Y}'$, it holds that $\{c_1, c_2\} \cap \{c_3, c_4\} = \varnothing$, since the characters c_1, c_2 hash into a different bin from c_3, c_4,
- if we denote the derived characters of $Y(c_1, c_2)$ by $\widetilde{h}_3(0c_1) = x_1$ and $\widetilde{h}_3(1c_1) = x_2$ and similarly, the derived characters of $Y(c_3, c_4)$ by $\widetilde{h}_3(0c_3) = y_1$ and $\widetilde{h}_3(1c_3) = y_2$, we have further that $\{x_1, x_2\} \cap \{y_1, y_2\} = \varnothing$. This is due to the way the derived characters are computed: on one hand, $x_1 = T_1^{(1)}(0) \oplus T_2^{(1)}(c_1) \neq T_1^{(1)}(0) \oplus T_2^{(1)}(c_3) = y_1$ because $T_2^{(1)}(c_1) \neq T_2^{(1)}(c_3)$ and similarly for $x_2 \neq y_2$. On the other hand, $x_1 \neq y_2$ because otherwise we would get that $T_2^{(1)}(c_1) \oplus T_2^{(1)}(c_3) = T_1^{(1)}(0) \oplus T_1^{(1)}(1)$, which would contradict the fact that $Y(c_1, c_2)$ and $Y(c_3, c_4)$ were generated by different subsets of bins.

In this context, the events in which sets in $\mathcal{Y}'$ survive the second round of tornado tabulation are independent. Specifically, let $Y(c_1, c_2) \in \mathcal{Y}'$ be a set as before with derived characters $\widetilde{h}_3(0c_1) = \widetilde{h}_3(0c_2) = x_1$ and $\widetilde{h}_3(1c_1) = \widetilde{h}_3(1c_2) = x_2$. We distinguish between whether the newly derived character is of Type A, B, or C. To this end, let $T_1^{(2)}, T_2^{(2)}, T_3^{(2)} : \Sigma \to \Sigma$ be the fully random hash functions involved in its computation. Then position 4 is of Type A if

$$T_1^{(2)}(0) \oplus T_3^{(2)}(x_1) = T_1^{(2)}(1) \oplus T_3^{(2)}(x_2)\ . \tag{24}$$

Position 4 is of Type B if

$$T_2^{(2)}(c_1) = T_2^{(2)}(c_2)\ , \tag{25}$$

and of Type C if

$$T_1^{(2)}(0) \oplus T_2^{(2)}(c_1) \oplus T_3^{(2)}(x_1) = T_1^{(2)}(1) \oplus T_2^{(2)}(c_2) \oplus T_3^{(2)}(x_2)\ . \tag{26}$$

Similar conditions hold for some other $Y(c_3, c_4) \in \mathcal{Y}'$, and moreover, each of them depends on values that are chosen independently from the values in $Y(c_1, c_2)$. Specifically, the analogues of Equation (24) for $Y(c_3, c_4)$ depends on the lookup table values of y_1 and y_2, where y_1 and y_2 are the derived characters $\widetilde{h}_3(0c_3) = y_1$ and $\widetilde{h}_3(1c_3) = y_2$, respectively. As noted before, we know that $\{x_1, x_2\} \cap \{y_1, y_2\} = \varnothing$, and so the analogue of Equation (24) for $Y(c_3, c_4)$ is independent of Equations (24), (25), and (26). Similar arguments can be made for the other cases.

Now fix some instantiation Y' of $\mathcal{Y}'$ of size $|\Sigma|/16$ and let $X_{Y'}$ denote the number of sets in Y' that survive the second round of tornado tabulation. We know from Lemma 24 that each set in

Y' survives the second round of tornado tabulation with probability $(3 - 2/|\Sigma|)/|\Sigma| \geqslant 2.75/16$ for $|\Sigma| \geqslant 8$. Furthermore, each set survives this second round independently from the others. It follows from Chernoff's inequality that X is then tightly concentrated around its mean. In particular,

$$\begin{aligned}\Pr\left(X_{Y'} \leqslant 0.01 \cdot 1/8 \;\middle|\; \mathcal{C}_1\right) &\leqslant \Pr\left(X_{Y'} \leqslant (1 - 0.99) \cdot \mathrm{E}[X_{\mathcal{Y}'}] \;\middle|\; \mathcal{C}_1\right) \\ &\leqslant e^{-\mathrm{E}[X_{\mathcal{Y}'}] \cdot 0.99^2/2} \leqslant e^{-2.75 \cdot 0.99^2/32} \leqslant e^{-0.08}\,.\end{aligned}$$

For our purposes, let $X_{\mathcal{Y}'}$ denote the random variable that counts the number of sets in $\mathcal{Y}'$ that survive the second round of tabulation hashing. Conditioned on the fact that $|\mathcal{Y}'| \geqslant |\Sigma|/16$, we can always pick an instantiation Y' of $\mathcal{Y}'$ on which to use the above bound. We then get that:

$$\Pr\left(X_{\mathcal{Y}'} \geqslant 0.01/8 \;\middle|\; |\mathcal{Y}'| \geqslant |\Sigma|/16 \wedge \mathcal{C}_1\right) \geqslant 1 - e^{-0.08}\,.$$

To put it all together, let $I_{\mathcal{Y}}$ denote the event that there exists $Y(c_1, c_2) \in \mathcal{Y}$ such that $Z_2(c_1, c_2)$. Recall that we defined the event $\mathcal{C}_1$ to be that $T_1^{(1)}(0) \neq T_1^{(1)}(1)$. Then:

$$\begin{aligned}\Pr(I_{\mathcal{Y}}) &\geqslant \Pr(I_{\mathcal{Y}} \wedge \mathcal{C}_1) \\ &= \left(1 - \frac{1}{|\Sigma|}\right) \cdot \Pr\left(I_{\mathcal{Y}} \;\middle|\; \mathcal{C}_1\right) \\ &\geqslant \left(1 - \frac{1}{|\Sigma|}\right) \cdot \Pr\left(I_{\mathcal{Y}} \wedge |\mathcal{Y}'| \geqslant |\Sigma|/16 \;\middle|\; \mathcal{C}_1\right) \\ &\geqslant \left(1 - \frac{1}{|\Sigma|}\right) \cdot \frac{1}{2} \cdot \Pr\left(I_{\mathcal{Y}} \;\middle|\; |\mathcal{Y}'| \geqslant |\Sigma|/16 \wedge \mathcal{C}_1\right) \\ &\geqslant \left(1 - \frac{1}{|\Sigma|}\right) \cdot \frac{1}{2} \cdot \Pr\left(X_{\mathcal{Y}'} \geqslant 1 \;\middle|\; |\mathcal{Y}'| \geqslant |\Sigma|/16 \wedge \mathcal{C}_1\right) \\ &\geqslant \left(1 - \frac{1}{|\Sigma|}\right) \cdot \frac{1}{2} \cdot (1 - e^{-0.08})\,.\end{aligned}$$

□

References

[1] Aamand, A., Das, D., Kipouridis, E., Knudsen, J. B. T., Rasmussen, P. M. R., and Thorup, M. No repetition: Fast and reliable sampling with highly concentrated hashing. *Proc. VLDB Endow. 15*, 13 (2022), 3989–4001.

[2] Aamand, A., Knudsen, J. B. T., Knudsen, M. B. T., Rasmussen, P. M. R., and Thorup, M. Fast hashing with strong concentration bounds. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing* (2020), pp. 1265–1278.

[3] Aamand, A., Knudsen, J. B. T., and Thorup, M. Load balancing with dynamic set of balls and bins. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021* (2021), S. Khuller and V. V. Williams, Eds., ACM, pp. 1262–1275.

[4] Amble, O., and Knuth, D. E. Ordered hash tables. *The Computer Journal 17*, 2 (1974), 135–142.

[5] Arbitman, Y., Naor, M., and Segev, G. Backyard cuckoo hashing: Constant worst-case operations with a succinct representation. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA* (2010), IEEE Computer Society, pp. 787–796.

[6] Bender, M. A., Kuszmaul, B. C., and Kuszmaul, W. Linear probing revisited: Tombstones mark the demise of primary clustering. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022* (2021), IEEE, pp. 1171–1182.

[7] Black, J. R., Martel, C. U., and Qi, H. Graph and hashing algorithms for modern architectures: Design and performance. In *Proc. 2nd International Workshop on Algorithm Engineering (WAE)* (1998), pp. 37–48.

[8] Celis, P., Larson, P.-A., and Munro, J. I. Robin hood hashing. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)* (1985), IEEE, pp. 281–288.

[9] Christiani, T., Pagh, R., and Thorup, M. From independence to expansion and back again. To appear, 2015.

[10] Dahlgaard, S., Knudsen, M. B. T., Rotenberg, E., and Thorup, M. Hashing for statistics over k-partitions. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science* (2015), IEEE, pp. 1292–1310.

[11] Dahlgaard, S., Knudsen, M. B. T., and Thorup, M. Practical hash functions for similarity estimation and dimensionality reduction. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA* (2017), I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., pp. 6615–6625.

[12] Dietzfelbinger, M., and auf der Heide, F. M. A new universal class of hash functions and dynamic hashing in real time. In *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, UK, July 16-20, 1990, Proceedings* (1990), M. Paterson, Ed., vol. 443 of *Lecture Notes in Computer Science*, Springer, pp. 6–19.

[13] Dietzfelbinger, M., and Rink, M. Applications of a splitting trick. In *Proc. 36th International Colloquium on Automata, Languages and Programming (ICALP)* (2009), pp. 354–365.

[14] Dietzfelbinger, M., and Weidling, C. Balanced allocation and dictionaries with tightly packed constant size bins. *Theoretical Computer Science 380*, 1 (2007), 47–68. Automata, Languages and Programming.

[15] Dietzfelbinger, M., and Woelfel, P. Almost random graphs with simple hash functions. In *Proc. 25th ACM Symposium on Theory of Computing (STOC)* (2003), pp. 629–638.

[16] Dietzfelbinger, M., and Woelfel, P. Almost random graphs with simple hash functions. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 2003), STOC '03, Association for Computing Machinery, p. 629–638.

[17] Dubhashi, D., and Ranjan, D. Balls and bins: A study in negative dependence. *Random Structures & Algorithms 13*, 5 (1998), 99–124.

[18] Flajolet, P., Éric Fusy, Gandouet, O., and Meunier, F. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *In Analysis of Algorithms (AOFA)* (2007).

[19] Fotakis, D., Pagh, R., Sanders, P., and Spirakis, P. G. Space efficient hash tables with worst case constant access time. *Theory Comput. Syst. 38*, 2 (2005), 229–248.

[20] Greenberg, S., and Mohri, M. Tight lower bound on the probability of a binomial exceeding its expectation. *Statistics & Probability Letters 86* (2014), 91–98.

[21] Heileman, G. L., and Luo, W. How caching affects hashing. In *Proc. 7th Workshop on Algorithm Engineering and Experiments (ALENEX)* (2005), p. 141–154.

[22] Houen, J. B. T., and Thorup, M. Understanding the moments of tabulation hashing via chaoses. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France* (2022), M. Bojanczyk, E. Merelli, and D. P. Woodruff, Eds., vol. 229 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 74:1–74:19.

[23] Klassen, T. Q., and Woelfel, P. Independence of tabulation-based hash classes. In *Proc. 10th Latin American Theoretical Informatics (LATIN)* (2012), pp. 506–517.

[24] Knudsen, M. B. T. Linear hashing is awesome. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)* (2016), IEEE, pp. 345–352.

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

[26] Li, P., Owen, A. B., and Zhang, C.-H. One permutation hashing. In *Proc. 26thAdvances in Neural Information Processing Systems* (2012), pp. 3122–3130.

[27] Mitzenmacher, M., and Vadhan, S. P. Why simple hash functions work: exploiting the entropy in a data stream. In *Proc. 19th ACM/SIAM Symposium on Discrete Algorithms (SODA)* (2008), pp. 746–755.

[28] Pagh, A., and Pagh, R. Uniform hashing in constant time and optimal space. *SIAM J. Comput. 38*, 1 (2008), 85–96.

[29] Pagh, A., Pagh, R., and Ružić, M. Linear probing with constant independence. *SIAM Journal on Computing 39*, 3 (2009), 1107–1120. See also STOC'07.

[30] Pătraşcu, M., and Thorup, M. On the k-independence required by linear probing and minwise independence. In *Proc. 37th International Colloquium on Automata, Languages and Programming (ICALP)* (2010), pp. 715–726.

[31] 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.

[32] Pătraşcu, M., and Thorup, M. Twisted tabulation hashing. In *Proc. 24th ACM/SIAM Symposium on Discrete Algorithms (SODA)* (2013), pp. 209–228.

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

[34] Thorup, M. Timeouts with time-reversed linear probing. In *Proc. IEEE INFOCOM* (2011), pp. 166–170.

[35] Thorup, M. Simple tabulation, fast expanders, double tabulation, and high independence. In *FOCS* (2013), pp. 90–99.

[36] 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.

[37] Wegman, M. N., and Carter, L. New classes and applications of hash functions. *Journal of Computer and System Sciences 22*, 3 (1981), 265–279. See also FOCS'79.

[38] 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

Hashing for Sampling-Based Estimation

Hashing for Sampling-Based Estimation*

Anders Aamand[1], Ioana O. Bercea[2], Jakob Bæk Tejs Houen[3], Jonas Klausen[1], and Mikkel Thorup[1]

[1]University of Copenhagen {aa,jokl,mthorup}@di.ku.dk
[2]Kungliga Tekniska Högskolan bercea@kth.se
[3]Alipes ApS jakn@di.ku.dk

December 2, 2024

Abstract

Hash-based sampling and estimation are common themes in computing. Using hashing for sampling gives us the coordination needed to compare samples from different sets. Hashing is also used when we want to count distinct elements. The quality of the estimator for, say, the Jaccard similarity between two sets, depends on the concentration of the number of sampled elements from their intersection. Often we want to compare one query set against many stored sets to find one of the most similar sets, so we need strong concentration and low error-probability.

In this paper, we provide strong explicit concentration bounds for Tornado Tabulation hashing [Bercea, Beretta, Klausen, Houen, and Thorup, FOCS'23] which is a realistic constant time hashing scheme. Previous concentration bounds for fast hashing were off by orders of magnitude, in the sample size needed to guarantee the same concentration. The true power of our result appears when applied in the local uniformity framework by [Dahlgaard, Knudsen, Rotenberg, and Thorup, STOC'15].

*This work was supported by the VILLUM Foundation grants 54451 and 16582.

Contents

1 Introduction

In designing and analyzing randomized algorithms, a common assumption is that we have access to fully random hash functions. These ideal hash functions have beautiful theoretical properties which turn out to be incredibly powerful in obtaining simple and reliable algorithms with strong theoretical guarantees on their performance. Unfortunately, fully random hash functions cannot be implemented in practice, and instead they serve as the objects of aspiration when studying practical hashing schemes. In other words, they motivate the following high level goal.

Provide a simple and practical hashing scheme sharing the most powerful probabilistic properties with fully random hashing.

An ambitious approach towards this goal is a framework introduced by [DKRT15] which they use to solve a variety of problems that had thus far been out of reach for any realistic hashing scheme. This framework combines two properties of the hash function, concentration bounds for *selection* and a notion of *local uniformity*[1]. A hash function enjoying both of these properties, provides powerful probabilistic guarantees for a realm of applications. We consider the Tornado Tabulation hashing scheme from [BBK$^+$23], a realistic, constant-time, hashing scheme, and demonstrate that it fits this framework like a glove. For instance, our work shows how to implement the hashing underlying most hash-based sampling and estimation schemes. Below, we describe the framework from [DKRT15].

1.1 Local Uniformity and Concentrated Selection

The key point in the machinery of [DKRT15] is to combine concentration bounds for *selection* with a certain *local uniformity*. We proceed to describe these notions. Local uniformity relates to how the hash function h behaves on a subset of selected keys X from a large set of keys A with $|A| = n$. The selection of keys is done by looking at the binary representation of their hash values. In particular, the bits of the hash value are partitioned into *free* and *select* bits such that a key is selected if and only if its select bits match some fixed bitmask. As an example, one could consider selecting all the keys whose hash values are strictly smaller than 16. In this case, the select bits would be all but the rightmost 4 bits of the hash value and the bitmask would require that they all be 0. In this work, we will use the same bitmask for each key in A.

With t select bits, we expect to select $\mu = n/2^t$ keys from our set A. Now suppose $\mu \leqslant s/2$, where s is a space parameter (for now it suffices to know that the hashing scheme uses space $O(s)$ with the O-notation hiding a small constant). Within these parameters, local uniformity implies that, with high probability, the free bits of the selected keys will be fully random. That is, if we define h^f to be the function mapping keys to their free bits, then local uniformity would imply that h^f is fully random on X. The above statement might appear a bit cryptic, but for understanding it, it is useful think about the generation of the hash function in two phases. The first phase settles the select bits of all keys and hence decides the selected set of keys X. The second phase generates the free bits. The point now is that with high probability over the random process of phase one, the free bits generated in phase two will be fully random for the keys in X. Importantly, the selection is *not* known when implementing the hash function, but only a tool for the analysis. For example, if we want to count distinct elements, the number of select bits in the analysis will depend on the number of distinct elements in the input.

[1]The term *local uniformity* was coined later by [BBK$^+$23]

As observed in [DKRT15], combining concentration bounds on the number of selected keys with local uniformity provides a powerful analytical framework for getting theoretical guarantees as if we had used fully random hashing for several important applications. Roughly speaking, for many hashing-based sampling and estimation algorithms, it suffices to understand the distribution of keys hashing to a small local region of the full hash range. The region is defined by the bitmask on the select bits, so whether or not a key hashes to this region is determined by whether or not the select bits of the hash value matches the bitmask. To see the similarity to fully random hashing, we view the generation of the fully random function in the same two phases, first generating the select bits and then the free bits. If we have strong concentration on the number of selected keys, we select almost the same number as with the fully random hash function and the remaining free bits are fully random in both cases. Now when the hash function is used in an algorithmic application, the algorithmic behavior within the select keys could be quite complicated. Nonetheless, the local uniformity framework ensures that, in a black box manner, we retain the same guarantees as if the hashing had been fully random.

Naturally, the above framework is only as strong as its individual components. Weak concentration bounds in the selection step will affect any application. Similarly, in order to be useful, the hashing scheme should provide the local uniformity property with high probability for realistic parameters. In the following two Sections 1.1.1 and 1.1.2 we discuss progress and challenges in obtaining strong versions of local uniformity and concentrated selection. We further state our main result in Section 1.1.2

1.1.1 Local Uniformity

In their original paper, [DKRT15] considered local uniformity with *Mixed Tabulation* hashing. We will discuss tabulation-based hashing schemes in detail later. For the present section, it suffices to know that in these schemes, keys from the universe $[u]$ are viewed as bit strings of length $\lg u$ partitioned into a small number, c, of characters each consisting of $(\lg u)/c$ bits. Defining $s = u^{1/c}$, the hash functions are defined through a small constant number of independent and fully random look-up tables of size s. In particular, the total space usage becomes $O(s)$ where the O-notation hides a small constant.

Using $c + 2b + 2$ such look-up tables, [DKRT15] obtained local uniformity with probability at least

$$1 - \left(\frac{O(\log s)^c}{s}\right)^b .$$

Above, the O-notation hides constants that are exponential in c and b. For space $s \to \infty$, this is interesting, but for more reasonable values of s, the above error probability bound may be above 1.

Recently, [BBK+23] introduced Tornado Tabulation, coining the term local uniformity in passing. Using $c + b + 2$ tables of size s, they obtained local uniformity with the much more useful *explicit* probability bound of at least

$$1 - (24(3/s)^b + 1/2^{s/2}) . \tag{1}$$

Note that this is a quite strong guarantee. For example, if $s = 2^{16}$, we only need a few extra look-up tables before the error probability becomes extremely small. In fact, they proved that local uniformity holds even when the selection is done according to the hashes of additional fixed query keys (e.g., select keys that have the same rightmost 4 bits as some given query key q). They also

show that their analysis is essentially tight in the sense that the additive $(3/s)^b$ term is necessary. While minor improvements of the constant 24 might be possible, their result arguably provides us with a hashing scheme with a very satisfactory local uniformity guarantee. Naturally, the next question is whether it provides strong concentration bounds for selection.

1.1.2 Concentration for Selection

In getting concentration bounds on the number of selected keys, we again aspire to emulate the fully random setting. In this setting, the most basic tool we have available is the classic Chernoff bound which gives that if X is the set of selected keys, then for any $\delta \in [0, 1]$,

$$\Pr[|X - \mu| \geqslant \delta\mu] < 2\exp(-\mu\delta^2/3). \tag{2}$$

The concentration bounds we discuss with practical hashing schemes will have the form

$$\Pr[|X - \mu| \geqslant \delta\mu] < 2\exp(-\mu\delta^2/\mathcal{F}) + P. \tag{3}$$

We shall refer to $\mathcal{F}$ as the *exponent factor* and to P as the *added error probability*. The exponent factor $\mathcal{F}$ plays a major role in this paper and is our main measure of the quality of the concentration bound. To see why, let us for a moment ignore the additive error probability P. Then $\mathcal{F}$ becomes a linear factor in the expected number of keys we need to select in order to stay within a desired relative error δ. Flipping the argument on its head, if we want the concentration bound to hold with probability $1 - Q$, we obtain the relative error bound $\delta = \sqrt{\frac{\mathcal{F}\log(1/Q)}{\mu}}$. If $\mathcal{F}$ is large, the framework in [DKRT15] becomes weak since the corresponding fully random experiment has a very different number of keys. As we will see shortly, all past work on practical hashing with general concentration bounds have suffered from the same issue, namely that $\mathcal{F}$ is a large constant.

Surprisingly, as shown in [BBK+23], there is an 'automatic' way of deriving an upper tail Chernoff bound from the property of local uniformity. Namely, if the selection of X is carried out using Tornado Tabulation hashing, then

$$\Pr[|X| > (1 + \delta)\mu] < \exp(-\mu\delta^2/3) + 24(3/s)^b + 1/2^{s/2}\,. \tag{4}$$

Thus (so far) Tornado Tabulation provides the peculiar guarantee of local uniformity combined with upper tail bounds. This suffices for many hash table applications where we only worry about hashing too many keys in the same bucket and where our particular concern is the number of keys colliding with a given query key. Indeed, one of their main applications was hash tables with linear probing, for which they proved Tornado Tabulation hashing behaves similarly to fully random hashing. However, for statistical estimation problems and in many algorithmic applications, we need *concentration*, not just upper tail bounds. We next discuss past work and state-of-the-art in obtaining such (two-sided) concentration bounds.

Hashing with Strong Concentration Designing a practical hashing scheme with strong concentration is an important and well-studied problem within the field of hashing. In the independence framework of Wegman and Carter [WC81], we say that the hash function h is k-independent if every k keys are mapped independently and uniformly into the hash range. We know from [SSS95] that in this case, the number of selected keys is concentrated as

$$\Pr[|X - \mu| \geqslant \delta\mu] < 2\exp(-\mu\delta^2/3) + \exp(-k/2). \tag{5}$$

In particular, this implies that for some desired error probability P, we would need to set $k \geqslant 2\ln(1/P)$. We could implement k as a degree $k-1$ polynomial, but this gets slow when k is large, and in particular, this is super-constant when $P = o(1)$.

A completely different approach to obtaining strong concentration bounds is to use tabulation hashing as pioneered by Patrascu and Thorup [PT12]. They considered simple tabulation hashing, a scheme dating back to Zobrist [Zob70]. Given the space parameter s, where $s^c = u$, simple tabulation hashing uses c independent and fully random tables of size s and computes a hash value by doing c table lookups. With tables stored in fast cache, this is faster than computing a degree-2 polynomial. Patrascu and Thorup [PT12] proved the following Chernoff-style concentration bound using simple tabulation hashing. Assuming $\mu \leqslant n^{1/(2c)} \leqslant \sqrt{s}$, for any $\delta \leqslant 1$:

$$\Pr[|X - \mu| \geqslant \delta\mu] < 2\exp(-\mu\delta^2/\mathcal{F}) + 1/n^\gamma. \tag{6}$$

where $\mathcal{F}$ depends exponentially on c and γ. In the Chernoff bound for fully random hashing (2), we had exponent factor 3 and no added error probability.

Getting better concentration bounds has been a main target for research in tabulation-based hashing [PT13, AKK+20, HT22]. For simple tabulation, [AKK+20] removed the requirement that $\mu \leqslant n^{1/(2c)}$ but had an added error probability of np^γ, so the sampling probability p had to be polynomially small. They also introduced tabulation-permutation hashing, which roughly doubled the number of tables, but removed the restriction on μ and reduced the added error probability to $1/u^\gamma$. That is, for any $\delta \leqslant 1$

$$\Pr[|X - \mu| \geqslant \delta\mu] < 2\exp(-\mu\delta^2/\mathcal{F}) + 1/u^\gamma. \tag{7}$$

The same bound was achieved for mixed tabulation in [HT22], which further described the dependence of $\mathcal{F}$ on c and γ as $\mathcal{F} = (c^2\gamma\,\mathcal{C})^c$, where $\mathcal{C}$ is a large unspecified universal constant. The work of [BBK+23] provided no lower tail bound, but Tornado Tabulation inherits the two-sided concentration in (7) from [HT22].

Our Technical Contribution. In this paper, we provide strong explicit lower tail bounds for Tornado Tabulation Hashing with $c + b + 3$ tables of size s. With this final piece of the puzzle, we get a hashing scheme fitting the powerful framework in [DKRT15] with explicit bounds. Below, X is still the set of selected keys as described above and $\mu = \mathrm{E}[X]$.

Theorem 1. *For any $b \geqslant 1$ and $c \leqslant \ln s$, if $s \geqslant 2^{16} \cdot b^2$, and $\mu \in [s/4, s/2]$. For any $\delta > 0$,*

$$\Pr[|X| < (1-\delta)\mu] < 3\exp\left(\frac{-\delta^2\mu}{7}\right) + (c+b+1)\ln(s) \cdot \left(49\left(\frac{3}{s}\right)^b + 3\left(\frac{1}{2}\right)^{s/2}\right). \tag{8}$$

We note again that the added error probability drops *rapidly*, even with a small choice of b. The proof of this theorem appears in Section 7.2. The concrete value of $\mathcal{F} = 7$ is an artifact of our analysis, and it is likely that a more careful argument will show that $\mathcal{F}$ is even closer to the 3 in (2). Additionally, we will see in the next section that the result can be bootstrapped to give $\mathcal{F} = 3$ at the cost of a constant blow-up in space. The proof of our lower tail bound requires significantly more work than the proof of the upper tail bound in (4) and several completely new ideas. The fundamental challenge is that in contrast to upper tail bounds, lower tail bounds must argue about the probability distribution of the keys that are *not selected*. Namely, standard proofs of the

Chernoff upper-bound use the Taylor expansion, of which each term represents the probability that some fixed set of keys is selected. Thanks to local uniformity, the selection of these fixed keys can be viewed as fully random, so bounding these probabilities is straightforward. In contrast, for the lower tail, the hash values of non-selected keys are far from fully random.

1.1.3 Threshold Sampling

To illustrate the power of the framework of [DKRT15] combined with strong guarantees on local uniformity and concentration of selection, we here apply it to the simple but fundamental algorithmic primitive *threshold sampling*. For threshold sampling, we view the hash values as numbers in $[0, 1)$, and given some sampling probability $p \in [0, 1]$, we sample a key $x \in A$ if $h(x) < p$. We would like the set of sampled keys X to accurately represent A in the sense that $|X|/p$ is a reliable estimator for $|A|$.

To apply the framework of [DKRT15], we require that $\mu = \mathrm{E}[X] = p|A| \ll s$. For the analysis we pick the smallest t such that $n/2^t \leqslant s/2$. We then use the t most significant bits as select bits, asking for all of them to be zero. It follows that the sampled keys are all selected, and we can view the threshold sampling as a supsample. By local uniformity, this subsample is fully random and we can apply the classic Chernoff bound (2). When s is large compared to μ, the deviation of Theorem 1 diminishes in comparison to the deviation of the fully random threshold sampling and this allows us to bootstrap the theorem to achieve $\mathcal{F} = 3$. As in the previous section, in the theorem below, we again consider Tornado Tabulation hashing with $c + b + 3$ look-up tables of size s.

Theorem 2. *Let $h : [u] \to [2^l]$ be a Tornado Tabulation hash function with $s \geqslant 2^{16}b^2$ and $c \leqslant \ln(s)$, A a set of keys, and $X = \{x \in A \mid h(x) < p\}$ for some $p \in [2^l]$. Suppose that $\mu = \mathrm{E}[X] \leqslant s/278$. Then for any $\delta < 1$, it holds that*

$$\Pr[||X| - \mu| > (1 + \delta)\mu] < 5 \exp\left(\frac{-\delta^2\mu}{3}\right) + (c + b + 2)\ln(s) \cdot \left(49\left(\frac{3}{s}\right)^b + 3\left(\frac{1}{2}\right)^{s/2}\right).$$

The proof of this theorem is given in Section 7.3. Comparing to the concentration bounds provided by past work (discussed in the previous section), our new exponent factor $\mathcal{F} = 3$ is smaller by several orders of magnitude. Recall that the bounds in [AKK+20, HT22] had $\mathcal{F} = (c^2\gamma\mathcal{C})^c$ where $\mathcal{C}$ is a large unspecified constant and c is such that $s^c = u$.

The requirement that $s \geqslant \max(2^{16}b^2, 278\mu)$ may seem disappointing but for large-scale applications it is not a big concern. The point is that we think of the Tornado Tabulation hash function h as a single *central* hash function used in the construction of millions of sketches. Storing each sketch requires space $\Omega(\mu)$, so the space used for storing the hash function is insignificant. The setting with many sketches also emphasizes the importance of having high probability bounds since we can then use a union bound to prove that the sketches all behave well simultaneously. Additionally, note that when $s \geqslant \max(2^{16}b^2, 278\mu)$, then the added error probability decays very quickly with increasing b, and we thus require fewer look-up tables for the hash function for a desired added error probability. Finally, the constant 278 is again an artifact of the analysis which could likely be reduced significantly.

1.1.4 Relationship to Highly Independent Hashing

A natural question is how well the classic k-independence framework by Wegman and Carter [WC81] fits into the local uniformity framework by [DKRT15]. First of all, it is clear that to obtain the property of local uniformity, we need space at least $s/2$ for the hash function. Indeed, in expectation $s/2$ keys will be fully random over the free bits. In particular, to employ the k-independence scheme, we would need $k \geqslant s/2$. If we implement the hash function as a degree $k-1$ polynomial, this becomes prohibitively slow. As an alternative, we can use the highly independent hashing introduced by Siegel [Sie04]. In this setting, Thorup's [Tho13b] double tabulation provides a simpler and more efficient implementation of such highly independent hashing schemes. Unfortunately, with space $O(s)$, the independence achieved by this construction is $O(s^{1/(5c)})$ which far from suffices for local uniformity. Moreover, with realistic parameters in the probabilistic guarantees, double tabulation is too slow for practical applications (see the discussions and experiments in [AKK$^+$20]).

Even with a hypothetical fast highly independent hashing scheme at hand, we would run into further issues. First, to fit the local uniformity framework, we would need two *independent* such hash functions, one for the select bits and one for the free bits. This is a bigger issue than it may seem, since the select bits appear only in the analysis and are not chosen by the algorithm design. In fact, for all of the applications in Section 2, the select bits depend on the input and for some of the analyses of these applications, we need to apply the local uniformity framework over several different choices of select bits.

Finally, one may ask if the locally uniformity framework is *necessary*. For instance, for the threshold sampling in Section 1.1.3, it suffices to use a $2\ln(1/P)$-independent hash function to get additive error probability P in eq. (5). There are two points to make in regards to this. First, in order to obtain high probability error bounds, say $n^{-\gamma}$, we must have $k = 2\gamma\ln(n)$ and both k-independent hashing as well as Thorup's construction [Tho13b] would be prohibitively slow. Secondly, for some of the applications we will discuss in Section 2 (e.g., the important Vector-k Sample), independence below k has no proven guarantees.

1.2 Roadmap of the Paper

In Section 2, we discuss hash based sampling and estimation schemes and how they fit in the local uniformity framework. In Section 3, we present the necessary preliminaries on Tornado Tabulation hashing. In Section 4, we discuss our main technical contribution and steps we need for the proof of Theorem 1. We will include a second roadmap by the end of Section 4 for an overview of where we take these steps.

2 Applications to Hash-Based Sampling and Estimation

Below in Section 2.1 we discuss different types of hash-based sampling and estimation schemes, starting from the most basic, and moving to those with the highest demand on the hash function. In each case, we first assume that the hash function h is fully random. Having seen all these applications of full randomness, we then argue in Section 2.2, in a black-box fashion, that Tornado Tabulation hashing performs almost as well as a fully random hash function.

2.1 Hash-Based Sampling and Estimation Schemes

Our starting point is the fundamental threshold sampling of Section 1.1.3. We review it again here.

Threshold Sampling. The most primitive form of hash-based sampling and estimation takes a threshold probability $p \in [0,1]$ and samples a key x if $h(x) < p$. For any key set $A \subseteq [u]$, let $S_p(A)$ be keys sampled from A. With $X = |S_p(A)|$ and $\mu = \mathrm{E}[X] = |A|p$, by standard Chernoff bounds, for $\delta \leqslant 1$, we have that

$$\Pr[|X - \mu| \geqslant \delta\mu] < 2\exp(-\mu\delta^2/3). \tag{9}$$

Thus, $X/p = |S_p(A)|/p$ is a strongly concentrated estimator of $|A|$. One advantage to storing $S_p(A)$ is that $S_p(A)$ can then be used to estimate the intersection size $|A \cap B|$ as $|S_p(A) \cap B|/p$, given any other set B. Having exponential concentration in particular is critical if we want low error probability bounds for a union bound over many events. For example, if we store $S_p(A_i)$ for many sets A_i and want to estimate the maximal intersection size with B, then it is important that none of the individual intersection estimates are too large. So far, however, we have not seen the advantage of hash-based sampling over independent sampling.

Benefits of Hash-Based or Coordinated Sampling. There are two main benefits to using hashing to coordinate the sampling. One is if the set of keys appear in a stream where a single key may appear multiple times. Then we can easily maintain a sample of the distinct keys. Another benefit, more critical to this paper, is that if we have sampled from two sets A and B, then we can compute the sample of their union $A \cup B$. For threshold sampling, this is done simply as $S_p(A \cup B) = S_p(A) \cup S_p(B)$, and likewise for the intersection as $S_p(A \cap B) = S_p(A) \cap S_p(B)$. Note that if we had instead sampled independently from A and B, then keys from the intersection would be too likely to be included in $S_p(A) \cup S_p(B)$ and less likely to be included in $S_p(A) \cap S_p(B)$.

Bottom-k Sampling and Order Statistics on Hash Values. With fully random hashing, the hash values from a set A are just a uniformly distributed set $h(A)$ of $|A|$ hash values from $(0,1)$. Let $h_{(1)}(A), \ldots, h_{(|A|)}(A)$ denote these hash values in sorted order. Assuming $k \leqslant |A|$, we know from order statistics (see, e.g., [Dav81]) that $\mathrm{E}[1/h_{(k+1)}(A)] = |A|/k$, so we can use $k/h_{(k+1)}(A)$ as an estimator for $|A|$. By definition, $p > h_{(k+1)}(A) \iff |S_p(A)| > k$. Therefore

$$k/h_{(k+1)}(A) < (1-\delta)|A| \iff k/((1-\delta)|A|) < h_{(k+1)}(A) \iff |S_{k/((1-\delta)|A|)}| > k.$$

Here $\mathrm{E}[|S_{k/((1-\delta)|A|)}| = k/(1-\delta)$, so lower tail bounds for $|S_p|$ with $p = k/((1-\delta)|A|)$ imply similar lower tail bounds for $k/h_{(k+1)}(A)$, and likewise for upper tail bounds. This argument for concentration of $1/h_{(k+1)}$ around $|A|/k$ is essentially taken from [BJK$^+$02], except that they use $1/h_{(k)}$ which is slightly off in the sense that its mean is $|A|/(k-1)$.

As in [CK07], we can also define the bottom-k sample of A as the subset $S^k(A)$ with the k smallest hash values. Together with $S^k(A)$, we can store $p^k(A) = h_{(k+1)}$, and then $S^k(A) = S_{p^k(A)}(S)$. Note that if we also have the bottom-k sample of a set B, then we can easily create the bottom-k sample for their union as $S^k(A \cup B) = S^k(S^k(A) \cup S^k(B))$. Note also that the bottom-1 sample is identical to Broder's famous MinHash [Bro97, BCFM00].

Frequency and Similarity. Based on the above observations, we now discuss a very powerful analysis for frequency and similarity estimation assuming sampling based on a fully random hash function. Very generally, given a set A, we assume that the sampling process is given the set of (distinct) hash values $h(A)$ and selects a subset Y of these hash values. It then returns the set of keys $S = \{x \in A \mid h(x) \in Y\}$. For threshold sampling, the selected hash values are those which hash below the threshold sampling probability, and for bottom-k it is the k smallest hash values that are selected.

With fully random hashing, for a given set A and a given set $h(A)$ of hash values, the set S is a fully-random sample without replacement from A. As a consequence, if B is a subset of A, then the frequency of B can be estimated as $|B \cap S|/|S|$. For a given sample size $|S|$, this estimator is the sum of negatively correlated 0-1 variables (does each sample belong to B or not), and all the standard Chernoff bounds, e.g., (9) hold in this case. For bottom-k samples, for $B \subseteq A$, we can use $|B \cap S^k(A)|/h^{(k+1)}(A)$ to estimate $|B|$. This estimator is unbiased as proved in [CK07], and it is concentrated thanks to the above concentrations of $|B \cap S^k(A)|$ and $1/h^{(k+1)}(A)$.

We are pointing out this analysis because we could do something more lossy using a union bound over different concentration bounds, as described in [Tho13a]. Assuming fully random hashing, we get the clean arguments presented above using the fact that the samples are independent. In a black-box fashion, we are going to argue that Tornado Tabulation hashing is similar to fully random hashing for frequency estimation, hence the above type of reasoning applies.

k-Partition-Min and Distinct Elements. We now discuss a very powerful and efficient way of creating sketches based on k-partitions. We use the first $\log k$ bits of the hash value to partition the keys between k buckets. We refer to these $\log k$ bits of the hash values as the bucket index, and the remaining bits as the local hash value. The idea is to look at the smallest local hash value within each bucket separately. We generally refer to this approach as k-Partition-Min, and it dates back at least to Flajolet and Martin [FM85] who used it for estimating the number of distinct elements. The more recent popular HyperLogLog algorithm [FEFGM07] is a compressed version, in that it only stores the number of leading zeros in the smallest local hash value.

The HyperLogLog sketch is very easy to maintain and update. When a new key comes to the bucket, we just have to check if it has more leading zeros than the current coordinate. This is faster than using a bottom-k approach, where we would need to keep a hash table over the sampled keys in order to check if the incoming key is new or a repeat. Likewise, given the HyperLogLog sketches from two sets, it is easy to construct the sketch of their union: for each bucket, we just have to find the maximal number of leading zeros from the two input sets.

Computing the estimate of the number of distinct elements from the HyperLogLog sketch is complicated and the analysis is involved even if we assume fully random hashing (see [FEFGM07]). Luckily, we will be able to claim that Tornado Tabulation hashing performs similarly in a black-box fashion, without needing to understand the details of the estimator. All we need to know is that it increases monotonically in each coordinate in the HyperLogLog sketch. Indeed, with a fixed hash function, it is clear that the coordinates of the HyperLogLog sketch can only increase as more keys are added, and hence so should the estimate of the number of distinct keys.

Vector-k Sample. Another powerful application of k-Partition-Min is when we store, for each bucket, the key with the smallest local hash value, i.e., the "min-key". For now, we assume that all buckets are non-empty. For a set A, we use $S^{\vec{k}}$ to denote the vector of these min-keys. This is the

One-Permutation Hashing scheme of [LOZ12]. If the hash function is fully-random, then the keys in $S^{\vec{k}}(A)$ are sampled uniformly, without replacement, just like the samples in the bottom-k sample $S^k(A)$. One important difference between the vector-k and bottom-k sample is that the vector-k sample is easier to update and maintain, the same as in the case of HyperLogLog: when a key is added, we only need to go to the bucket it hashes to and compare it with the current min-key. In contrast, with bottom-k, we would need to maintain a priority queue.

A more fundamental difference appears when we want to estimate the similarity of two sets A and B. Then we only have to compare $S^{\vec{k}}(A)$ and $S^{\vec{k}}(B)$ coordinate-wise: the Jaccard similarity is estimated as $\sum_{i=0}^{k-1}\left[S^{\vec{k}}(A)[i] = S^{\vec{k}}(B)[i]\right]/k$. Comparing coordinate-wise is necessary for some very important applications. As described in [LSMK11], it implies that we can estimate the similarity between sets as a dot-product and use this in Support Vector Machines (SVM) in Machine Learning. To get a standard bit-wise dot-product, [LSMK11] suggest that we hash the min-key in each bucket uniformly into $\{\texttt{01}, \texttt{10}\}$ (we could earmark the least significant bit of the hash value of the min-key for this purpose). If the min-keys in a coordinate are different, then with probability 1/2, they remain different, so dissimilarity is expected to be halved in this reduction. More importantly, more similar sets are expected to get larger dot-products, and this is all we need for the SVM applications. Mathematically, a cleaner alternative is to use the least significant bit to map the min-key in a bucket to $\left\{\frac{-1}{\sqrt{k}}, \frac{1}{\sqrt{k}}\right\}$. Now, in expectation, the dot-product is exactly the Jaccard similarity.

Having a vector sample is also important for Locality Sensitive Hashing (LSH) [IM98] as explained in detail in [DKT17]. The point is that using k-Partition-Min to compute a k-vector sample replaces the much slower approach to computing the MinHash [Bro97, BCFM00] with k independent hash functions, using the min-key with the ith hash function as the ith coordinate. With this $k\times$MinHash, we need to compute k hash values for every key while k-Partition-Min requires only one hash computation per key. This makes a big difference if k is large, e.g., $k = 10,000$ as suggested in [Li15].

A caveat of k-Partition-Min is that if bucket i is empty, then the ith sample is undefined. The "error" that some bucket is empty happens with probability at most P if $|A| \geqslant k\ln(k/P)$. It was shown in [DKT17] that we can fill the holes with error probability at most P by hashing indexed keys from $[j] \times A$ where $j|A| \geqslant \max\{|A|, k\ln(k/p)\}$. The total number of hash computations are then at most $\max\{|A|, 2k\ln(k/P)\}$, which is still much better than the $k|A|$ hash computations needed for $k\times$MinHash. The resulting vector-k sample becomes a mix of sampling with and without replacement. As proved in [DKT17], assuming fully random hashing, the number of samples from any subset of A will still be exponentially concentrated as in our Chernoff bound (9).

We note that in the applications of vector-k sample, we are typically comparing one set with many sets, to find the most similar set. Concentration is crucial to making sure that the most similar sample is not just similar due to noise.

The fundamental challenge in implementing k-Partition-Min with a realistic hash function is that we want the min-keys of different buckets to act as if they were independent except for being without replacement. In the q-independence paradigm of Wegman and Carter [WC81], it is not clear if any q less than $|A|$ would suffice. Nevertheless, Tornado Tabulation hashing will make all the applications work similarly to fully-random hashing.

We will now discuss how we can apply local uniformity and the concentration bounds to sampling and frequency estimation. Some of the applications are taken from [DKRT15], but we review them

here to underline the power of Theorem 1.

2.2 Applying Concentration of Selection and Local Uniformity

We next discuss the power of the local uniformity framework by [DKRT15] when employed with a hashing scheme with strong concentration for selection and local uniformity guarantees.

Concentration Bounds with Subsampling. We have already discussed the concentration bounds that we obtain for threshold sampling using the local uniformity framework in Section 1.1.3 and refer the reader to Theorem 2

Selecting Enough Keys for Applications. The original paper [DKRT15] did not introduce any new concentration bounds, but below we review how they used concentration bounds and local uniformity to analyze the more complex sampling and estimation.

The basic requirement is that the selected keys, with high probability, should contain all keys relevant to the final estimators (for threshold sampling, this was trivial). For instance, let us consider bottom-k sampling. As for threshold sampling, we will select keys based on 0s in their t most significant bits. If this leads to selection of more than k keys, then we know that we have selected all keys and hash values relevant to the estimators. If $s \geqslant 5k$ and t is the smallest value for which $\mu = n/2^t \leqslant s/2$, then $\mu \geqslant s/4 = 1.25k$. Thus, using our concentration bound in Theorem 1, we get that, with high probability, we select more than k keys.

For k-partition-min, the selection is a bit more subtle. We select keys based on 0s in the t most significant positions of their local hash value. We call such a hash value "locally small" regardless of the bucket index. With high probability, we want the smallest hash value in every bucket to be locally small. If we select more than $k \ln(k/P)$ locally small keys, then, with probability at least $1 - P$, we get one in each bucket. Thus we must pick $s \geqslant 5k \ln(k/P)$. To apply Theorem 1, we of course further have to assume that $s \geqslant 2^{16}b^2$.

The extra factor $\ln(k/P)$ for vector-k sampling may seem disappointing, but as explained in [DKT17], we do not know of any other reasonable way to implement vector-k sampling if we want exponential concentration bounds. We already mentioned the issue in using Wegman and Carter's independence paradigm [WC81]. Another tempting approach would be to use one hash function to split the keys between buckets, and then use an independent hash function for each bucket. However, the best implementation of MinHash uses tabulation [DT14], and then we would need k sets of tables yielding much worse space overall. Again our contribution is that we get an explicit and reasonable constant in the exponential concentration.

We finally note that while the Tornado Tabulation hash function may dominate the space of the streaming algorithm producing the vector-k sample of a given set, the general point is to produce vector-k samples for many sets, and use them as high-quality compact sketches in support vector machines and locality sensitive hashing.

Coupling for Counting Keys. We now discuss a stochastic dominance argument where we couple a Tornado Tabulation hashing experiment on a set A with a fully random hashing experiment on a slightly different set A'. Let us first consider the case of counting (distinct) keys, as in HyperLogLog applied to k-partition-min. Let h be the tornado hash function and $\tilde{h}$ be the fully-random hash function. Assuming that distinct keys hash to distinct values, the estimator only

depends on the set of hash values. Furthermore, as described above, we have made the selection such that with high probability, the estimator only depends on the hash values of the selected keys.

Now, for both hash functions, we first compute the select bits of the hash values and let L denote the set of hash values matching the bitmask. This defines the sets of selected keys $X = \{x \in A \mid h(A) \in L\}$ and $X' = \{x \in A' \mid \tilde{h}(A') \in L\}$. Next, we perform a maximal matching between keys in X and X', thus matching all keys in the smaller set. Since the free bits of the hash values of the selected keys are fully random in both cases, we can couple the hash values of matched pairs of keys so that matched keys have the same free bits in both experiments. As a result, we end up with the following relations

$$|X| \leqslant |X'| \iff (h(A) \cap L) \subseteq (\tilde{h}(A') \cap L) \implies HLL(A,h) \leqslant HLL(A',\tilde{h})$$
$$|X| \geqslant |X'| \iff (h(A) \cap L) \supseteq (\tilde{h}(A') \cap L) \implies HLL(A,h) \geqslant HLL(A',\tilde{h}).$$

Above HHL is the HyperLogLog estimator [FEFGM07] applied to the k-Partition-Min sketch. All we need to know is that it is increasing in the number of leading zeros of the min-key in each bucket. We assumed that $h(A) \cap L$ contained min-keys from each bucket. Therefore, if $|X| \leqslant |X'|$, then we get that $HLL(A,h) \leqslant HLL(A',\tilde{h})$. If, on the other hand, $|X'| \leqslant |X|$, then $(\tilde{h}(A') \cap L)$ could be missing keys in some bucket. Since $h(A) \cap L$ has keys in these buckets, $h(A)$ has at least t leading zeros while $\tilde{h}(A')$ has $< t$ leading zeros in these buckets. Therefore implying that $HLL(A,h) \geqslant HLL(A',\tilde{h})$. In other words, the estimator from $HLL(A',\tilde{h})$ would be lower because it has seen higher hash values $\tilde{h}(A')$.

The question now is how to set up the parameters such that the Tornado Tabulation hashing estimator is smaller than the fully random estimator with high probability. For this, we pick A' so large that, with high probability, we have $|X'| \geqslant |X|$. For some target error probability $O(P)$ and $\mu' = \mathrm{E}[|X'|] = |A'|/2^t$, it would be sufficient to have that

$$\mu + \sqrt{3\mu \ln(1/P)} \leqslant \mu' - \sqrt{2\mu' \ln(1/P)}\,.$$

This is using that we have the Chernoff upper-tail bound from (8) on $|X|$ and the classic Chernoff lower-tail bound on $|X'|$. Assuming $\mu' \leqslant 2\mu$ and $\mu \geqslant s/4$, we see it suffices that the following holds

$$\mu' \geqslant \mu\left(1 + \sqrt{12\ln(1/P)/s} + \sqrt{16\ln(1/P)/s}\right),$$

which in turn holds if

$$\mu' \geqslant \mu\left(1 + 8\sqrt{\ln(1/P)/s}\right)$$

Since $\mu = |A|/2^t$ and $\mu' = |A'|/2^t$, this means that if we want $|X| \geqslant |X'|$ to hold with probability $2P + 24(3/s)^b + 1/2^{s/2}$, it suffices to compare Tornado Tabulation hashing on A with fully-random hashing on A' with

$$|A'| = \left\lceil |A|\left(1 + 8\sqrt{\ln(1/P)/s}\right)\right\rceil.$$

When we want $|X'| \leqslant |X|$, we need to employ our new lower-tail bound from Theorem 1 on $|X|$ in combination with the classic Chernoff upper-tail bound on $|X'|$. Thus we want

$$\mu - \sqrt{7\mu \ln(1/P)} \geqslant \mu' + \sqrt{3\mu' \ln(1/P)}\,.$$

Assuming $\mu \geqslant s/4$, we see it suffices that

$$\mu' \leqslant \mu(1 - \sqrt{28\ln(1/P)/s} - \sqrt{12\ln(1/P)/s}),$$

which in turn holds if

$$\mu' \leqslant \mu(1 - 9\sqrt{\ln(1/P)/s})$$

Thus for $|X'| \leqslant |X|$ to hold with probability $3P + 24(3/s)^b + 1/2^{s/2}$, it suffices to compare Tornado Tabulation hashing on A with fully-random hashing on A' where

$$|A'| = \left\lfloor |A| \left(1 - 9\sqrt{\ln(1/P)/s}\right) \right\rfloor .$$

Coupling for Estimating Frequency. Finally, we consider the problem of estimating frequency, again using a coupling argument. This was discussed as a key example in [DKRT15], but using only O-notation (hiding large exponential constants). To get more precise bounds, one has to employ the same carefulness as we did above for counting keys.

Here, we have a set A of red and blue keys and we want to estimate the frequency of the least frequent color since this implies the best frequency estimate for both colors. Assume without loss of generality that red is the least frequent color. A main point in [DKRT15] is that we perform selection on two different levels. If we have r red keys and n keys in total in A, then we let a be the smallest number such that $r/2^a \leqslant s/2$ and t the smallest number such that $n/2^t \leqslant s/2$. We then first select based on the first a select bits (as a pre-selection). In expectation, this leads to between $s/4$ and $s/2$ pre-selected red keys. For an upper bound on the Tornado Tabulation estimator, we want more pre-selected red keys in A' (using fully random hashing) than in A (using Tornado Tabulation hashing). On the red keys, all remaining bits are fully random, so we can use the same coupling as we did above, just for counting. We note here that the pre-selection is essential if we want to get good bounds when the frequency of the red keys is very small.

Next, we settle the following $t - a$ select bits. At this point, we drop all pre-selected keys that weren't also selected in this second step (in effect, we do a sub-selection). From the perspective of the red keys, this sub-selection is fully random on and so the previous coupling ensures that The argument for the lower bound is symmetric: we decrease the number of red keys in A' and increase the total number of keys by adding more blue keys, using the same parameters as we did in the previous subsection where we were counting distinct keys.

3 Preliminaries: Tornado Tabulation Hashing

We now review the formal definition of tornado tabulation hashing, as well as the relevant technical results from Bercea et. al [BBK$^+$23]. First, we recall that a *simple tabulation hash function* [Zob70, WC81] is a function from some universe Σ^c to some range of hash values $\mathcal{R} = [2^r]$. Namely, we view the keys as being a concatenation of c *characters* from some alphabet Σ. In fact, our space parameter from the introduction is $s = |\Sigma|$. We use $x_1, \ldots, x_c$ to denote these characters, thus $x = x_1 \ldots x_c$. A simple tabulation hash function h associates with each character position $i = 1 \ldots c$ a table $T_i : \Sigma \longrightarrow \mathcal{R}$ that maps each character to a fully-random hash value. These tables are independent across different character positions. Given a key $x \in \Sigma^c$, the final hash value of x is computed as an exclusive or of all the individual character lookups:

$$h(x) = T_1[x_1] \oplus \cdots \oplus T_c[x_c] .$$

A tornado tabulation hash function uses multiple such simple tabulation hash functions, and can be thought of as a two-step process. In the first step, it extends the original key $x \in \Sigma^c$

into a *derived key* $\widetilde{h}(x) = \widetilde{x} \in \Sigma^{c+d}$, where $d \geqslant 0$ is an internal parameters that controls the final probability bounds we obtain. We refer to d as the number of derived characters. Namely, the first $c-1$ characters of $\widetilde{x}$ match x: if $\widetilde{x}_i$ denotes the i^{th} character of $\tilde{x}$, then $\widetilde{x}_i = x_i$ for all $i < c$. To compute $\tilde{x}_c$, we use a simple tabulation hash function $h_0 : \Sigma^{c-1} \to \Sigma$ and set $\tilde{x}_c = x_c \oplus h_0(\widetilde{x}_1 \cdots \widetilde{x}_{c-1})$. This character is often referred to as being twisted. For the remaining characters $\tilde{x}_{c+1}, \ldots, \tilde{x}_{c+d}$ (the derived characters), we employ a series of simple tabulation hash functions $\widetilde{h}_i : \Sigma^{c+i-1} \longrightarrow \Sigma$ and set

$$\widetilde{x}_{c+i} = \widetilde{h}_i\left(\widetilde{x}_1 \cdots \widetilde{x}_{i+c-1}\right) \text{ for } i = 1 \ldots d\,.$$

The last step in computing the hash value is to do one final round of simple tabulation hashing on the derived key. We denote this last round by $\widehat{h} : \Sigma^{c+d} \longrightarrow \mathcal{R}$. Then $h(x) = \widehat{h}(\widetilde{x})$.

Below is the C-code implementation of tornado tabulation for 64-bit keys, with $\Sigma = [2^{16}]$ $c = 4$, $d = 3$, and $R = [2^{64}]$. The function takes as input the key x, and $c + d$ fully random tables of size Σ, containing 128-bit values.

```
INT64 Tornado(INT64 x, INT128[7][65536] H) {
    INT32 i; INT128 h=0; INT16 c;
    for (i=0;i<3;i++) {
        c=x;
        x>>=16;
        h^=H[i][c];}
    h^=x;
    for (i=3;i<7;i++) {
        c=h;
        h>>16;
        h^=H[i][c];}
    return (INT64)h;}
```

Selection. We consider the setting in which a key is selected based on its value and its hash value. We do not consider query keys in our selection, as in [BBK+23]. Formally, we have a selector function $f : \Sigma^c \times \mathcal{R} \to \{0, 1\}$ and let $p_x := \Pr_{r \sim \mathcal{U}(\mathcal{R})}[f(x, r) = 1]$, i.e., the probability that a key is selected when its hash value is chosen uniformly at random from $\mathcal{R}$. The set of selected keys is then defined as

$$X = \{x \in \Sigma^c \mid f(x, h(x)) = 1\}\ ,$$

with $\mathrm{E}[|X|] = \mu = \sum_{x \in \Sigma^c} p_x$.

Local uniformity is shown for selector functions that select keys based on bitmasks. That is, we partition the bit representation of the final hash value $h(x)$ into s *selection bits* and t *free bits*, and let $h^{(s)}(x) \in [2^s]$ denote the s selection bits of $h(x)$. Then the selector function f has the property that $f(x, h(x)) = f(x, h^{(s)}(x))$. The remaining t bits of $h(x)$ are denoted by $h^{(t)}(x) \in [2^t]$ and are not involved in the selection process. Going back a step, we can define a similar partition on the bits of the final simple tabulation hash function $\widehat{h}$. That is, we let $\widehat{h}^{(s)}(x)$ denote the s selection bits of $\widehat{h}(x)$ and note that: $h^{(s)}(x) = \widehat{h}^{(s)}(\tilde{x})$. Similarly for the free bits of $\widehat{h}$, we have $h^{(t)}(x) = \widehat{h}^{(t)}(\tilde{x})$.

Linear Independence. A crucial ingredient in [BBK+23] is the notion of linear independence of a set of keys. Consider some set Y of keys in Σ^k, each consisting of k characters. Then the set Y is linearly independent if, for every subset $Y' \subseteq Y$, the keys in Y' have the following property: there exists a character that occurs an odd number of times in some position $i \in \{1, \ldots, k\}$. Conversely,

it cannot be that in each character position, all characters appear an even number of times across the keys in Y'. We then define

$$\mathcal{I}(Y) = \text{the event that the set } Y \text{ is linearly independent .}$$

For such linearly independent sets, Thorup and Zhang [TZ12] showed that a simple tabulation hash function is fully-random on the set Y if and only if Y is linearly independent. In the context of tornado tabulation, we focus on sets Y of derived keys and thus, linear independence is an event that depends only on the randomness of $\widetilde{h}$. Bercea et. al [BBK+23] then show the following:

Theorem 3. *Let $h = \widehat{h} \circ \widetilde{h} : \Sigma^c \to \mathcal{R}$ be a random tornado tabulation hash function with d derived characters and f as described above. If $\mu \leqslant \Sigma/2$, then the event $\mathcal{I}(\tilde{h}(X))$ fails with probability at most*

$$7\mu^3(3/|\Sigma|)^{d+1} + 1/2^{|\Sigma|/2} . \tag{10}$$

They also showed the following:

Theorem 4. *Let $h = \widehat{h} \circ \widetilde{h} : \Sigma^c \to \mathcal{R}$ be a random tornado tabulation hash function with d derived character and f as described above. Then, for any $\delta > 0$, we have that*

$$\Pr\left[|X| \geqslant (1+\delta)\cdot\mu \wedge \mathcal{I}(\widetilde{h}(X))\right] \leqslant \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu} .$$

We note that, from the above, one can obtain the classic Chernoff-style concentration (without $\mathcal{I}(\widetilde{h}(X))$), by summing the error probabilities from Theorem 3 and Theorem 4.

4 Technical Contribution

In this section, we describe the setup and techniques used in the proof of Theorem 1 which we restate below.

Theorem 1. *For any $b \geqslant 1$ and $c \leqslant \ln s$, if $s \geqslant 2^{16}\cdot b^2$, and $\mu \in [s/4, s/2]$. For any $\delta > 0$,*

$$\Pr[|X| < (1-\delta)\mu] < 3\exp\left(\frac{-\delta^2\mu}{7}\right) + (c+b+1)\ln(s)\cdot\left(49\left(\frac{3}{s}\right)^b + 3\left(\frac{1}{2}\right)^{s/2}\right) . \tag{8}$$

4.1 The Upper-Tail Bound (Theorem 4)

To appreciate our new lower-tail bound, we first briefly review the simple proof of the upper-tail bound from [BBK+23]. We will see why the same techniques breaks down for the lower-tail bound. However, it importantly turns out that we can still use some of the techniques for the upper tail bound in the proof of the lower tail bound. Namely, to get in a position to bound the *lower tail*, it helps to exclude certain *upper tail* error events. We will return to this point shortly.

The upper tail bound is the classic Chernoff bound as long as we also ask for the selected derived keys to be linearly independent, that is,

$$\Pr[X \geqslant (1+\delta)\mu \wedge \mathcal{I}(\tilde{h}(X))] \leqslant \exp(-\mu\delta^2/3).$$

The proof of this statement is basically the observation that in the standard proof of Chernoff bounds, the probability bound is a sum over different sets Y of the probability that $Y \subseteq X$, and this probability should be bounded by $\prod_{x \in Y} p_x$ where p_x is the marginal probability that x is selected. If the keys $\widetilde{Y} = \tilde{h}(Y)$ derived from Y are linearly independent (this depends only on $\tilde{h}$), then when we pick the top simple tabulation function at random,

$$\Pr_{\hat{h}}[Y \subseteq X \mid \mathcal{I}(\widetilde{h}(Y))] = \prod_{x \in Y} p_x.$$

However, $\mathcal{I} \wedge (Y \subseteq X)$ implies that $\tilde{Y}$ is linearly independent, and therefore

$$\Pr[Y \subseteq X \wedge \mathcal{I}(\widetilde{h}(Y))] \leqslant \Pr[Y \subseteq X \mid \mathcal{I}(\widetilde{h}(Y))] = \prod_{x \in Y} p_x.$$

We would like to do the same kind of argument for the lower tail bound, but here the Taylor expansion in the standard proof also sums over the probabilities of events that sets Y are non-selected in the sense $Y \cap X = \varnothing$. However, the hash values of non-selected keys are very dependent. In particular, we have no upper bound on the probability that $\Pr[(Y \cap X = \varnothing) \wedge I]$ which would, if independent, have been bounded by $\prod_{x \in Y}(1 - p_x)$.

4.2 High-Level Analysis for Lower-Tail Bound

For the analysis, we first partition the selected elements into buckets based on their last derived character, that is, we let X_α be the set of selected keys that have α as their final derived character, i.e.,

$$X_\alpha = \{x \in X \mid \widetilde{x}_{c+d} = \alpha\} .$$

Note that $X = \bigcup_{\alpha \in \Sigma} X_\alpha$ and we define $f = \mathrm{E}[|X_\alpha|] = \mu/|\Sigma| \leqslant 1/2$.

With full randomness the $(|X_\alpha|)_{\alpha \in \Sigma}$ would be independent Poisson distributed random variables. With Tornado Tabulation, they are neither independent nor Poisson distributed. However, we can argue that with high probability, in a certain sense they approximate this ideal. Below, we will introduce two *experiments* which describe this sense.

Introducing $\bar{h}$. Key to our analysis is to break up the definition of the hash function in a new way. Specifically, we divide the process of computing h differently. Let $\widehat{h}_{c+d} : \Sigma \to \mathcal{R}$ denote the table corresponding to the last derived character in our top simple tabulation function $\hat{h}$. In our C-code, this is the last table we look up in before we output the final hash value $h(x)$. Everything that comes before this last table lookup, we denote by $\bar{h} : \Sigma^c \to \mathcal{R} \times \Sigma$ (this includes the computations needed to obtain the full derived key $\widetilde{x}$). Note that $\overline{h}$ outputs two values. The first value, denoted as $\bar{h}[0]$, is a value in $\mathcal{R}$ and is the exclusive or of the first $c + d - 1$ table lookups that $\widehat{h}$ makes. The second value, denoted as $\bar{h}[1]$, is equal to the last derived character $\widetilde{x}_{c+d}$. Under this view, the final hash value can be computed as

$$h(x) = \bar{h}[0](x) \oplus \widehat{h}_{c+d}(\bar{h}[1](x)) .$$

Our tornado hash function h is thus defined by the two independent random variables $\bar{h}$ and $\hat{h}_{c+d}$.

The High-Level Analysis. Using the principle of deferred decision, we are going to make two different analyses depending on which of $\bar{h}$ and $\hat{h}_{c+d}$ is generated first. Each order provides a different understanding, and at the end, we link the two in a subtle way.

Experiment 1. Suppose we first fix $\bar{h}$ arbitrarily, while leaving $\hat{h}_{c+d}$ random. We claim that this does not change the expectation, that is, $\mathrm{E}\big[|X| \;\big|\; \bar{h}\big] = \mathrm{E}[|X|]$. Moreover, the $|X_\alpha|$ are completely independent, for when $\bar{h}$ is fixed, then so are all the derived keys, and then X_α only depends on the independent $\widehat{h}_{c+d}(\alpha)$.

The problem that remains is that the distribution of each $|X_\alpha|$ depends completely on how we fixed $\bar{h}$.

Experiment 2. Suppose instead we first fix $\hat{h}_{c+d}$ arbitrarily, while leaving $\bar{h}$ random. In this case, expanding a lot on the upper-tail bound techniques from [BBK+23], we will argue that with high probability, on the average, the values $|X_\alpha|$ will follow a distribution not much worse than if they were independent random Poisson variables. More precisely, we will argue that, w.h.p., for any $i \in \mathbb{N}$, the fraction of $\alpha \in \Sigma$ for which $|X_\alpha| \geqslant i$ is not much bigger than $\frac{f^i}{i!}$.

Compare this to a Poisson distributed variable Y with $\mathrm{E}[Y] = f$, where $\Pr[Y = i] = \frac{f^i}{i!}e^{-f}$. Naturally, $\Pr[Y \geqslant i] \geqslant \Pr[Y = i]$, and so the loss of our analysis relative to that of Poisson distributed variables is less than a factor e^f.

Linking the Experiments. We now want to link the above experiments. For $i \in \mathbb{N}$ define

$$S_i = |\{\alpha \in \Sigma : |X_\alpha| \geqslant i\}|$$

to be the number of characters α for which X_α contains at least i selected keys. Note that $|X| = \sum_{i\in\mathbb{N}} S_i$. In particular,

$$\mathrm{E}[|X|] = \mathrm{E}[|X| \mid \bar{h}] = \mathrm{E}[\sum_{i\in\mathbb{N}} S_i \mid \bar{h}] \tag{11}$$

by the discussion below Experiment 1. As stated under Experiment 2, w.h.p., S_i is not much bigger than $\frac{f^i}{i!}|\Sigma|$.

We now go back to Experiment 1 where $\bar{h}$ was fixed first. We would like to claim that, w.h.p., $E[|S_i| \mid \bar{h}]$ is also not much larger than $\frac{f^i}{i!}|\Sigma|$. If we can prove this, we are in a good shape to employ concentration bounds, for with $\bar{h}$ fixed, S_i is the sum of independent 0-1 variables, which are sharply concentrated by standard Chernoff bounds. Moreover, $|X| = \sum_{i\in\mathbb{N}} S_i$ and $\mathrm{E}\big[|X| \;\big|\; \bar{h}\big] = \mu$, so the error $X - \mu$ is the sum of the *layer* errors $S_i - \mathrm{E}[S_i \mid \bar{h}]$ where each i defines a layer. The rough idea then is to carefully apply concentration bounds within each layer and argue that the total sum of errors across layers is not too big.

Initially, we are only able to bound the probability that S_i is not too big, but we need an upper tail bound for $\mathrm{E}[S_i \mid \bar{h}]$. To get this, we need to link the two experiments. This link between the Experiment 2 analysis yielding high probability bounds on the size of S_i and high probability bounds on $\mathrm{E}[S_i \mid \bar{h}]$ as needed for Experiment 1 is Lemma 5 below.

Lemma 5. *For any* λ

$$\Pr\big[\mathrm{E}\big[S_i \;\big|\; \overline{h}\big] \geqslant \lambda + 1\big] \leqslant 2\Pr[S_i \geqslant \lambda]\ . \tag{12}$$

Proof. The proof of (12) is simple but subtle. We know from the discussion on Experiment 1 that conditioned on $\bar{h} = \bar{h}_0$ for any fixed $\bar{h}_0$, the distribution of S_i is a sum of independent $\{0,1\}$ variables. We wish to show that *if* $\overline{h}_0$ is such that $\mathrm{E}\big[S_i \,\big|\, \overline{h} = \bar{h}_0\big] \geqslant \lambda + 1$, *then* $\Pr\big[S_i \geqslant \lambda \,\big|\, \overline{h} = \bar{h}_0\big] \geqslant 1/2$. Conditioned on $\overline{h} = \bar{h}_0$, this is a question about sums of independent $\{0,1\}$ variables. We will use the following restatement of a corollary from [JS68].

Claim 6. [JS68, Corollary 3.1] Let $(Z_j)_{j=1}^m$ be independent random $\{0,1\}$ variables, $Z = \sum_{j=1}^m Z_j$, and $\mu = \mathrm{E}[Z]$. Then $\Pr[Z \geqslant \mathrm{E}[Z] - 1] > 1/2$.

To prove our lemma, write

$$\Pr[S_i \geqslant \lambda] \geqslant \Pr\big[\mathrm{E}\big[S_i \,\big|\, \overline{h}\big] \geqslant \lambda + 1\big] \cdot \Pr\big[S_i \geqslant \lambda \mid \mathrm{E}\big[S_i \,\big|\, \overline{h}\big] \geqslant \lambda + 1\big]$$

We thus have to show that $\Pr\big[S_i \geqslant \lambda \mid \mathrm{E}\big[S_i \,\big|\, \overline{h}\big] \geqslant \lambda + 1\big] \geqslant 1/2$. This follows more or less directly from claim 6, but let us write out the proof. Let $H = \{\bar{h}_0 \mid \mathrm{E}\big[S_i \,\big|\, \overline{h} = \bar{h}_0\big] \geqslant \lambda + 1\}$. From the corollary and the fact that conditioned on any $\bar{h} = \bar{h}_0$, S_i is the sum of independent $\{0,1\}$ variables, it follows that for any $\bar{h}_0 \in H$, $\Pr[S_i \geqslant \lambda \mid \overline{h} = \bar{h}_0] \geqslant 1/2$. Now we may write

$$\Pr\big[S_i \geqslant \lambda \mid \mathrm{E}\big[S_i \,\big|\, \overline{h}\big] \geqslant \lambda + 1\big] = \frac{\sum_{\bar{h}_0 \in H} \Pr[S_i \geqslant \lambda \mid \overline{h} = \bar{h}_0] \cdot \Pr[\overline{h} = \bar{h}_0]}{\Pr[\bar{h} \in H]} \geqslant \frac{\frac{1}{2}\sum_{\bar{h}_0 \in H} \Pr[\overline{h} = \bar{h}_0]}{\Pr[\bar{h} \in H]} = \frac{1}{2},$$

where we used the general formula $\Pr[A \mid \bigcup_{j\in J} B_j] = \frac{\sum_{j\in J} \Pr[A|B_j]\Pr[B_j]}{\Pr[\bigcup_{j\in J} B_j]}$ for disjoint events $(B_j)_{j\in J}$. This completes the proof. □

In fact, we do not have a direct way of bounding $\Pr[S_i \geqslant c]$ and we will have to consider a more restricted event which we can bound the probability of using the local uniformity result from [BBK+23]. This will be the content of Corollary 11 below.

Let us now reflect on what we achieved above and how it is useful for our goal. We are interested in lower tail bounds for $\mathrm{E}[X] = \sum_{i=1}^\infty S_i$ and we already observed in (11) that $\sum_{i=1}^\infty \mathrm{E}[S_i \mid \bar{h}] = \mathrm{E}[X]$. We would like to do a high probability bound over $\bar{h}$ to ensure that it has certain 'good' properties. If these properties hold, we hope to provide lower tail bounds for $\sum_{i=1}^\infty \mathrm{E}[S_i \mid \bar{h}]$. Conditioned on $\bar{h}$, each S_i is a sum of $\{0,1\}$ variables, and we know how to prove concentration bounds for such sums. Below we list the good properties which we will show that $\bar{h}$ satisfies with high probability.

1. $\sum_{i=1}^\infty \mathrm{E}[S_i \mid \bar{h}] = \mathrm{E}[X]$.
2. $\mathrm{E}[S_i \mid \bar{h}] \lesssim \frac{f^i}{i!}$.
3. $\mathrm{E}[S_i \mid \bar{h}] = 0$ when i is larger than some $i_{\max}$.

We already saw (Experiment 1) that 1. holds with probability 1. For 2., we will see in Section 4.4 how we can use an *upper tail* bound for Tornado Tabulation hashing to bound the probability that S_i is large (Experiment 2), and then the result will follow from Lemma 5 which links the two experiments. Finally, for 3., we need to prove a stronger version of the local uniformity theorem of [BBK+23] appearing as Theorem 17. The proof is technical but follows a similar path to the one used in [BBK+23]. However, we do require a novel combinatorial result for bounding dependencies of *simple tabulation hashing*. This result is Theorem 38 in Section 8

Layers With the properties 1.-3. in hand, the rough idea next is to prove a bound for each *layer* i of the form $\Pr[S_i \leqslant \mathrm{E}[S_i \mid \bar{h}] - \Delta_i] \leqslant p_i$. Note that each p_i will include the additive error probability $\Pr[\bar{h}$ not good$]$. As the layers are not independent, we have to union bound over each layer to bound the total error. Defining $\Delta = \sum_{i=1}^{i_{\max}} \Delta_i$ and $p = \sum_{i=1}^{i_{\max}} p_i$, we obtain

$$\Pr[|X| \leqslant \mathrm{E}[|X|] - \Delta] \leqslant \sum_{i=1}^{i_{\max}} \Pr[S_i \leqslant \mathrm{E}[S_i \mid \bar{h}] - \Delta_i] + p$$

which will be our desired bound. The most technical part of our proof thus employs various concentration bounds for events of the form $[S_i < \mathrm{E}\big[S_i \mid \overline{h}\big] - \Delta_i]$ depending on the values of $\bar{\mu}_i := \mathrm{E}[S_i \mid \bar{h}]$. Namely, we partition these S_i's into two different types of layers and use different lower tail techniques depending on which kind of layer we are dealing with. The challenge lies in setting the relative deviation Δ_i of each layer so that we get the desired overall deviation for $|X|$ and we do not incur a large penalty in the probability by conditioning on $\overline{h}$. We distinguish between *bottom* layers (which have large μ_i), *regular* layers and *non-regular* layers (which have excessively small μ_i).

Bounding Bucket Sizes. The fact that, we need that $\mathrm{E}[S_i \mid \bar{h}] = 0$ for all $i \geqslant i_{\max}$ comes from the fact that we need to union bound over all layers, and the error probability t_i for a single layer i includes the additive $\Pr[\bar{h}$ not good$]$. Without an upper bound on i, we might even have to union bound over $|\Sigma|^c$ layers which will come as a significant cost for our error bounds. In fact, the upper limit $i_{\max}$ will be such that, with high probabiliy over $\overline{h}$, $|X_\alpha| \leqslant i_{\max}$ for all $\alpha \in \Sigma$. Our bound on $i_{\max}$ appears in Lemma 16 with the proof appearing in Section 5.6.

In the next two subsections, we will zoom in on Experiment 1 and Experiment 2.

4.3 Experiment 1

In this experiment, we first fixed $\bar{h}$ arbitrarily, noting that then the $|X_\alpha|$ are completely independent since X_α now only depends on $\widehat{h}_{c+d}(\alpha)$. We also claimed that the fixing of $\bar{h}$ did not change the expectation of $|X|$. More specifically, we claim that conditioning on $\overline{h}$ does not change the probability that a specific key x gets selected.

Observation 7. Let I_x be the indicator random variable that is 1 if x gets selected and 0 otherwise. Then, for every fixed key x and every fixed value of $\bar{h}$, we have that:

$$\Pr[I_x] = \Pr\big[I_x \mid \overline{h}\big] \ .$$

This result is well-known from [BBK+23], but we include a proof for completeness.

Proof. We first note that $\Pr[I_x] = \sum_{r \in \mathcal{R}} \Pr[h(x) = r] \cdot \Pr[I_x \mid h(x) = r]$. By definition, once we fix the key x and its hash value, then selection becomes deterministic. Therefore

$$\Pr[I_x \mid h(x) = r] = \Pr\big[I_x \mid h(x) = r \wedge \overline{h}\big] \ .$$

The only thing left to prove therefore is that $\Pr[h(x) = r] = \Pr\big[h(x) = r \mid \overline{h}\big]$. On one hand, we know that h hashes uniformly in the range of hash values, so $\Pr[h(x) = r] = 1/|\mathcal{R}|$. On the other hand, we know that

$$\begin{aligned}\Pr\big[h(x) = r \mid \overline{h}\big] &= \Pr\big[\bar{h}[0](x) \oplus T_{c+d}(\bar{h}[1](x)) = r \mid \overline{h}\big] \\ &= \Pr\big[T_{c+d}(\bar{h}[1](x)) = r \oplus \bar{h}[0](x) \mid \overline{h}\big] \\ &= 1/|\mathcal{R}|\ ,\end{aligned}$$

where the last inequality holds because the last table lookup $T_{c+d}(\bar{h}[1](x))$ picks a value uniformly at random from $\mathcal{R}$. □

As seen in the proof above, once we condition on $\overline{h}$, the randomness in selection only comes from the last table lookup. That is, conditioned on $\overline{h}$, the random variables $\{X_\alpha\}_{\alpha\in\Sigma}$ become independent, i.e., elements across different X_α's will be selected independently. This, however, is not enough. That is because if we condition on $\overline{h}$, we no longer know how many of the selected keys have a particular last derived character. Thus, even though the random variables $\{X_\alpha\}_{\alpha\in\Sigma}$ are independent, they have different, unknown distributions. We cannot bound their variance nor apply Chernoff on their scaled versions and get competitive bounds.

4.4 Experiment 2

In this experiment, we first fix $\hat{h}_{c+d}$ arbitrarily, while leaving $\bar{h}$ random. As stated, we will analyze this case expanding on the techniques from [BBK+23].

For a given key x, let $\widetilde{x}_{<c+d}$ denote the derived key except the last derived character $\widetilde{x}_{c+d}$. Moreover, we define $\tilde{h}_{<c+d}$ such that $\tilde{h}_{<c+d}(x) = \widetilde{x}_{<c+d}$. We will be very focused on the event that these shortened derived selected keys are linearly independent, and for ease of notation, we define the event

$$\mathcal{J} = \mathcal{I}(\widetilde{h}_{<c+d}(X)).$$

Using Theorem 3, we prove that this event happens with high probability. More precisely,

Theorem 8.

$$\Pr[\mathcal{J}] \geqslant 1 - (24(3/|\Sigma|)^{d-3} + 1/2^{|\Sigma|/2}).$$

Proof. We will prove that

$$\Pr[\mathcal{J} \mid \hat{h}_{c+d}] \geqslant 1 - (24(3/|\Sigma|)^{d-3} + 1/2^{|\Sigma|/2}).$$

Since the bound holds for any $\hat{h}_{c+d}$, it also holds unconditionally. We consider the function $\bar{f}\colon \Sigma^c \times (\mathcal{R} \times \Sigma)$ defined by $\bar{f}(x,(r,\alpha)) = f(x, r \oplus \hat{h}_{c+d}(\alpha))$. Since $h(x) = \bar{h}[0](x) \oplus \hat{h}_{c+d}(\bar{h}[1](x))$ then $f(x,h(x)) = \bar{f}(x,\bar{h}(x))$ and $X = \{x \in U \mid f(x,h(x)) = 1\} = \{x \in U \mid \bar{f}(x,\bar{h}(x)) = 1\}$. When $\hat{h}_{c+d}$ is fixed, $\bar{f}$ is a deterministic function and since $\bar{h}$ is a tornado hash function with $d-1$ derived characters, the result of Theorem 3 gives the claim. □

We can now think of keys being picked independently for α. In the same way, as we proved Chernoff upper bounds for events like $[X \geqslant (1+\delta)\mu \wedge \mathcal{I}(\widetilde{h}(X))]$, we can prove

Lemma 9. *For any $i \in \mathbb{N}$, we have that*

$$\Pr[|X_\alpha| \geqslant i \wedge \mathcal{J}] \leqslant \frac{f^i}{i!}\ .$$

The proof of Lemma 9 can be found in Section 5.2. Recall that $S_i = |\{\alpha \in \Sigma : |X_\alpha| \geqslant i\}|$. By Lemma 9 and linearity of expectation, we have that

$$\mathrm{E}[S_i \cdot [\mathcal{J}]] \leqslant |\Sigma| \cdot \frac{f^i}{i!} = \bar{\mu}_i \ .$$

Moreover, we can show (also in Section 5.2) a bound on the upper tail of $|S_i|$ in terms of $\bar{\mu}_i$ as such:

Lemma 10. *For any $\delta > 0$:*

$$\Pr[S_i \geqslant (1+\delta) \cdot \bar{\mu}_i \wedge \mathcal{J}] \leqslant \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^{\bar{\mu}_i} \ .$$

With these lemmas in place, we will briefly revisit the issue of bounding the conditional expectation $\mathrm{E}\big[S_i \,\big|\, \overline{h}\big]$ by bounding S_i, as discussed previously. The following corollary follows directly from Lemma 5, bringing in event $\mathcal{J}$ to give an expression that we can bound with Lemma 10. We note that the event $\Pr[\neg\mathcal{J}]$ can be bounded using Theorem 8.

Corollary 11. *For any λ*

$$\Pr\big[\mathrm{E}\big[S_i \,\big|\, \overline{h}\big] \geqslant \lambda + 1\big] \leqslant 2\Pr[(S_i \geqslant \lambda) \wedge \mathcal{J}] + 2\Pr[\neg\mathcal{J}] \ .$$

4.5 Roadmap of Technical Part of the Paper

In Section 5, we will provide lower tail bounds for the deviation incurred in each layer. In Section 5.1, we describe the main ideas in the analysis and introduce the different types of layers. We provide some preliminary tools for the analysis in Section 5.2. Next, in Sections 5.3 to 5.5 we analyse respectively bottom layers, regular layers, and non-regular layers. Finally Section 5.6 bounds $i_{\max}$.

In Section 6, we prove our improved local uniformity theorem (needed for bounding $i_{\max}$). We provide necessary definitions (from [BBK$^+$23]) in Section 6.1. In Sections 6.2 to 6.4, we show how to modify the obstructions from [BBK$^+$23] and how to union bound over them.

Section 7 is dedicated to prove our main results. For this, we need a technical theorem which we prove in Section 7.1. In Section 7.2, we prove Theorem 1 and in Section 7.3, we prove the subsampling Theorem 2.

Finally, Section 8 contains our new combinatorial result on bounding dependencies for simple tabulation hashing.

In Appendix A, we include a Chernoff bound working under a slightly weaker assumption than independence. We will need this Chernoff bound in the layer analysis.

5 Layers

In this section we present and prove technical theorems for the layers, and the associated parameters, as used for proving the main theorems. Our main technical proof is a union bound over events of the form "$S_i < \mathrm{E}\big[S_i \,\big|\, \overline{h}\big] - \Delta_i$" that will roughly hold with probability at most p_i. We will treat these events differently, depending on the values of $\bar{\mu}_i = |\Sigma| \cdot f^i/i!$, which is an an upper bound on $\mathrm{E}[S_i]$ (see Lemma 18).

Namely, for i large enough, when $\bar{\mu}_i < p$, we can design events such that both the sum of deviations $\sum \Delta_i$ and associated error probabilities $\sum p_i$ form geometric series, and are thus finite

(see Lemma 26). However, as will be apparent in the statements of the theorems given below, we still incur a small constant error probability for each layer handled, originating from our applications of Corollary 11. This accumulated error probability will be too high in general, so we further argue that we incur it only for a (relatively) small number of layers. Namely, in Lemma 16 (Section 5.6), we show that, with high probability, $\mathrm{E}\big[S_i \bigm| \overline{h}\big] = 0$ for all i larger than a threshold $i_{\max}$. Thus, the deviation for these layers will be zero.

The main technical challenge thus lies in handling the layers where $\bar{\mu}_i \geqslant p$. As $\bar{\mu}_1 = \mu$ there will be $\Omega\left(\ln(\mu) + \ln(1/p)\right)$ such layers before Lemma 26 applies. With one event defined for each layer, we are thus dealing with a superconstant number of events, and we will need to perform some scaling of the error probabilities if we want them to sum to $O(p)$. Again, the method for doing this depends on the expected size of the layer.

We define quite a few symbols in the treatment of the different layer types. A reference is given in table 1 on page 44. For building intuition, we ignore symbols s_{sec} and s_{all} in the following paragraphs. These can both be considered to equal 1 without altering the structure of the proof, which will suffice to build a theorem with error probability $O(p)$. The role of these parameters is covered in the final paragraph below and serves to control the constant hidden in the O-notation.

Roadmap. In Section 5.1, we review and explain the main lemmas we use to bound the deviation in each layer. In particular, we partition the layers into bottom, regular and non-regular. After some preliminaries in Section 5.2, we then prove each of the lemmas in the following sections. Namely, the proofs for the bottom layers are given in Section 5.3. The proof for the regular layers is given in Section 5.4. Finally, the proof for non-regular layers is given in Section 5.5.

5.1 Main ingredients

We here discuss the main ideas needed for carrying out the layer analysis.

Regular Layers. A layer is said to be *regular* if $\bar{\mu}_i \geqslant \ln(1/p_i)$, in which case $\mathrm{E}\big[S_i \bigm| \overline{h}\big]$ won't be significantly larger than $\bar{\mu}_i$, by Lemma 19.

As $\mathrm{E}[|X_\alpha|] = \mu/|\Sigma| \leqslant 1/2$ most elements of X are expected to be found in the very first layers. In particular, we handle the combined deviation of layers 1, 2, and 3 through an application of Bernstein's inequality, which gives better bounds than those obtained through individual treatment of the layers.

Theorem 12 (3 layers with Bernstein)**.** *Assume that* $\ln(s_{sec}/p) \leqslant \bar{\mu}_3$*. Then*

$$\Pr\left[S_{\leqslant 3} \leqslant \mathrm{E}\big[S_{\leqslant 3} \bigm| \overline{h}\big] - \sqrt{\frac{7}{3}\ln(1/p)\mu \cdot (1+\varepsilon_3)} - \ln(1/p)\right] < (1 + 4/s_{sec}) \cdot p + 4\Pr[\neg\mathcal{J}] \ .$$

If layer 3 isn't regular, we have the following alternate theorem.

Theorem 13 (2 layers with Bernstein)**.** *Assume that* $\ln(s_{sec}/p) \leqslant \bar{\mu}_2$*. Then*

$$\Pr\left[S_{\leqslant 2} \leqslant \mathrm{E}\big[S_{\leqslant 2} \bigm| \overline{h}\big] - \sqrt{2\ln(1/p)\mu \cdot (1+\varepsilon_2)} - \frac{2}{3}\ln(1/p)\right] < (1 + 2/s_{sec}) \cdot p + 2\Pr[\neg\mathcal{J}] \ ,$$

where $\varepsilon_2 = \sqrt{6\ln(s_{sec}/p)/|\Sigma|} + (\frac{2}{9}\ln(1/p) + 2)/\mu$.

Note that $\varepsilon_2 \leqslant \varepsilon_3$ (see table 1), and we thus use the latter in the statement of Theorem 35. Both theorems are proven in Section 5.3.

If the following layer(s) are also regular we treat these individually with Lemma 23, which first gives an upper bound on $\mathrm{E}\big[S_i \mid \overline{h}\big]$ and then bounds the deviation between the conditional expectation and S_i through a second Chernoff bound as described in the previous section.

Let i_{nr} be the first layer which is not regular. That is, $i_{nr} = \min\{i : \bar{\mu}_i < \ln(1/p_i\}$. Then Lemma 23 is applied to $i_{nr} - 4$ layers in total. As $\bar{\mu}_i = \mu f^{i-1}/i! \leqslant \mu/2 \cdot (1/6)^{i-2}$, we have $i_{nr} - 4 \leqslant \log_6\left(\frac{\mu/2}{\ln(1/p)}\right)$ when all $p_i \leqslant p$. Setting $p_i = p_{reg} = \frac{p}{\log_6\left(\frac{\mu/2}{\ln(1/p)}\right)}$ for each layer $i \in \{4, \ldots, i_{nr} - 1\}$ would thus ensure that the total error probability on these layers is $O(p)$.

However, as allocating a smaller error probability incurs a larger deviation relative to the expected size of the layer, it seems unwise to set the same low error probability for all regular layers. They will have a progressively smaller impact on the final result, after all.

Instead, we let $p_4 = p$ and set $p_{i+1} = \max\{p_i/e, p_{reg}\}$ such that $\sum_{i=4}^{i_{nr}-1} p_i \leqslant (1.59 + 1) \cdot p$. Applying Lemma 23 with these values for p_i gives the following bound, the proof of which is found in Section 5.4.

Theorem 14.

$$\Pr\left[\sum_{i=4}^{i_{nr}-1} S_i < \sum_{i=4}^{i_{nr}-1} \mathrm{E}\big[S_i \mid \overline{h}\big] - \Delta_{reg}\right] < (1.59 + 1/s_{all}) \cdot (1 + 2/s_{sec}) \cdot p + (i_{nr} - 4) \cdot 2\Pr[\neg\mathcal{J}] \ .$$

Non-regular layers. The non-regular layers are handled by three different lemmas: Lemma 25 for the single layer i_{nr} where $\bar{\mu}_i \approx \ln(1/p_{reg})$, Lemma 26 for layer i_∞ and up where $\bar{\mu}_i < p$, and Lemma 27 for the layers between the two.

To keep the total error probability of these "top" layers at $O(p)$ we set $p_i \leqslant p/n_{top}$ for each layer treated by Lemma 27, where n_{top} is (an upper bound on) the number of such layers. As the top layers are those where $p < \bar{\mu}_i \leqslant \ln(1/p_{reg})$ and we assume that $i_{nr} \geqslant 3$ (and thus $\bar{\mu}_{i+1} \leqslant \bar{\mu}_i/6$) there will at most be $\log_6(\ln(1/p_{reg}) \cdot 1/p) = n_{top}$ of these layers.

As an extra complication, note that i_{nr} is defined in terms of the threshold p_{reg} used for the regular layers, and thus the tools used for the non-regular layers only hold for $p_i < p_{reg}$. This leads to the somewhat cumbersome definition of $p_{top} = \max\{p_{reg}, p/n_{top}\}$.

At this stage, our bounds on $\mathrm{E}\big[S_i \mid \overline{h}\big]$ will be smaller than $\ln(1/p_i)$ and thus a Chernoff bound will no longer give a meaningful bound on the probability that S_i is smaller than $\mathrm{E}\big[S_i \mid \overline{h}\big]$. Instead we use the trivial observation that $\mathrm{E}\big[S_i \mid \overline{h}\big] - S_i \leqslant \mathrm{E}\big[S_i \mid \overline{h}\big]$ as S_i is non-negative.

The combined deviation and error probability of these non-regular layers is summarized in Theorem 15 below, the proof of which can be found in Section 5.5.

Theorem 15. *If $i_{nr} \geqslant 3$, then*

$$\Pr\left[\sum_{i=i_{nr}}^{i_{\max}} S_i < \sum_{i=i_{nr}}^{i_{\max}} \mathrm{E}\big[S_i \mid \overline{h}\big] - \Delta_{nonreg}\right] < p \cdot 6/s_{all} + (i_{\max} - i_{nr}) \cdot 2\Pr[\neg\mathcal{J}] \ .$$

Bounding the number of layers. As discussed above, we bound the number of layers under consideration to limit the accumulation of error terms from applications of Corollary 11. Specifically, we let $\mathcal{J}_{\max}$ be the event that $\sum_{i=1}^{i_{\max}} \mathrm{E}\big[S_i \mid \overline{h}\big] = \mu$, such that summing the deviation found in these

layers represents the full deviation between μ and $|X|$. The following Lemma bounds the probability of $\mathcal{J}_{\max}$.

Lemma 16. *Let $s \leqslant |\Sigma|/2$ be the number of selection bits and $i_{\max} = \ln(|\Sigma|^{d-2}2^s)$. Then*

$$\Pr\left[\sum_{i=i_{\max}+1}^{\infty} \mathrm{E}\big[S_i \mid \overline{h}\big] > 0\right] \leqslant \left(\frac{1}{|\Sigma|}\right)^{d-3} + 3^{c+1}\left(\frac{3}{|\Sigma|}\right)^{d-1} + \left(\frac{1}{|\Sigma|}\right)^{|\Sigma|/2-1}.$$

In order to obtain this result, we require an improvement ofTheorem 3. Namely, we need a better bound on the probability that the derived keys are linearly dependent. This requires a modification of the analysis in [BBK+23]. The result is as follows.

Theorem 17. *Let $h = \widehat{h} \circ \widetilde{h} : \Sigma^c \to \mathcal{R}$ be a random (simple) tornado tabulation hash function with d derived characters and f as described above. If $\mu \leqslant \Sigma/2$, then the event $\mathcal{I}(\tilde{h}(X))$ fails with probability at most*

$$3^c\,|\Sigma|/n \cdot 3\mu^3(3/|\Sigma|)^{d+1} + f^{|\Sigma|/2} \tag{13}$$

We note that the first term is a factor $3^c\,|\Sigma|/n$ smaller than the bound in Theorem 3. Indeed, we prove it by showing a different analysis than the one for Theorem 3 from [BBK+23]. Details can be found in Section 6.

Parameters s_{sec} and s_{all}. As seen in the theorems above, parameters s_{sec} and s_{all} serve to scale the error probabilities. For our proof, we set $s_{sec} = 20$ and $s_{all} = 180$ (as given in table 1), but all of the theorems hold regardless of the values chosen – as long as the same values are used across all layer types.

Specifically, s_{all} scales the error probability of most events, (including the threshold p_{reg} used for the regular layers as well the events defined on non-regular layers) while s_{sec} alters the ratio of the error probabilities between the two events considered on each regular layer: As a slightly looser bound on $\mathrm{E}\big[S_i \mid \overline{h}\big]$ has little impact on the deviation obtained from the regular layers, we opt for a more conservative bound on these, in exchange for a smaller error probability. If we allocate error p_i for the "primary" event, in which we bound the absolute difference $(\mathrm{E}\big[S_i \mid \overline{h}\big] - S_i)$ for a fixed value of the conditional expectation, we instead spend error p_i/s_{sec} on the "secondary" event where we bound $\mathrm{E}\big[S_i \mid \overline{h}\big]$.

With this terminology, the regular layers consist of both primary and secondary events, while the non-regular layers exclusively consist of secondary events. This aligns with an intuitive understanding that the regular layers lead to deviation proportional to $\sqrt{\ln(1/p)\mu}$ while the non-regular layers contribute deviation in terms of $\ln(1/p)$. Note, however, that the error probability of the secondary event, at p_{reg}/s_{sec} is what determines the boundary i_{nr} between regular and non-regular layers, which is why s_{sec} ends up appearing in the deviation found in the non-regular layers.

For Theorem 35 we've set s_{sec} and s_{all} quite high in order to bring the total error probability down to $3p$. At the other extreme, setting $s_{sec} = s_{all} = 1$ would make for an equally viable theorem with fewer additive terms in the deviation. It's total error probability would be roughly $19p$, however.

5.2 Preliminaries

We need the following general tools for bounding S_i which, together with Corollary 11, allows us to bound $\mathrm{E}\big[S_i \mid \overline{h}\big]$.

Lemma 18 (Expected size of a layer)**.**

$$\mathrm{E}[S_i \cdot [\mathcal{J}]] \leqslant \bar{\mu}_i = |\Sigma| \cdot \frac{f^i}{i!}$$

Proof. Let $p_x = \Pr[x \in X]$ be the probability that each key x is selected. Note that $\Pr[x \in X_\alpha] = p_x/|\Sigma|$ as the last derived character of $\overline{h}(x)$ is uniformly distributed over Σ. When restricted to event $\mathcal{J}$ we further have $\Pr[Y \subseteq X_\alpha \wedge \mathcal{J}] \leqslant \prod_{x \in Y} \Pr[x \in X_\alpha]$.

$$\begin{aligned}
\Pr[|X_\alpha| \geqslant i \wedge \mathcal{J}] &\leqslant \sum_{\{x_1,\ldots,x_i\} \in \binom{n}{i}} \prod_{k=1}^{i} \frac{p_{x_k}}{|\Sigma|} \\
&\leqslant \frac{1}{i! \cdot |\Sigma|^i} \sum_{\langle x_1,\ldots,x_i \rangle \in [n]^i} \prod_{k=1}^{i} p_{x_k} \\
&= \frac{1}{i! \cdot |\Sigma|^i} \prod_{k=1}^{i} \sum_{x \in [n]} p_x \\
&= \frac{\mu^i}{i! \cdot |\Sigma|^i} = \frac{f^i}{i!}\,.
\end{aligned}$$

Then

$$\mathrm{E}[S_i \cdot [\mathcal{J}]] = \sum_{\alpha \in \Sigma} \Pr[|X_\alpha| \geqslant i \wedge \mathcal{J}] \leqslant |\Sigma| \cdot \frac{f^i}{i!}\,.$$

□

Lemma 19 (Upper tail for layer size)**.**

$$\Pr[S_i > (1+\delta)\bar{\mu}_i \wedge \mathcal{J}] \leqslant \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}} \right)^{\bar{\mu}_i}$$

where $\bar{\mu}_i = |\Sigma| f^i/i!$ *and* $\delta > 0$.

Proof. First observe that bounding $(S_i \wedge \mathcal{J})$ is equivalent to bounding the sum of indicator variables $\sum_{\alpha \in \Sigma} [|X_\alpha| \geqslant i \wedge \mathcal{J}]$. We will show that these indicator variables satisfy the conditions of a slightly generalized Chernoff bound (Lemma 41 in Appendix A) with $p_\alpha = f^i/i!$ for all $\alpha \in \Sigma$ such that $\sum_{\alpha \in \Sigma} p_\alpha = \bar{\mu}_i$. Hence we need to show that, for any set of characters $\{\alpha_1, \ldots, \alpha_k\} \subseteq \Sigma$, we have $\Pr\left[\bigwedge_{l=1}^{k} |X_{\alpha_i}| \geqslant i \wedge \mathcal{J}\right] \leqslant (f^i/i!)^k$.

For each $x \in \Sigma^c$ let $q_x = \Pr[x \in X]$ such that $\sum q_x = \mu$. Then

$$\begin{aligned}
\Pr\left[\bigwedge_{\ell=1}^{k} |X_{\alpha_i}| \geqslant i \wedge \mathcal{J}\right] &= \Pr\left[\bigwedge_{\ell=1}^{k} \exists A_\ell \in \binom{n}{i} : A_\ell \subseteq X_{\alpha_\ell} \wedge \mathcal{J}\right] \\
&\leqslant \sum_{\substack{\langle A_1,\ldots,A_k\rangle \in \binom{n}{i}^k \\ \text{disjoint}}} \Pr\left[\bigwedge_{\ell=1}^{k} A_\ell \subseteq X_{\alpha_\ell} \wedge \mathcal{J}\right] \\
&\leqslant \sum_{\substack{\langle A_1,\ldots,A_k\rangle \in \binom{n}{i}^k \\ \text{disjoint}}} \prod_{x \in \bigcup_{\ell=1}^{k} A_k} \frac{q_x}{|\Sigma|} \leqslant \frac{1}{(i! \cdot |\Sigma|^i)^k} \sum_{\mathcal{A} \in [n]^{i \cdot k}} \prod_{x \in \mathcal{A}} q_x \\
&= \frac{1}{(i! \cdot |\Sigma|^i)^k} \prod_{\ell=1}^{i \cdot k} \sum_{x \in [n]} q_x = \frac{1}{(i! \cdot |\Sigma|^i)^k} \prod_{\ell=1}^{i \cdot k} \mu = \left(\frac{f^i}{i!}\right)^k .
\end{aligned}$$

□

5.3 Bottom layers: Proof of Theorems 12 and 13

The proofs of Theorems 12 and 13 are both based on an application of Bernstein's inequality. For non-centered independent variables $X_1, X_2, \ldots$, Bernstein's inequality states that

$$\Pr\Big[\sum X_i \leqslant \mathrm{E}\Big[\sum X_i\Big] - t\Big] \leqslant \exp\left(\frac{-0.5t^2}{\sum \mathrm{Var}[X_i] + t \cdot M/3}\right) \tag{14}$$

where M is a value such that $|X_i| \leqslant M$.

For Theorem 13 we sum over $\hat{X}_\alpha = \min\{3, |X_\alpha|\}$ such that $S_{\leqslant 3} = S_1 + S_2 + S_3 = \sum_{\alpha \in \Sigma} \hat{X}_\alpha$. As the $\hat{X}_\alpha$ are independent when conditioned on $\overline{h}$ we can apply Bernstein's when we've established a bound on $\sum \mathrm{Var}\big[\bar{X}_\alpha \mid \overline{h}\big]$.

Lemma 20. *If* $\ln(s_{sec}/p) \leqslant |\Sigma| f^3/6 = \bar{\mu}_3$,

$$\Pr\left[\sum_{\alpha \in \Sigma} \mathrm{Var}\Big[\hat{X}_\alpha \mid \overline{h}\Big] > (7/6 + \varepsilon)\,\mu\right] < p \cdot 4/s_{sec} + 4\Pr[\neg\mathcal{J}]$$

where $\varepsilon = (2 + \sqrt{6})\sqrt{\ln(s_{sec}/p)/|\Sigma|} + 6/\mu$.

Proof. Define $\hat{\mu} = \sum_\alpha \mathrm{E}\Big[\hat{X}_\alpha \;\Big|\; \overline{h}\Big] = \mathrm{E}\big[S_{\leqslant 3} \mid \overline{h}\big]$ and $\hat{f} = \hat{\mu}/|\Sigma|$. It then holds that

$$\sum_{\alpha \in \Sigma} \Big(\mathrm{E}\Big[\hat{X}_\alpha \;\Big|\; \overline{h}\Big]\Big)^2 \geqslant \sum_{\alpha \in \Sigma} (\hat{\mu}/|\Sigma|)^2 = |\Sigma| \hat{f}^2$$

and thus

$$\begin{aligned}
\sum_{\alpha\in\Sigma} \mathrm{Var}\Big[\hat{X}_\alpha \mid \overline{h}\Big] &= \sum_{\alpha\in\Sigma} \mathrm{E}\Big[\hat{X}_\alpha^2 \,\Big|\, \overline{h}\Big] - \sum_{\alpha\in\Sigma} \Big(\mathrm{E}\Big[\hat{X}_\alpha \,\Big|\, \overline{h}\Big]\Big)^2 \\
&\leqslant \sum_{\alpha\in\Sigma} \mathrm{E}\Big[\hat{X}_\alpha^2 \,\Big|\, \overline{h}\Big] - |\Sigma|\hat{f}^2 \\
&= \sum_{\alpha\in\Sigma} \mathrm{E}\Big[\hat{X}_\alpha(\hat{X}_\alpha - 1) \,\Big|\, \overline{h}\Big] + \sum_{\alpha\in\Sigma} \mathrm{E}\Big[\hat{X}_\alpha \,\Big|\, \overline{h}\Big] - |\Sigma|\hat{f}^2 \\
&= 2\cdot \mathrm{E}\Big[\Big|\Big\{\alpha : \hat{X}_\alpha = 2\Big\}\Big| \,\Big|\, \overline{h}\Big] + 6\cdot \mathrm{E}\Big[\Big|\Big\{\alpha : \hat{X}_\alpha = 3\Big\}\Big| \,\Big|\, \overline{h}\Big] + \hat{\mu} - |\Sigma|\hat{f}^2 \\
&= 2\cdot \mathrm{E}\big[S_2 \mid \overline{h}\big] + 4\cdot \mathrm{E}\big[S_3 \mid \overline{h}\big] + \hat{\mu} - |\Sigma|\hat{f}^2\,.
\end{aligned}$$

We will now bound $\mathrm{E}\big[S_2 \mid \overline{h}\big]$ and $\mathrm{E}\big[S_3 \mid \overline{h}\big]$. Applying Lemma 19 with $\delta_2 = \sqrt{3\ln(s_{sec}/p)/\bar{\mu}_2}$ we have

$$\Pr[S_2 \geqslant (1+\delta_2)\bar{\mu}_2 \wedge \mathcal{J}] \leqslant p/s_{sec}\,.$$

Invoking Corollary 11, we have for $\mu_2^+ = (1+\delta_2)\bar{\mu}_2 + 1$

$$\Pr\big[\mathrm{E}\big[S_2 \mid \overline{h}\big] \geqslant \mu_2^+\big] \leqslant 2p/s_{sec} + 2\Pr[\neg\mathcal{J}]\ .$$

Likewise for $\delta_3 = \sqrt{3\ln(s_{sec}/p)/\hat{\mu}_3}$ and $\mu_3^+ = (1+\delta_3)\bar{\mu}_3 + 1$,

$$\Pr\big[\mathrm{E}\big[S_3 \mid \overline{h}\big] \geqslant \mu_3^+\big] \leqslant 2p/s_{sec} + 2\Pr[\neg\mathcal{J}]\ .$$

Hence, with probability at least $1 - p\cdot 4/s_{sec} - 4\Pr[\neg\mathcal{J}]$ we have,

$$\begin{aligned}
\sum_{\alpha\in\Sigma} \mathrm{Var}\Big[\hat{X}_\alpha \mid \overline{h}\Big] &< 2\mu_2^+ + 4\mu_3^+ + \hat{\mu} - |\Sigma|\hat{f}^2 \\
&= 2(1+\delta_2)\bar{\mu}_2 + 4(1+\delta_3)\bar{\mu}_3 + \hat{\mu} - |\Sigma|\hat{f}^2 + 6 \\
&= 2\delta_2\bar{\mu}_2 + 4\delta_3\bar{\mu}_3 + 2|\Sigma|f^2/2 + 4|\Sigma|f^3/6 + \hat{\mu} - |\Sigma|\hat{f}^2 + 6 \\
&= \delta_2|\Sigma|f^2 + \frac{2}{3}\delta_3|\Sigma|f^3 + |\Sigma|f^2 + \frac{2}{3}|\Sigma|f^3 + \hat{\mu} - |\Sigma|\hat{f}^2 + 6 \\
&= \left(1 + \delta_2 + \frac{2f}{3}\cdot(1+\delta_3)\right)|\Sigma|f^2 + \hat{\mu} - |\Sigma|\hat{f}^2 + 6\,.
\end{aligned}$$

Note that the function $x \mapsto x - x^2$ is increasing in $x \in [0, 1/2]$. As $\bar{f} \leqslant f \leqslant 1/2$ we thus have $\bar{f} - \bar{f}^2 \leqslant f - f^2$ and

$$|\Sigma|f^2 + \hat{\mu} - |\Sigma|\hat{f}^2 = |\Sigma|(f^2 + \hat{f} - \hat{f}^2) \leqslant |\Sigma|\cdot f = \mu$$

giving

$$\begin{aligned}
\sum_{\alpha\in\Sigma} \mathrm{Var}\Big[\hat{X}_\alpha \mid \overline{h}\Big] &\leqslant \left(\delta_2 + \frac{2f}{3}\cdot(1+\delta_3)\right)|\Sigma|f^2 + \mu + 6 \\
&\leqslant \big(7/6 + \delta_2 f + \delta_3\cdot 2f^2/3\big)\,\mu + 6\,.
\end{aligned}$$

Finally,

$$\begin{aligned}\delta_2 f + \delta_3 \cdot 2f^2/3 &= f\sqrt{6\ln(s_{sec}/p)/(f^2|\Sigma|)} + f^2\sqrt{18\ln(s_{sec}/p)/(f^3|\Sigma|)} \cdot 2/3 \\ &= \sqrt{6\ln(s_{sec}/p)/|\Sigma|} + \sqrt{18f\ln(s_{sec}/p)/|\Sigma|} \cdot 2/3 \\ &= \sqrt{6\ln(s_{sec}/p)/|\Sigma|} + \sqrt{8f\ln(s_{sec}/p)/|\Sigma|} \\ &\leqslant (2+\sqrt{6}) \cdot \sqrt{\ln(s_{sec}/p)/|\Sigma|}\end{aligned}$$

and thus $\sum_\alpha \operatorname{Var}\left[\hat{X}_\alpha \mid \overline{h}\right]$ will be roughly $\mu \cdot 7/6$ whenever μ is large and $\ln(s_{sec}/p) \ll \mu$. □

With the bound on $\sum \operatorname{Var}\left[\hat{X}_\alpha \mid \overline{h}\right]$ in place, we can prove Theorem 12.

Theorem 12 (3 layers with Bernstein)**.** *Assume that* $\ln(s_{sec}/p) \leqslant \bar{\mu}_3$*. Then*

$$\Pr\left[S_{\leqslant 3} \leqslant \mathrm{E}\big[S_{\leqslant 3} \mid \overline{h}\big] - \sqrt{\frac{7}{3}\ln(1/p)\mu \cdot (1+\varepsilon_3)} - \ln(1/p)\right] < (1 + 4/s_{sec}) \cdot p + 4\Pr[\neg\mathcal{J}]\ .$$

Proof. Let $\varepsilon = (2+\sqrt{6})\sqrt{\ln(s_{sec}/p)/|\Sigma|} + 6/\mu$ as defined in Lemma 20. As $\left|\hat{X}_\alpha\right| \leqslant 3$, Bernstein's (eq. (14)) takes the following form when conditioning on $\sum_{\alpha\in\Sigma} \operatorname{Var}\left[\hat{X}_\alpha \mid \overline{h}\right] \leqslant \mu' = (7/6+\varepsilon)\,\mu$,

$$\Pr\left[S_{\leqslant 3} \leqslant \mathrm{E}\big[S_{\leqslant 3} \mid \overline{h}\big] - t \;\middle|\; \overline{h},\ \sum_{\alpha\in\Sigma} \operatorname{Var}\left[\hat{X}_\alpha \mid \overline{h}\right] \leqslant \mu'\right] \leqslant \exp\left(\frac{-0.5t^2}{\mu'+t}\right)\ .$$

Solving for t we find

$$\begin{aligned} t &\geqslant \sqrt{2\ln(1/p)\left(\mu' + \frac{1}{2}\ln(1/p)\right)} + \ln(1/p) \\ &\implies \exp\left(\frac{-0.5t^2}{\mu'+t}\right) \leqslant p\,. \end{aligned}$$

As $\sqrt{\frac{7}{3}\ln(1/p)(1+\varepsilon_3)\mu} + \ln(1/p) \geqslant \sqrt{2\ln(1/p)\left(\mu' + \frac{1}{2}\ln(1/p)\right)} + \ln(1/p)$, the theorem follows when we add the probability that $\sum_\alpha \operatorname{Var}\left[\hat{X}_\alpha \mid \overline{h}\right] > \mu'$. □

We prove Theorem 13 in the same way, with $\bar{X}_\alpha = \min\{2, |X_\alpha|\}$, $S_{\leqslant 2} = S_1 + S_2 = \sum_\alpha \bar{X}_\alpha$ and the following bound on $\sum \operatorname{Var}\big[\bar{X}_\alpha \mid \overline{h}\big]$.

Lemma 21. *If* $\ln(s_{sec}/p) \leqslant |\Sigma| f^2/2 = \bar{\mu}_2$*,*

$$\Pr\left[\sum_{\alpha\in\Sigma} \operatorname{Var}\big[\bar{X}_\alpha \mid \overline{h}\big] > (1+\varepsilon)\,\mu\right] < p \cdot 2/s_{sec} + 2\Pr[\neg\mathcal{J}]$$

where $\varepsilon = \sqrt{6\ln(s_{sec}/p)/|\Sigma|} + 2/\mu$*.*

Proof. The proof proceeds in the same way as that of Lemma 20. Define $\bar{\mu} = \sum_\alpha \mathrm{E}\big[\bar{X}_\alpha \mid \overline{h}\big] = \mathrm{E}\big[S_{\leqslant 2} \mid \overline{h}\big]$ and $\bar{f} = \bar{\mu}/|\Sigma|$. Then

$$\begin{aligned}\sum_{\alpha\in\Sigma} \mathrm{Var}\big[\bar{X}_\alpha \mid \overline{h}\big] &\leqslant \sum_{\alpha\in\Sigma} \mathrm{E}\big[\bar{X}_\alpha(\bar{X}_\alpha - 1) \mid \overline{h}\big] + \sum_{\alpha\in\Sigma} \mathrm{E}\big[\bar{X}_\alpha \mid \overline{h}\big] - |\Sigma|\bar{f}^2 \\ &= 2\cdot \mathrm{E}\big[S_2 \mid \overline{h}\big] + \bar{\mu} - |\Sigma|\bar{f}^2\,.\end{aligned}$$

By Corollary 11 and Lemma 19, for $\delta = \sqrt{3\ln(s_{sec}/p)/\bar{\mu}_2} = \sqrt{6\ln(s_{sec}/p)/(f^2|\Sigma|)}$ and $\mu^+ = (1+\delta)|\Sigma|f^2/2 + 1$,

$$\Pr\big[\mathrm{E}\big[S_2 \mid \overline{h}\big] > \mu^+ \wedge \mathcal{J}\big] \leqslant p\cdot 2/s_{sec} + 2\Pr[\neg\mathcal{J}]\,.$$

Hence, with probability at least $1 - p\cdot 2/s_{sec} - 2\Pr[\neg\mathcal{J}]$ we have

$$\begin{aligned}\sum_{\alpha\in\Sigma} \mathrm{Var}\big[\bar{X}_\alpha \mid \overline{h}\big] &< 2\mu^+ + \bar{\mu} - |\Sigma|\bar{f}^2 \\ &= (1+\delta)|\Sigma|f^2 + \bar{\mu} - |\Sigma|\bar{f}^2 + 2 \\ &\leqslant (1+\delta f)\mu + 2\,.\end{aligned}$$

Finally,

$$\begin{aligned}\delta f &= f\sqrt{6\ln(s_{sec}/p)/(f^2|\Sigma|)} \\ &= \sqrt{6\ln(s_{sec}/p)/|\Sigma|}\end{aligned}$$

and the lemma follows. □

Theorem 13 (2 layers with Bernstein)**.** *Assume that* $\ln(s_{sec}/p) \leqslant \bar{\mu}_2$*. Then*

$$\Pr\left[S_{\leqslant 2} \leqslant \mathrm{E}\big[S_{\leqslant 2} \mid \overline{h}\big] - \sqrt{2\ln(1/p)\mu\cdot(1+\varepsilon_2)} - \frac{2}{3}\ln(1/p)\right] < (1 + 2/s_{sec})\cdot p + 2\Pr[\neg\mathcal{J}]\,,$$

where $\varepsilon_2 = \sqrt{6\ln(s_{sec}/p)/|\Sigma|} + (\frac{2}{9}\ln(1/p) + 2)/\mu$*.*

Proof. Let $\varepsilon = \sqrt{6\ln(s_{sec}/p)/|\Sigma|} + 2/\mu$ as defined in Lemma 21. As $\big|\bar{X}_\alpha\big| \leqslant 2$, Bernstein's (eq. (14)) takes the following form when conditioning on $\sum_{\alpha\in\Sigma}\mathrm{Var}\big[\bar{X}_\alpha \mid \overline{h}\big] \leqslant \mu' = (1+\varepsilon)\,\mu$,

$$\Pr\left[S_{\leqslant 2} \leqslant \mathrm{E}\big[S_{\leqslant 2} \mid \overline{h}\big] - t \;\middle|\; \overline{h},\ \sum_{\alpha\in\Sigma}\mathrm{Var}\big[\bar{X}_\alpha \mid \overline{h}\big] \leqslant \mu'\right] \leqslant \exp\left(\frac{-0.5t^2}{\mu' + t\cdot 2/3}\right)\,.$$

Solving for t we find

$$\begin{aligned}t &\geqslant \sqrt{2\ln(1/p)\left(\mu' + \frac{2}{9}\ln(1/p)\right)} + \frac{2}{3}\ln(1/p) \\ &\implies \exp\left(\frac{-0.5t^2}{\mu' + t\cdot 2/3}\right) \leqslant p\,.\end{aligned}$$

The theorem follows when we add the probability that $\sum_\alpha \mathrm{Var}\big[\bar{X}_\alpha \mid \overline{h}\big] > \mu'$. □

5.4 Regular Layers: Proof of Theorem 14

In order to prove Theorem 14 we first need the following lemmas bounding the difference between S_i and its conditional expectation.

Lemma 22. *Let* $\delta = \sqrt{3\ln(1/p)/\bar{\mu}_i}$. *If* $\ln(1/p) \leqslant \bar{\mu}_i$ *then*

$$\Pr\big[\mathrm{E}\big[S_i \mid \overline{h}\big] \geqslant (1+\delta)\bar{\mu}_i + 1\big] \leqslant 2p + 2\Pr[\neg\mathcal{J}]\,.$$

Proof. By the assumption of the theorem $\delta \leqslant \sqrt{3}$ and thus $\left(e^{\delta}/(1+\delta)^{(1+\delta)}\right) \leqslant \exp(-\delta^2/3)$. It follows from Lemma 19 that,

$$\Pr[S_i \geqslant (1+\delta)\bar{\mu}_i \wedge \mathcal{J}] \leqslant p\,.$$

By Corollary 11

$$\Pr\big[\mathrm{E}\big[S_i \mid \overline{h}\big] \geqslant (1+\delta)\bar{\mu}_i + 1\big] \leqslant 2p + 2\Pr[\neg\mathcal{J}]\,.$$

□

Lemma 23. *Assume that* $\ln(s_{sec}/p_i) \leqslant \bar{\mu}_i$. *Then*

$$\Pr\Big[S_i \leqslant \mathrm{E}\big[S_i \mid \overline{h}\big] - \sqrt{2\ln(1/p_i)(\bar{\mu}_i\cdot(1+\varepsilon_i)+1)}\Big] < (1+2/s_{sec})\cdot p_i + 2\Pr[\neg\mathcal{J}]$$

where $\varepsilon_i = \sqrt{3\ln(s_{sec}/p_i)/\bar{\mu}_i}$.

Proof. As $\ln(s_{sec}/p_i) \leqslant \bar{\mu}_i$ we can apply Lemma 22 with $p = p_i/s_{sec}$ and get that

$$\Pr\big[\mathrm{E}\big[S_i \mid \overline{h}\big] \geqslant (1+\varepsilon_i)\bar{\mu}_i + 1\big] < 2/s_{sec}\cdot p_i + 2\Pr[\neg\mathcal{J}]\,.$$

Let $\mu^+ = (1+\varepsilon_i)\bar{\mu}_i + 1$ and $\delta' = \sqrt{2\ln(1/p_i)/\mu^+}$. When conditioned on $\overline{h}$, S_i is a sum of independent 0/1-variables (with unknown, non-identical distributions). Conditioning on $\mathrm{E}\big[S_i \mid \overline{h}\big] < \mu^+$, we thus have

$$\Pr\big[S_i < \mathrm{E}\big[S_i \mid \overline{h}\big] - \delta'\mu^+ \mid \overline{h},\, \mathrm{E}\big[S_i \mid \overline{h}\big] < \mu^+\big] < \exp\left(-\mu^+\delta'^2/2\right) = p_i\,.$$

As this bound holds for all realizations of $\overline{h}$ where $\mathrm{E}\big[S_i \mid \overline{h}\big] < \mu^+$ the bound also holds without conditioning on $\overline{h}$:

$$\Pr\big[S_i < \mathrm{E}\big[S_i \mid \overline{h}\big] - \delta'\mu^+ \mid \mathrm{E}\big[S_i \mid \overline{h}\big] < \mu^+\big] < \exp\left(-\mu^+\delta'^2/2\right) = p_i\,.$$

Combining the pieces,

$$\begin{aligned}
\Pr\big[S_i \leqslant \mathrm{E}\big[S_i \mid \overline{h}\big] - \delta'\mu^+\big] &\leqslant \Pr\big[\big(S_i \leqslant \mathrm{E}\big[S_i \mid \overline{h}\big] - \delta'\mu^+\big) \wedge \big(\mathrm{E}\big[S_i \mid \overline{h}\big] < \mu^+\big)\big] + \Pr\big[\mathrm{E}\big[S_i \mid \overline{h}\big] \geqslant \mu^+\big] \\
&\leqslant \Pr\big[S_i \leqslant \mathrm{E}\big[S_i \mid \overline{h}\big] - \delta'\mu^+ \mid \mathrm{E}\big[S_i \mid \overline{h}\big] < \mu^+\big] + 2/s_{sec}\cdot p_i + 2\Pr[\neg\mathcal{J}] \\
&\leqslant (1+2/s_{sec})\cdot p_i + 2\Pr[\neg\mathcal{J}]\,.
\end{aligned}$$

The lemma follows as $\delta'\mu^+ = \sqrt{2\ln(1/p_i)\cdot((1+\varepsilon_i)\bar{\mu}_i+1)}$. □

We are now ready to prove Theorem 14.

Theorem 14.

$$\Pr\left[\sum_{i=4}^{i_{nr}-1} S_i < \sum_{i=4}^{i_{nr}-1} \mathrm{E}\big[S_i \mid \overline{h}\big] - \Delta_{reg}\right] < (1.59 + 1/s_{all}) \cdot (1 + 2/s_{sec}) \cdot p + (i_{nr} - 4) \cdot 2\Pr[\neg\mathcal{J}] \,.$$

Proof. We define $p_4 = p$ and $p_{i+1} = \max\{p_i/e, p_{reg}\}$ as discussed in Section 5.1. Let $\varepsilon_i = \sqrt{3\ln(s_{sec}/p_i)/\bar{\mu}_i}$ and $\Delta_i = \sqrt{2\ln(1/p_i)\cdot((1+\varepsilon_i)\bar{\mu}_i + 1)}$. By Lemma 23, for all $i \in \{4, \ldots, i_{nr}-1\}$,

$$\Pr\big[S_i < \mathrm{E}\big[S_i \mid \overline{h}\big] - \Delta_i\big] < (1 + 2/s_{sec}) \cdot p_i + 2\Pr[\neg\mathcal{J}] \,.$$

The theorem follows when we have shown that $\sum_{i=4}^{i_{nr}-1} \Delta_i \leqslant \Delta_{reg}$ and $\sum_{i=4}^{i_{nr}-1} p_i \leqslant (1.59 + 1/s_{all}) \cdot p$. We start with the latter, and recall that $p_{reg} \leqslant \frac{p}{s_{all}\cdot(i_{nr}-4)}$ as discussed in Section 5.1. Hence

$$\sum_{i=4}^{i_{nr}-1} p_i \leqslant (i_{nr} - 4) \cdot p_{reg} + \sum_{k=0}^{\infty} \frac{p}{e^k} \leqslant \left(\frac{1}{s_{all}} + 1.59\right) \cdot p \,.$$

To bound the sum of Δ_i's we distinguish between three cases:
First, assume $i_{nr} = 5$. We thus have to show that $\Delta_4 \leqslant \Delta_{reg}$. As $\varepsilon_i \leqslant \sqrt{3}$,

$$\Delta_4 \leqslant \sqrt{2\ln(1/p)\cdot((1+\sqrt{3})\bar{\mu}_4 + 1)} \leqslant 0.169\sqrt{\ln(1/p)\mu} + \frac{\sqrt{2\ln(1/p)}}{2\sqrt{\bar{\mu}_4}} \leqslant 0.169\sqrt{\ln(1/p)\mu} + \frac{\sqrt{2}}{2} \,.$$

Second, assume $i_{nr} = 6$. Then $\ln(1/p_5) \leqslant \bar{\mu}_5 = \bar{\mu}_4 \cdot f/5$. As $\ln(1/p_4) \leqslant \ln(1/p_5)$ we then have $\varepsilon_4 = \sqrt{3\ln(1/p_4)/\bar{\mu}_4} \leqslant \sqrt{3/10}$.

$$\begin{aligned}
\Delta_4 + \Delta_5 &\leqslant \sqrt{2\ln(1/p_4)\cdot((1+\sqrt{3/10})\bar{\mu}_4 + 1)} + \sqrt{2\ln(1/p_5)\cdot((1+\sqrt{3})\bar{\mu}_5 + 1)} \\
&\leqslant 0.127\sqrt{\ln(1/p_4)\mu} + \frac{\sqrt{2\ln(1/p_4)}}{2\sqrt{\bar{\mu}_4}} + 0.054\sqrt{\ln(1/p_5)\mu} + \frac{\sqrt{2\ln(1/p_5)}}{2\sqrt{\bar{\mu}_5}} \\
&\leqslant 0.181\sqrt{\ln(1/p)\mu} + \frac{0.054\sqrt{\mu}}{\sqrt{\ln(1/p)}} + \sqrt{2} \,.
\end{aligned}$$

Finally, assume $i_{nr} \geqslant 7$. Then $\ln(1/p_6) \leqslant \bar{\mu}_6 = \mu_5 \cdot f/6 = \mu_4 \cdot f^2/30$ and thus $\varepsilon_4 \leqslant \sqrt{3/120}$ while $\varepsilon_5 \leqslant \sqrt{3/12}$. First we bound $\Delta_4 + \Delta_5$, in the same way as in the previous case:

$$\begin{aligned}
\Delta_4 + \Delta_5 &\leqslant \sqrt{2\ln(1/p_4)\cdot((1+\sqrt{3/120})\bar{\mu}_4 + 1)} + \sqrt{2\ln(1/p_5)\cdot((1+\sqrt{3/12})\bar{\mu}_5 + 1)} \\
&\leqslant 0.110\sqrt{\ln(1/p_4)\mu} + \frac{\sqrt{2\ln(1/p_4)}}{2\sqrt{\bar{\mu}_4}} + 0.041\sqrt{\ln(1/p_5)\mu} + \frac{\sqrt{2\ln(1/p_5)}}{2\sqrt{\bar{\mu}_5}} \\
&\leqslant 0.151\sqrt{\ln(1/p)\mu} + \frac{0.041\sqrt{\mu}}{\sqrt{\ln(1/p)}} + \frac{\sqrt{2}}{2}\sum_{i=4}^{5}\sqrt{\frac{\ln(1/p_i)}{\bar{\mu}_i}} \,.
\end{aligned}$$

For $i = 6, \dots, i_{nr} - 1$ we stick with the simple bound $\varepsilon_i \leqslant \sqrt{3}$.

$$\begin{aligned}\sum_{i=6}^{i_{nr}-1} \Delta_i &\leqslant \sum_{i=6}^{i_{nr}-1} \sqrt{2\ln(1/p_i) \cdot ((1+\sqrt{3})\bar{\mu}_i + 1)} \\ &\leqslant \sum_{i=6}^{i_{nr}-1} \sqrt{2\ln(1/p) \cdot (1+\sqrt{3})\bar{\mu}_i} + \frac{(i-4)\sqrt{2(1+\sqrt{3})\bar{\mu}_i}}{2\sqrt{\ln(1/p)}} + \frac{\sqrt{2\ln(1/p_i)}}{2\sqrt{\bar{\mu}_i}} \\ &\leqslant 0.0209\sqrt{\ln(1/p)\mu} + 0.0245\sqrt{\frac{\mu}{\ln(1/p)}} + \frac{\sqrt{2}}{2}\sum_{i=6}^{i_{nr}-1}\sqrt{\frac{\ln(1/p_i)}{\bar{\mu}_i}}\end{aligned}$$

hence

$$\sum_{i=4}^{i_{nr}-1} \Delta_i \leqslant 0.172\sqrt{\ln(1/p)\mu} + 0.066\sqrt{\frac{\mu}{\ln(1/p)}} + \frac{\sqrt{2}}{2}\sum_{i=4}^{i_{nr}-1}\sqrt{\frac{\ln(1/p_i)}{\bar{\mu}_i}}\,.$$

Using that $\bar{\mu}_{i_{nr}-1} \geqslant \ln(1/p_{i_{nr}-1})$ we thus have $\bar{\mu}_{i_{nr}-1-k} \geqslant \bar{\mu}_{i_{nr}-1} \cdot 10^k \geqslant \ln(1/p_{i_{nr}-1}) \cdot 10^k$ when $k \leqslant i_{nr} - 5$, and we can bound the final sum:

$$\frac{\sqrt{2}}{2}\sum_{i=4}^{i_{nr}-1}\sqrt{\frac{\ln(1/p_i)}{\bar{\mu}_i}} \leqslant \frac{\sqrt{2}}{2}\sum_{k=0}^{i_{nr}-5}\frac{1}{\sqrt{10^k}} \leqslant \sqrt{2}\,.$$

Thus we have shown that $\sum_i \Delta_i \leqslant \Delta_{reg}$ in all three cases, which together cover all outcomes. □

5.5 Non-Regular Layers: Proof of Theorem 15

Theorem 15 covers the layers from i_{nr} and up. Define i_∞ to be the first integer i such that $\bar{\mu}_i \leqslant p/s_{all}$. Then Lemmas 25 to 27 below cover all of the non-regular layers, and Theorem 15 is obtained through a union bound over the three lemmas.

Before proving the lemmas, we need the following definition and bound:

Definition 1 (W)**.** The Lambert W function is the function that solves the equation

$$W(x) \cdot \exp(W(x)) = x\,.$$

Lemma 24 (Theorem 2.3 of [HH08])**.** *For $x > 1/e$,*

$$W(x) \leqslant \ln\left(\frac{2x}{\ln(x)+1}\right)\,.$$

Lemma 25.

$$\Pr\big[\mathrm{E}\big[S_{i_{nr}} \,\big|\, \overline{h}\big] > \Delta_{i_{nr}}\big] \leqslant p \cdot 2/s_{all} + 2\Pr[\neg\mathcal{J}]\,.$$

Proof. Let $p_i = p_{reg}/s_{sec} \leqslant p/s_{all}$ such that $\ln(1/p_i) \geqslant \bar{\mu}_{i_{nr}}$ and define $\delta = 1.33e\ln(1/p)/\bar{\mu}_i - 1 > 1.33e - 1$ such that $(1+\delta)\bar{\mu}_i = 1.33e\ln(1/p_i)$. As $1.33 > e^{W(1/e)}$,

$$1.33e\ln(1/p_i) > \ln(1/p_i)/\ln(1.33) = \log_{1.33}(1/p_i)\,.$$

Then, by Lemma 19,

$$\begin{aligned}\Pr[S_{i_{nr}} \geqslant 1.33e\ln(1/p_i) \wedge \mathcal{J}] &\leqslant \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\bar{\mu}_i} \\ &\leqslant \left(\frac{e}{1+\delta}\right)^{(1+\delta)\bar{\mu}_i} \\ &\leqslant \left(\frac{e}{1.33e}\right)^{1.33e\ln(1/p_i)} \\ &< \left(\frac{1}{1.33}\right)^{\log_{1.33}(1/p_i)} = p_i\,.\end{aligned}$$

By Corollary 11, $\Pr\big[\mathrm{E}\big[S_{i_{nr}} \mid \overline{h}\big] > \Delta_{i_{nr}}\big] \leqslant 2p_i + 2\Pr[\neg\mathcal{J}] \leqslant p \cdot 2/s_{all} + 2\Pr[\neg\mathcal{J}]$. □

Lemma 26.

$$\Pr\left[\sum_{i=i_\infty}^{i_{\max}} \mathrm{E}\big[S_i \mid \overline{h}\big] > 3\right] < p \cdot 2/s_{all} + (i_{\max} - i_\infty) \cdot 2\Pr[\neg\mathcal{J}]\ .$$

Proof. Let $i^+ = i - i_\infty$ and set $\varepsilon_i = 2/3^{i^+}$. As $\sum_{i_\infty}^{i_{\max}} \varepsilon_i < \sum_{i=i_\infty}^{\infty} \varepsilon_i = 3$,

$$\Pr\left[\sum_{i=i_\infty}^{i_{\max}} \mathrm{E}\big[S_i \mid \overline{h}\big] \geqslant 3\right] \leqslant \sum_{i=i_\infty}^{i_{\max}} \Pr\big[\mathrm{E}\big[S_i \mid \overline{h}\big] \geqslant \varepsilon_i\big]\ .$$

By Corollary 11 and Markov's inequality

$$\Pr\big[\mathrm{E}\big[S_i \mid \overline{h}\big] \geqslant \varepsilon_i\big] \leqslant 2\Pr[S_i \geqslant \varepsilon_i \wedge \mathcal{J}] + 2\Pr[\neg\mathcal{J}] \leqslant \frac{2\mathrm{E}[S_i \cdot [\mathcal{J}]]}{\varepsilon_i} + 2\Pr[\neg\mathcal{J}] \leqslant \frac{2\bar{\mu}_i}{\varepsilon_i} + 2\Pr[\neg\mathcal{J}]\ .$$

Using that $\bar{\mu}_i \leqslant \bar{\mu}_{i_\infty}/6^{i^+}$ we thus have

$$\begin{aligned}\sum_{i=i_\infty}^{i_{\max}} \Pr\left[\mathrm{E}\big[S_i \mid \overline{h}\big] \geqslant \frac{2}{3^{i^+}}\right] &\leqslant \bar{\mu}_{i_\infty} \cdot \sum_{k=0}^{i_{\max}} \left(\frac{3}{6}\right)^k + (i_{\max} - i_\infty) \cdot 2\Pr[\neg\mathcal{J}] \\ &\leqslant p \cdot 2/s_{all} + (i_{\max} - i_\infty) \cdot 2\Pr[\neg\mathcal{J}]\ .\end{aligned}$$

□

Lemma 27.

$$\Pr\left[\sum_{i=\max\{i_{nr}+1.\,4\}}^{i_\infty} \mathrm{E}\big[S_i \mid \overline{h}\big] \geqslant \Delta_{top}\right] \leqslant p \cdot 2/s_{all} + (i_\infty - i_{nr}) \cdot 2\Pr[\neg\mathcal{J}]\ .$$

Lemma 27 is obtained by applying the following lemma on each layer between i_{nr} and i_∞.

Lemma 28. *For $i > i_{nr}$ and $p_i \leqslant p_{reg}/s_{sec}$,*

$$\Pr\left[S_i \geqslant \frac{2\ln(1/p_i)}{(i - i_{nr}) \cdot \ln(i/ef)} \wedge \mathcal{J}\right] \leqslant p_i\,.$$

Proof of Lemma 28. Define $i^+ = i - i_{nr}$ and let $k = (1+\delta)\bar{\mu}_i$ for some $\delta > 0$. By Lemma 19

$$\Pr[S_i > k \wedge \mathcal{J}] \leqslant \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^{\bar{\mu}_i} < \left(\frac{e}{(1+\delta)}\right)^{(1+\delta)\bar{\mu}_i} = \left(\frac{e\bar{\mu}_i}{k}\right)^k .$$

As $\bar{\mu}_i = \bar{\mu}_{i_{nr}} \cdot \frac{f^{i^+} i_{nr}!}{i!}$ and $\bar{\mu}_{i_{nr}} < \ln(s_{sec}/p_{reg}) \leqslant \ln(1/p_i)$ we have

$$\Pr[S_i > k \wedge \mathcal{J}] \leqslant \left(\frac{e\bar{\mu}_i}{k}\right)^k \leqslant \left(\frac{e\ln(1/p_i)}{k} \cdot \frac{f^{i^+} i_{nr}!}{i!}\right)^k .$$

By Stirling's approximation,

$$\frac{i_{nr}!}{i!} \leqslant e^{i^+} \cdot \frac{(i_{nr})^{i_{nr}}}{i^i} = e^{i^+} \cdot \left(\frac{i_{nr}}{i}\right)^{i_{nr}} \cdot \frac{1}{i^{i^+}} \leqslant \left(\frac{e}{i}\right)^{i^+}$$

and thus

$$\Pr[S_i \geqslant k \wedge \mathcal{J}] \leqslant \left(\frac{e\ln(1/p_i)}{k} \cdot \left(\frac{ef}{i}\right)^{i^+}\right)^k .$$

Define $k = \frac{2\ln(1/p_i)}{i^+ \ln(i/ef)}$ and

$$\eta = \frac{2}{e} \cdot \frac{(i/ef)^{i^+}}{i^+ \ln(i/ef)} = \frac{2(i/ef)^{i^+}/e}{\ln\left((i/ef)^{i^+}\right)} = \frac{2(i/ef)^{i^+}/e}{\ln\left((i/ef)^{i^+}/e\right) + 1} \geqslant \exp\left(W\left((i/ef)^{i^+}/e\right)\right) ,$$

with the final inequality due to Lemma 24, as $i/ef \geqslant 1$ when $i \geqslant 2$. Hence $\eta\ln(\eta) \geqslant (i/ef)^{i^+}/e$ or, equivalently, $\frac{\eta \cdot e}{(i/ef)^{i^+}} \geqslant 1/\ln(\eta)$. Observe that $k = \ln(1/p_i) \cdot \frac{\eta \cdot e}{(i/ef)^{i^+}}$ and thus

$$\begin{aligned}
\Pr[S_i \geqslant k \wedge \mathcal{J}] &\leqslant \left(\frac{e\ln(1/p_i)}{k} \cdot \left(\frac{ef}{i}\right)^{i^+}\right)^k \\
&\leqslant \left(\frac{e}{2} \cdot i^+ \ln(i/ef) \cdot \left(\frac{ef}{i}\right)^{i^+}\right)^k \\
&\leqslant \left(\frac{1}{\eta}\right)^{\frac{\ln(1/p_i)}{\ln(\eta)}} = \left(\frac{1}{\eta}\right)^{\log_\eta(1/p_i)} = p_i .
\end{aligned}$$

□

We can now prove Lemma 27:

Proof of Lemma 27. First, note that $i_\infty - i_{nr} \leqslant n_{top} = \log_6(\ln(s_{sec}/p_{reg}) \cdot s_{all}/p)$. This bound comes from the fact that $\bar{\mu}_{i_{nr}} \leqslant \ln(s_{sec}/p_{reg})$ while $\bar{\mu}_{i_\infty - 1} \geqslant p/s_{all}$ and $\bar{\mu}_{i+1} \leqslant \bar{\mu}_i/6$ for all $i \geqslant 2$.

For the i'th layer let $k_i = \ln(1/p_{top}) \cdot \frac{2}{i^+ \ln(i/ef)} + 1$ where $i^+ = i - i_{nr}$. By Lemma 28 and Corollary 11, $\Pr\big[\mathrm{E}\big[S_i \,\big|\, \overline{h}\big] > k_i\big] < 2p_{top} + 2\Pr[\neg\mathcal{J}]$. As $p_{top} \leqslant p/(n_{top} \cdot s_{all})$, we have

$$\sum_{\max\{i_{nr}+1,\,4\}}^{i_\infty} \Pr\big[\mathrm{E}\big[S_i \,\big|\, \overline{h}\big] > k_i\big] \leqslant p \cdot 2/s_{all} + (i_\infty - i_{nr}) \cdot 2\Pr[\neg\mathcal{J}] .$$

Left is to show that $\sum k_i \leqslant \Delta_{top}$. As $i \geqslant 4$ we have $\ln(i/ef) > 0$ and the k_i's are decreasing. Their sum can thus be bounded by a definite integral:

$$\begin{aligned}
\sum_{i=\max\{i_{nr}+1,\,4\}}^{i_\infty} \frac{2}{i^+ \ln(i/ef)} &\leqslant \sum_{j=1}^{i_\infty - i_{nr}} \frac{2}{j \ln((j+3)/ef)} \\
&\leqslant \sum_{j=1}^{4} \frac{2}{j \ln(4/ef)} + \sum_{j=5}^{i_\infty - i_{nr}} \frac{2}{j \ln(j/ef)} \\
&\leqslant \frac{25/6}{\ln(4/ef)} + \int_4^{i_\infty - i_{nr}} \frac{2}{x \ln(x/ef)}\,\mathrm{d}x \\
&= \frac{25/6}{\ln(4/ef)} + \int_{4/ef}^{(\inf - i_{nr})/ef} \frac{2}{u \ln(u)}\,\mathrm{d}u \\
&= \frac{25/6}{\ln(4/ef)} + 2\left(\ln\ln((i_\infty - i_{nr})/ef) - \ln\ln(4/ef)\right) \\
&\leqslant 3.9 + 2\ln\ln(n_{top})\,.
\end{aligned}$$

Hence

$$\sum_{i=\max\{i_{nr}+1,4\}}^{i_\infty} k_i \leqslant 2\ln(1/p_{top}) \cdot (2 + \ln\ln(n_{top})) + n_{top}\,.$$

□

Finally, we are now ready to prove Theorem 15

Theorem 15. *If $i_{nr} \geqslant 3$, then*

$$\Pr\left[\sum_{i=i_{nr}}^{i_{\max}} S_i < \sum_{i=i_{nr}}^{i_{\max}} \mathrm{E}\big[S_i \mid \overline{h}\big] - \Delta_{nonreg}\right] < p \cdot 6/s_{all} + (i_{\max} - i_{nr}) \cdot 2\Pr[\neg\mathcal{J}]\,.$$

Proof. Each S_i is non-negative, and thus $\Pr\big[S_i < \mathrm{E}\big[S_i \mid \overline{h}\big] - k\big] \leqslant \Pr\big[\mathrm{E}\big[S_i \mid \overline{h}\big] > k\big]$ for any k. By Lemmas 25 to 27, we thus have

$$\begin{aligned}
\Pr\left[\sum_{i=i_{nr}}^{i_{\max}} S_i < \sum_{i=i_{nr}}^{i_{\max}} \mathrm{E}\big[S_i \mid \overline{h}\big] - \Delta_{nonreg}\right] &\leqslant \Pr\left[\sum_{i=i_{nr}}^{i_{\max}} \mathrm{E}\big[S_i \mid \overline{h}\big] > \Delta_{nonreg}\right] \\
&< p \cdot 6/s_{all} + (i_{\max} - i_{nr}) \cdot 2\Pr[\neg\mathcal{J}]
\end{aligned}$$

as $\Delta_{nonreg} = \Delta_{i_{nr}} + \Delta_{top} + 3$. □

5.6 No Big Layers

We finally prove Lemma 16 which handles layers beyond $i_{\max}$. For this, we require our new Theorem 17 for bounding the probability that the selected set is linearly dependent. We postpone the proof of this theorem to Section 6.

Lemma 16. *Let $s \leqslant |\Sigma|/2$ be the number of selection bits and $i_{\max} = \ln(|\Sigma|^{d-2}2^s)$. Then*

$$\Pr\left[\sum_{i=i_{\max}+1}^{\infty} \mathrm{E}\big[S_i \mid \overline{h}\big] > 0\right] \leqslant \left(\frac{1}{|\Sigma|}\right)^{d-3} + 3^{c+1}\left(\frac{3}{|\Sigma|}\right)^{d-1} + \left(\frac{1}{|\Sigma|}\right)^{|\Sigma|/2-1}.$$

Proof. Note that $\{S_i\}_{i\in\mathbb{N}}$ is a non-increasing sequence. Hence $\big\{\mathrm{E}\big[S_i \mid \overline{h}\big]\big\}_{i\in\mathbb{N}}$ is non-increasing, and it suffices to prove that $\Pr\big[\mathrm{E}\big[S_i \mid \overline{h}\big] > 0\big]$ is small for $i = i_{\max}+1$.

Let $\texttt{select} : \Sigma^c \times [2]^s \to [2]$ be the selector function defining the set X (referred to as 'f' in Section 3). For each character $\alpha \in \Sigma$ and value $r \in [2]^s$ of the selection bits, define

$$X_{\alpha,r} = \big\{x \in S \mid \texttt{select}(x, \overline{h}[0](x) \oplus r) = 1 \wedge \overline{h}[1](x) = \alpha\big\}.$$

That is, $X_{\alpha,r}$ is the set of keys with final derived character α which will be selected if $T_{c+d}[\alpha] = r$. Note that $X_{\alpha,r}$ is completely determined by $\overline{h}$. When conditioning on $\overline{h}$, X_α is thus uniformly distributed among the values $\{X_{\alpha,r}\}_{r\in[2]^s}$. We will show that, w.h.p., $|X_{\alpha,r}| \leqslant i_{\max}$ for all r. This entails that $|X_\alpha| \leqslant i_{\max}$, in turn giving that $\mathrm{E}\big[S_i \mid \overline{h}\big] = 0$.

Recall that, for each element $x \in A$, there exists exactly one value r' such that $\texttt{select}(x, r') = 1$. Thus the sets $X_{\alpha,r}$ partition A, and each x is distributed uniformly among the sets. As the expected number of selected elements is $|A|/2^s = \mathrm{E}[|X|] = f|\Sigma|$, the expected size of each set will be $\mathrm{E}[|X_{\alpha,r}|] = |A|/(|\Sigma|2^s) = f \leqslant 1/2$.

As $\overline{h}$ is a tornado tabulation function (albeit with only $d-1$ derived characters), we can invoke Theorem 4 to bound the probability of $X_{\alpha,r}$ being large. Setting $(1+\delta) = 1/f \cdot \ln(|\Sigma|^{d-2}2^s) > e^2$ we obtain

$$\begin{aligned}\Pr\Big[|X_{\alpha,r}| \geqslant \ln\left(|\Sigma|^{d-2}2^s\right) \wedge \mathcal{I}\left(\widetilde{h}_{<c+d}(X_{\alpha,r})\right)\Big] &< \left(\frac{e}{1+\delta}\right)^{(1+\delta)f} \\ &= \left(\frac{1}{e}\right)^{\ln(|\Sigma|^{d-2}2^s)} \\ &= \frac{1}{|\Sigma|^{d-2}2^s}.\end{aligned}$$

By Theorem 17 we further have, plugging in expected size $\mu/|\Sigma| = f$, ratio 'f' of $1/(2|\Sigma|)$, and $d-1$ derived characters,

$$\Pr\Big[\neg\mathcal{I}\left(\widetilde{h}_{<c+d}(X_{\alpha,r})\right)\Big] \leqslant \frac{3^c|\Sigma|}{n} \cdot 3\left(\frac{\mu}{|\Sigma|}\right)^3\left(\frac{3}{|\Sigma|}\right)^d + \left(\frac{1}{2|\Sigma|}\right)^{|\Sigma|/2}.$$

Adding the two error probabilites, and performing a union bound over all $|\Sigma| \cdot 2^s$ sets $X_{\alpha,r}$, the

statement follows:

$$\begin{aligned}
\Pr\Big[\exists X_{\alpha,r} : |X_{\alpha,r}| \geqslant \ln\left(|\Sigma|^{d-2}2^s\right)\Big] &\leqslant |\Sigma|2^s \cdot \Pr\Big[|X_{\alpha,r}| \geqslant \ln\left(|\Sigma|^{d-2}2^s\right)\Big] \\
&\leqslant |\Sigma|2^s \left(\frac{1}{|\Sigma|^{d-2}2^s} + \frac{3^c|\Sigma|}{n} \cdot 3\left(\frac{\mu}{|\Sigma|}\right)^3 \left(\frac{3}{|\Sigma|}\right)^d + \left(\frac{1}{2|\Sigma|}\right)^{|\Sigma|/2}\right) \\
&\leqslant \frac{1}{|\Sigma|^{d-3}} + \frac{3^c|\Sigma|^2}{\mu} \cdot 3\left(\frac{\mu}{|\Sigma|}\right)^3 \left(\frac{3}{|\Sigma|}\right)^d + |\Sigma|2^s \left(\frac{1}{2|\Sigma|}\right)^{|\Sigma|/2} \\
&= \frac{1}{|\Sigma|^{d-3}} + 9 \cdot 3^c \left(\frac{\mu}{|\Sigma|}\right)^2 \left(\frac{3}{|\Sigma|}\right)^{d-1} + |\Sigma|2^s \left(\frac{1}{2|\Sigma|}\right)^{|\Sigma|/2} \\
&\leqslant \frac{1}{|\Sigma|^{d-3}} + 3^{c+1}\left(\frac{3}{|\Sigma|}\right)^{d-1} + \left(\frac{1}{|\Sigma|}\right)^{|\Sigma|/2-1},
\end{aligned}$$

where we've used that $\mu = n/2^s$ and assumed that $s \leqslant |\Sigma|/2$. □

6 Proof of Theorem 17

In this section, we describe the main ingredients needed for the proof of Theorem 17 and how they can be combined together.

Theorem 17. *Let $h = \widehat{h} \circ \widetilde{h} : \Sigma^c \to \mathcal{R}$ be a random (simple) tornado tabulation hash function with d derived characters and f as described above. If $\mu \leqslant \Sigma/2$, then the event $\mathcal{I}(\tilde{h}(X))$ fails with probability at most*

$$3^c\,|\Sigma|\,/n \cdot 3\mu^3(3/|\Sigma|)^{d+1} + f^{|\Sigma|/2} \tag{13}$$

We first note that Theorem 3 holds for a simpler version of tornado tabulation hashing, which we call *simple tornado hashing*. In this version, we do not change the last character of the (original) key. Formally, for a key $x = x_1 \cdots x_c$, its corresponding derived key $\widetilde{x} = \widetilde{x}_1 \ldots \widetilde{x}_{c+d}$ is computed as

$$\widetilde{x}_i = \begin{cases} x_i & \text{if } i = 1, \ldots, c \\ \widetilde{h}_{i-c}\left(\widetilde{x}_1 \ldots \widetilde{x}_{i-1}\right) & \text{otherwise.} \end{cases}$$

The main idea is to revisit the proof of Theorem 3 in [BBK+23]. Note that, if the set of derived keys in $\tilde{h}(X)$ are linearly dependent, then their prefixes are also linearly dependent (i.e., when we consider only the first $c+d-1$ or $c+d-2$ characters). The main idea is then to argue that, if the derived keys in $\tilde{h}(X)$ are linearly dependent, then we can find a certain *obstruction* that captures how the keys remain linearly dependent as we add derived characters one at a time. Each such obstruction in unlikely to occur. By performing a union bound over all such possible obstructions, we then get the bound in Theorem 3.

6.1 Preliminaries

Position Characters, Generalized Keys and Linear Independence We view any key $x \in \Sigma^b$ as a set of b *position characters* $(1, x_1) \ldots (b, x_b)$. We can then define the symmetric difference of two keys as being the symmetric difference of the corresponding sets of position characters. A

generalized key can be any subset of position characters $\{1 \dots b\} \times \Sigma$. For such a generalized key x, we can then define

$$x[i] = \{(i, a) \in x\}, \quad x[< i] = \{(j, a) \in x \mid j < i\}$$

and

$$x[\leqslant i] = \{(j, a) \in x \mid j \leqslant i\}.$$

This also extends naturally to any set X of generalized keys, i.e.,

$$X[< i] = \{x[< i] \mid x \in X\}.$$

When it comes to defining linear independence over a set Y of generalized keys, we can define $\triangle Y$ to be the symmetric difference of all the subsets of position characters, i.e., the set of position characters that appear an odd number of times in the subset in Y. If $\triangle Y$ is the empty set (and hence, every position character appears an even number of times), we say that the set Y is a *zero-set*. If Y contains a zero-set, then we say that Y is *linearly dependent*. Otherwise, we say that the (generalized) keys in Y are *linearly independent*.

Levels and Matching For the sake of consistency, we follow the setup in [BBK+23]. The idea is to bound the probability that the keys in $\tilde{h}(X)$ are dependent with respect to each derived character separately.

To this end, for $i = 1, \dots, d$, we focus on position $c+i$ of a derived key and refer to such positions at being at *level* i. Linear dependence in the derived keys means that, for each level, we can pair up derived keys that have the same derived character at that level. Formally, we say that a matching $M \subseteq \binom{|\Sigma|^c}{2}$ on the keys Σ^c is an *i-matching* if for all $\{x, y\} \in M$, it holds that $\widetilde{x}[c+i] = \widetilde{y}[c+i]$. We further say that such a matching is an *i-zero, i-dependent, or i-independent* if the corresponding

$$\text{DiffKeys}(M, i) = \{(\widetilde{x} \triangle \widetilde{y})[\leqslant c + i] \mid \{x, y\} \in M\}$$

is a zero-set, linearly dependent, or linearly independent, respectively. Similarly, we say that a set Z is of (original) keys is *i-zero, i-dependent, or i-independent* if the set of prefixes $\widetilde{Z}[\leqslant c + i]$ is a zero-set, linearly dependent, or linearly independent, respectively, where $\widetilde{Z}$ denotes the set of derived keys of keys in Z. The following observation from [BBK+23] connects the notions:

Observation 29 (Observation 11 in [BBK+23])**.** Let M be a partial matching on Σ^c and $Z = \bigcup M$. Then M is an i-zero matching iff Z is an i-zero set. Furthermore, if M is i-dependent then Z is also i-dependent (but not vice versa).

Moreover, when moving from one level to the next, we will use the following observation:

Observation 30 (Observation 12 in [BBK+23])**.** If Z is an i-zero set, then there is a perfect j-matching on Z for every level $j \leqslant i$.

The obstructions we build will consist of matching at each level. To bound the probability that such an obstruction exists, we will use the following bound repeatedly:

Lemma 31 (Lemma 10 in [BBK+23])**.** *Let M be a partial matching on Σ^c. Conditioning on M being $(i-1)$-independent, M is an i-matching with probability $1/|\Sigma|^{|M|}$.*

6.2 Defining an Obstruction on the Top Two Levels

We distinguish between the top two levels d and $d-1$, and the remaining bottom levels $1, \ldots, d-2$. The obstruction on the top two levels is defined similarly to how it is defined in [BBK+23]. Namely: if a set of derived keys $\widetilde{X}$ is linearly dependent, then it must be the case that there exists a mininal subset $Z \subseteq X$ that is a d-zero set ([BBK+23] had some special concerns about query keys, but these query keys are not considered here).

By Observation 30, the set $\widetilde{Z}$ exhibits a (perfect) d-matching M_d^* and a (perfect) $(d-1)$-matching M_{d-1}^* o(we also have perfect matchings on all the other levels). We follow the edges of these two matching in order to build our obstruction. Namely, these two matchings form alternating cycles on the keys in Z. For every such cycle, we choose an arbitrary start vertex x_1 and follow the edge from M_{d-1}^*. We land at some other vertex x_2 and the follow the edge from M_d^*, and so on an so forth. When we are done with one cycle, we continue with the next one in a similar fashion. We end up with a sequence of vertices $x_1, \ldots, x_{|Z|}$ that describe all vertices in Z such that edges $\{x_1, x_2\}, \{x_3, x_4\}, \ldots, \{x_{|Z|-1}, x_{|Z|}\}$ describe the edges in M_{d-1}^*.

Among the edges in M_{d-1}^*, we now identify a minimal $(d-2)$-dependent sub-matching M_{d-1} by defining w as the smallest value for which $\{x_1, x_2\}, \ldots, \{x_{w-1}, x_w\}$. We let $W = \{x_1 \ldots x_w\}$ be the support of this sub-matching and note that w is even. We also let M_d be to be the restriction of M_d^* to $\{x_1 \ldots x_{w-1}\}$ (without the last vertex we visit). Note that M_d how has $w/2-1$ edges. We use the following simple lemma that was not part of [BBK+23]:

Lemma 32. *There is exactly one submatching $L_{d-1} \subseteq M_{d-1}$ such that L_{d-1} is a $(d-2)$-zero matching.*

Proof. First, we know L_{d-1} exists because M_{d-1} is $(d-1)$-dependent. Suppose we had an alternative $L'_{d-1} \neq L_{d-1}$. Then $L''_{d-1} = L'_{d-1} \Delta L_{d-1}$ is also a $(d-1)$-zero matching, but since L_{d-1} and L_{d-1} both include $\{x_{w-1}, x_w\}$, L''_{d-1} does not include $\{x_{w-1}, x_w\}$, but this means that $\{x_1, x_2\}, \ldots, \{x_{w-3}, x_{w-2}\}$ is $d-2$-dependent. □

Finally, we define L_{d-1} uniquely as in Lemma 32, and set $Z_{d-1} = \bigcup L_{d-1}$. Then Z_{d-1} is a $(d-2)$-zero set which will play a very crucial role.

The Obstruction on the First Two Levels For the purposes of our argument, we distinguish between two cases depending on w. We define $w_{\max}$ to be the smallest even number above $0.63\,|\Sigma|$. In our obstruction above, we do not want $w > w_{\max}$, so if $w > w_{\max}$, we reduce w to $w_{\max}$. In this case, $M_{d-1} = \{x_1, x_2\}, \ldots, \{x_{w-1}, x_{w_{\max}}\}$ is $(d-2)$-independent with at least $w/2-1$ edges. If $w > w_{\max}$, we say that the obstruction was *truncated*. Otherwise, we say that the obstruction is *complete*. We then define the following obstruction $\mathcal{O} = (W, M_d, M_{d-1}, L_{d-1})$ for the first two levels:

- A set of keys $W \subseteq \Sigma^c$ of some even size w.
- A matching M_d of size $w/2-1$ on W.
- A perfect matching M_{d-1} on W. This matching also contains a $(d-2)$-independent submatching M'_{d-1} with at least $w/2-1$ edges. If $w > w_{\max}$, (the truncated case), $M'_{d-1} = M_{d-1}$. Otherwise, M'_{d-1} is M_{d-1} minus any edge from L_{d-1}.

- If $w \leqslant w_{\max}$ (the complete case), we have a submatching $L_{d-1} \subseteq M_{d-1}$ with support $Z_{d-1} = \bigcup L_{d-1}$ and size less than $w_{\max}$. Here Z_{d-1} should contain at least one vertex not matched by M_d (this corresponds to the vertex x_w that we do not mention explicitly among the components). Note that in the truncated case, we do not store L_{d-1} and Z_{d-1}. In this case, we are satisfied having the $(d-1)$-independent M_d and the $(d-2)$-independent M_{d-1} matching with a total of at least $w-2$ edges.

6.3 Confirming an Obstruction

For an obstruction $\mathcal{O} = (W, M_d, M_{d-1}, L_{d-1})$ to occur among the selected keys, the tornado tabulation hash function $h = \widehat{h} \circ \widetilde{h}$ must satisfy the following conditions:

1. The keys in W are all selected, that is, $W \subseteq X^{f,h}$.
2. Either W is d-independent, or it is minimally d-dependent. A minimally d-dependent W corresponds to the case where $W = Z$.
3. M_d is a d-matching.
4. M_d is $(d-1)$-independent.
5. M_{d-1} is a $(d-1)$-matching.
6. M_{d-1} contains a $(d-2)$-independent submatching M'_{d-1} with at least $w/2-1$ edges.
7. For a complete obstruction, $Z_{d-1} = \bigcup L_{d-1}$ is a $(d-2)$-zero set.

For a given obstruction we use $(1), (2), \ldots$ to denote the event where the tornado tabulation hash function satisfies each of the conditions given above.

When a hash function h satisfies the above conditions, we say that it *confirms* an obstruction, and we want to prove that this happens with small probability. Our probability bound is parameterized by $w = |W|$.

We bound the probability of satisfying all conditions as

$$\Pr\left[\bigcap_{i=1}^{7}(i)\right] \leqslant \Pr[(6) \cap (7)] \cdot \Pr[(5) \mid (6) \cap (7)] \cdot \Pr\left[(3) \,\middle|\, \bigcap_{i>3}(i)\right] \cdot \Pr\left[(1) \,\middle|\, \bigcap_{i>1}(i)\right].$$

For $\Pr[(1) \mid \bigcap_{i>1}(i)]$, by conditioning on (2) we know that at least $w-1$ derived keys are hashed independently by $\hat{h}$. As each is selected with probability p, we get

$$\Pr\left[(1) \,\middle|\, \bigcap_{i>1}(i)\right] \leqslant p^{w-1}.$$

For a truncated obstruction, however, we know that W is a strict subset of Z, and hence W is d-independent, giving

$$\Pr\left[(1) \,\middle|\, \bigcap_{i>1}(i)\right] \leqslant p^{w}.$$

For $\Pr[(3) \mid \bigcap_{i>3}(i)]$, by conditioning on (4) we know that all $|M_d| = w/2 - 1$ diff-keys from M_d are hashed independently by $\widetilde{h}_d$, so

$$\Pr\left[(3) \;\middle|\; \bigcap_{i>3}(i)\right] \leqslant 1/|\Sigma|^{w/2-1}\,.$$

For $\Pr[(5) \mid (6) \cap (7)]]$, by conditioning on (6) there exists a $(d-2)$-independent M'_{d-1} whose keys are hashed independently by $\widetilde{h}_{d-1}$, so the probability of M_{d-1} (and thus also M'_{d-1}) being a $(d-1)$-matching is at most

$$\Pr[(5) \mid (6) \cap (7)] \leqslant 1/|\Sigma|^{w/2-1}\,.$$

Finally, for truncated obstructions we apply the trivial bound $\Pr[(6) \cap (7)] \leqslant 1$, while for complete obstructions we apply Lemma 33, given below, with $|Z_{d-1}| \leqslant |W| = w$ to obtain

$$\Pr[(6) \cap (7)] \leqslant \Pr[(7)] \leqslant (3/|\Sigma|)^{d-2} \cdot 2^{w/4+1}\,.$$

Putting it all together, we obtain the following bounds on h confirming a given obstruction $\mathcal{O} = (W, M_d, M_{d-1}, L_{d-1})$ with $|W| = w$

$$\begin{aligned}\Pr[h \text{ confirms truncated } \mathcal{O}] &\leqslant |\Sigma|^2 \cdot (p/|\Sigma|)^w\\ \Pr[h \text{ confirms complete } \mathcal{O}] &\leqslant p^{w-1}|\Sigma|^{2-w}(3/|\Sigma|)^{d-2}2^{w/4+1}\,.\end{aligned}$$

Lemma 33 ([BBK+23])**.** *If $z_{d-1} = |Z_{d-1}| \leqslant 0.63 \cdot |\Sigma|$ and $|\Sigma| \geqslant 256$ then*

$$\Pr_{\widetilde{h}_{\leqslant d-2}}[Z_{d-1} \text{ is an } (d-2)\text{-zero set}] \leqslant (3/\,|\Sigma|)^{d-2} \cdot 2^{z_{d-1}/4+1}. \tag{15}$$

Proof sketch. The bound is implicitly present in [BBK+23], so we sketch the arguments here and refer to reader to the appropriate sections in [BBK+23]for details. The main idea is to proceed similarly to how we defined L_{d-1} (and Z_{d-1}) from M_{d-1} (Section 3.2 in [BBK+23]). Namely, while going through the matching M_i (which is (minimally) i-dependent), we identify the submatching L_i which is an i-zero matching. This then gives rise to a matching M_{i-1} on Z_{i-1} (which is the support of L_{i-1}. We do this for every layer from $i = d-2$ to $i = 1$. This describes a general obstruction that includes all the matching M_i and supports Z_i(Section 3.3 in [BBK+23]). The probability that an obstruction is confirmed is bound in Lemma 14 in [BBK+23]. The difference with what we have is that the only care about the part of the obstruction that deals with levels $1, \ldots, d-2$ (excluding the top two levels). In particular, the event that Z_{d-1} is a $(d-2)$-zero set corresponds to the conjunction $\bigwedge_{i=1}^{d-2\mathcal{C}^{(i)}}$ over all possible realizations of $M_{d-2}, Z_{d-2}, M_{d-3}$ etc. We get the following:

$$\begin{aligned}\Pr_{\widetilde{h}_{\leqslant d-2}}[Z_{d-1} \text{ is an } (d-2)\text{-zero set}] &\leqslant \prod_{i=1}^{d-2}\max_{Z_{i+1}}\left(\sum_{M_i,e_i,L_i,Z_i}|\Sigma|^{1-|M_i|}\right)\\ &\leqslant 2^{z_{d+1}/4-1}\cdot\prod_{i=1}^{d-2}\max_{Z_{i+1}}\left(\sum_{M_i,e_i,L_i,Z_i}|\Sigma|^{1-|M_i|}\Big/2^{(|Z_{i+1}|-|Z_i|)/4}\right)\,.\end{aligned}$$

In sections 4.2 and 5.1 (specifically Eq (21)), it is shown that:

$$\max_{Z_{i+1}}\left(\sum_{M_i,e_i,L_i,Z_i}|\Sigma|^{1-|M_i|}\Big/2^{(|Z_{i+1}|-|Z_i|)/4}\right) \leqslant 4\cdot(3/|\Sigma|)^{d-2}\,.$$

Since $4(3/\,|\Sigma|)^{d-2}2^{z_{d-1}/4-1} = (3/\,|\Sigma|)^{d-2}2^{z_{d-1}/4+1}$, we get the claim. □

6.4 Union Bounds over All Obstructions

To obtain the bound stated in Theorem 17 we perform a union bound over the probability of confirming each possible obstruction. As we have defined two types of obstructions, we treat these separately.

Truncated obstructions We start with the case where the obstruction has been truncated.

As our bound on the probability of a truncated obstruction is confirmed is identical for all truncated obstructions (they are all of size $|W| = w_{\max}$) we just have to count the number of obstructions $\mathcal{O} = (W, M_d, M_{d-1})$ to obtain the first part of our union bound.

In the following we let $w = w_{\max}$ for improved readability. The set W can be specified in $\binom{n}{w} \leqslant \frac{n^w}{w!}$ ways. The matching M_d of size $w/2 - 1$ over $\{1, \ldots, w\}$ can be described as a perfect matching on W with one edge removed, giving $(w-1)!! \cdot w/2$ possibilities [2]. The matching M_{d-1} is perfect, so it can be chosen in $(w-1)!!$ ways. In total, this means that there exists at most $((w-1)!!)^2 \cdot w/2 \cdot n^w/w!$ choices for $\mathcal{O} = (W, M_d, M_{d-1})$.

We conclude that

$$\begin{aligned}
\Pr[h \text{ confirms a truncated obstruction}] &\leqslant \sum_{\text{truncated } \mathcal{O}} \Pr[h \text{ confirms } \mathcal{O}] \\
&\leqslant \frac{w}{2} \cdot \frac{n^w}{w!} \cdot ((w-1)!!)^2 \cdot \left(\frac{p}{|\Sigma|}\right)^w \cdot |\Sigma|^2 \\
&= f^w \cdot \frac{((w-1)!!)^2}{2(w-1)!} \cdot |\Sigma|^2 \\
&= f^w \cdot \frac{(w-1)!!}{2(w-2)!!} \cdot |\Sigma|^2 ,
\end{aligned}$$

using that $np/|\Sigma| = \mu/|\Sigma| = f$ and $(w-1)! = (w-1)!! \cdot (w-2)!!$. Note that $(w-1)!!/(w-2)!! \leqslant 3/2$ for all $w \geqslant 4$. As $w = w_{\max} \geqslant 0.63|\Sigma|$, we have $f^w \leqslant f^{|\Sigma|/2} \cdot f^{0.13|\Sigma|} \leqslant f^{|\Sigma|/2} \cdot (1/2)^{0.13|\Sigma|}$.

For $|\Sigma| > 100$ we have $\frac{3}{2 \cdot 2} \cdot |\Sigma|^2 \cdot (1/2)^{0.13|\Sigma|} < 1$, and thus

$$\Pr[h \text{ confirms a truncated obstruction}] \leqslant f^{|\Sigma|/2} .$$

Complete Obstructions For complete obstructions where $w \leqslant w_{\max}$ we need to be more careful, as the probability of an obstruction being confirmed depends on its size $w = |W|$. We will consider each value of w in turn, and let $P(w) = \Pr[h \text{ confirms a complete obstruction of size } w]$. Summing $P(w)$ over all even $w \in \{4, 6, \ldots, w_{\max}\}$ we get a bound on $\Pr[h \text{ confirms a complete obstruction}]$.

Instead of the set W we will let the first component of the obstruction be a vector $\vec{W} = (x_1, \ldots, x_w) \in S^w$. Before specifying $\vec{W}$, however, we will define M_d, M_{d-1}, L_{d-1} as matching on the index set $\{1, \ldots, w\}$.

We specify the obstruction in the following order:

1. First we choose which indices correspond to M_d and M_{d-1}.
2. Next we specify which edges of M_{d-1} are contained in L_{d-1}.
3. Then we describe which keys of S go into the locations of $\vec{W}$ corresponding to $Z_{d-1} = \bigcup L_{d-1}$.

[2] We use the notation, $k!! = k \cdot (k-2) \cdot (k-4) \cdot \ldots \cdot 1$.

4. Finally, we choose which keys go into the remaining positions of $\vec{W}$.

In this way each obstruction will be accounted for $w!$ times. Note that for $Z_{d-1} = \bigcup L_{d-1}$ to be a $(d-2)$-zero set, then Z_{d-1} must also be a zero set – the keys themselves, before computing any derived characters. Thus

$$P(w) \leqslant \sum_{\substack{M_d, M_{d-1}, L_{d-1}, \\ \vec{W}=(x_1,\ldots,x_w)\in S^w, \\ \Delta_{i\in Z_{d-1}} x_i = \varnothing}} \frac{\Pr\Big[h \text{ confirms } \mathcal{O} = (\vec{W}, M_d, M_{d-1}, L_{d-1})\Big]}{w!}.$$

In the following, we bound the number of ways to perform each of the four steps outlined above.

1. As discussed in the previous section, M_d, M_{d-1} can be chosen among the w indices in $((w-1)!!)^2 \cdot w/2$ ways.

2. Let $\{i, j\}$ be the two indices of $\{1, \ldots, w\}$ not covered by M_d. At least one of these indices must be covered by L_{d-1}, as discussed in Section 6.2. We distniguish between two cases: If $\{i, j\} \in M_{d-1}$, this edge must be included in L_{d-1}, giving at most $2^{|M_{d-1}|-1} = 2^{w/2-1}$ valid submatchings of M_{d-1}.

If $\{i, j\} \notin M_{d-1}$, then there exists edges $\{i, i'\}$ and $\{j, j'\}$ in M_{d-1}. We can include one, the other, or both of these in L_{d-1}, along with any subset of the remaining $|M_{d-1}| - 2 = w/2 - 2$ edges, giving $3 \cdot 2^{w/2-2}$ options.

The choice made in step 1 decides which case applies, and thus there are at most $3 \cdot 2^{w/2-2}$ ways of performing step 2.

3. Let $z = |Z_{d-1}| = 2\,|L_{d-1}|$. As these z entries of $\vec{W}$ must form a zero set, Corollary 39 states that the keys for these positions can be chosen in at most $3^c \cdot n^{z-2}$ ways.

4. The remaining $w - z$ entries of $\vec{W}$ can be chosen in at most n^{w-z} ways.

Multiplying the number of choices for each of the four steps, the total number of complete obstructions of size w is bounded by

$$3 \cdot 2^{w/2-2} \cdot ((w-1)!!)^2 \cdot w/2 \cdot n^{w-2} \cdot 3^c$$

and hence

$$\begin{aligned}
P(w) &\leqslant 3 \cdot 2^{w/2-2} \cdot ((w-1)!!)^2 \cdot w/2 \cdot n^{w-2} \cdot 3^c \cdot \frac{p^{w-1}|\Sigma|^{2-w}(3/|\Sigma|)^{d-2}2^{w/4+1}}{w!} \\
&= \frac{3}{2} \cdot 2^{3w/4-1} \cdot \frac{((w-1)!!)^2}{(w-1)!} \cdot \left(\frac{pn}{|\Sigma|}\right)^{w-1} \cdot \frac{|\Sigma|}{n} \cdot 3^c \cdot \left(\frac{3}{|\Sigma|}\right)^{d-2} \\
&= 9 \cdot 2^{3w/4-2} \cdot \frac{(w-1)!!}{(w-2)!!} \cdot f^{w-1} \cdot \frac{3^c}{n} \cdot \left(\frac{3}{|\Sigma|}\right)^{d-3} \\
&= \frac{2^{3w/4-2}}{3} \cdot \frac{(w-1)!!}{(w-2)!!} \cdot f^{w-4} \cdot \mu^3 \cdot \frac{3^c}{n} \cdot \left(\frac{3}{|\Sigma|}\right)^{d}.
\end{aligned}$$

What is left is to bound $\sum_{\text{even } w=4}^{w_{\max}} P(w)$. Let $g(w) = \frac{2^{3w/4-2}}{3} \cdot \frac{(w-1)!!}{(w-2)!!} \cdot f^{w-4}$.

Observation 34.

$$\sum_{\text{even } w=4}^{w_{\max}} g(w) \leqslant 3$$

Proof. First, observe that $g(w+2) = \frac{w+1}{w} \cdot 2^{3/2} f^2 \cdot g(w) \leqslant \frac{w+1}{w} \cdot 0.71 \cdot g(w)$, when $f \leqslant 1/2$. For any fixed k we thus have

$$\sum_{\text{even } w=k}^{w_{\max}} g(w) \leqslant g(k) \cdot \sum_{i=0}^{\infty} \left(\frac{k+1}{k} \cdot 0.71 \right)^i .$$

For $k = 4$ we thus have

$$\sum_{\text{even } w=4}^{w_{\max}} g(w) \leqslant g(4) \cdot 9 = 9 \,,$$

as $g(4) = \frac{2}{3} \cdot \frac{3}{2} = 1$. □

Hence $\sum_{\text{even } w=4}^{w_{\max}} P(w) \leqslant 9\mu^3 (3/|\Sigma|)^d \cdot 3^c/n$, and the probability that h confirms *any* obstruction is bounded by $9\mu^3 (3/|\Sigma|)^d \cdot 3^c/n + f^{|\Sigma|/2}$. Thus we get the claim in Theorem 17.

7 Proof of Theorem 1 and Theorem 2

As mentioned earlier, Theorem 2 will follow from Theorem 1 and we prove this in Section 7.3. In order to prove our main Theorem 1, we start with the following quite technical result. In this result, $\mathcal{J}_{\max}$ denotes the event that there are no large layers of Lemma 16, namely the event that $\mathrm{E}\big[S_i \bigm| \overline{h}\big] = 0$ for all $i \geqslant i_{\max}$.

Theorem 35. *If $|\Sigma| \geqslant 2^{11}$ and $\mu \in [|\Sigma|/4, |\Sigma|/2]$, then the following holds for any $p > 0$*

$$\Pr\Big[X < \mu - \sqrt{\ln(1/p)\mu} \cdot \gamma_1 - \gamma_2\Big] < 3p + \mathcal{P}_{error} \,,$$

where $\mathcal{P}_{error} = i_{\max} \cdot 2\Pr[\neg\mathcal{J}] + \Pr[\neg\mathcal{J}_{\max}]$.

The definitions of γ_1 and γ_2 are quite involved and relies on a number of symbols that will be defined and motivated in Section 5.1. For $|\Sigma| \geqslant 2^{11}$, γ_1 can be considered to be approximately 2 while γ_2 is of order $\widetilde{O}(\ln(1/p) \cdot \ln\ln(\mu))$. The full definition of γ_1 and γ_2 can be found in table 1 on page 44.

7.1 Proof of Theorem 35

The proof essentially combines the analyses done for the different layers in Section 5.

Proof. The proof will proceed by applying a union bound over the contribution of all layers. We begin by noting that we can assume that $\ln(1/p) \leqslant \mu/8.6$. Namely, we claim that $\ln(1/p) > \mu/8.6$ implies that $\gamma_2 > \mu$, which, in turn, makes the event in Theorem 35 trivially false. To see this, we note that $\gamma_2 \geqslant 8.6 \ln(1/p)$ always, since the following hold regardless of p:

- $\Delta_{i_{nr}} \geqslant 1.33 \cdot e \cdot \ln(1/p) \approx 3.61 \cdot \ln(1/p)$ because $s_{sec}/p_{reg} \geqslant 1/p$ and $p_{reg} \leqslant p$

Symbol	Definition	Description
s_{sec}	20	Scales error probability of secondary events
s_{all}	160	Scales error probability in each layer
p_{reg}	$\frac{p}{\log_6\left(\frac{\mu/2}{\ln(s_{sec}/p)}\right)} \cdot \frac{1}{s_{all}}$	Threshold for error probability in regular layers
i_{nr}	$\min\{i : \bar{\mu}_i < \ln(s_{all}/p_{reg})\}$	First non-regular layer
$i_{\max}$	$\ln(\lvert\Sigma\rvert^{d-2}2^s)$	Maximum number of layers with non-zero expected size (whp), where s is the number of selection bits. Defined in Lemma 16.
n_{top}	$\log_6(\ln(s_{sec}/p_{reg}) \cdot s_{all}/p)$	Bound on the number of layers handled by Lemma 27
p_{top}	$\min\left\{\frac{p_{reg}}{s_{sec}}, \frac{p}{n_{top} \cdot s_{all}}\right\}$	Error probability for each layer in Lemma 27
ε_3	$(2+\sqrt{6})\sqrt{\ln(s_{sec}/p)/\lvert\Sigma\rvert} + (\frac{1}{2}\ln(1/p)+6)/\mu$	Stretch factor of multiplicative deviation of Theorem 12
Δ_{reg}	$0.181\sqrt{\ln(1/p)\mu} + 0.066\sqrt{\frac{\mu}{\ln(1/p)}} + \sqrt{2}$	Total deviation of regular layers, from 4 and up (Theorem 14)
$\Delta_{i_{nr}}$	$1.33e\ln(s_{sec}/p_{reg}) + 1$	Deviation of layer i_{nr} (Lemma 25)
Δ_{top}	$2\ln(1/p_{top}) \cdot (2 + \ln\ln(n_{top})) + n_{top}$	Deviation of layers $i_{nr}+1$ through i_∞ (Lemma 27)
Δ_{nonreg}	$\Delta_{i_{nr}} + \Delta_{top} + 3$	Total deviation of non-regular layers, i_{nr} and above (Theorem 15)
γ_1	$\sqrt{7/3 \cdot (1+\varepsilon_3)} + 0.181$	Multiplicative term of deviation in Theorem 35
γ_2	$\Delta_{reg} - 0.181\sqrt{\ln(1/p)\mu} + \Delta_{nonreg} + \ln(1/p)$	Additive term of deviation in Theorem 35

Table 1: Overview of the symbols used in the statement of Theorem 35. See discussion in Section 5.1.

- $\Delta_{top} \geqslant 4 \cdot \ln(1/p)$ since $p_{top} \leqslant p$
- $\Delta_{reg} \geqslant 0.181\sqrt{\ln(1/p)\mu}$ by definition .

Assuming that $\ln(1/p) \leqslant \mu/8.6$, together with the assumptions in the theorem statement, gives us that $\bar{\mu}_2 \geqslant \ln(s_{sec}/p)$. This means that layers 1 and 2 are certainly regular, i.e., $i_{nr} \geqslant 3$ (recall that i_{nr} was defined as the smallest index for which the corresponding layer is *not* regular). We now distinguish between whether layer 3 is also regular or not.

If layer 3 is regular, we use Theorem 12 to bound the contribution of the first three layers. This, together with the contribution of the remaining regular layers from Theorem 14, gives us:

$$\Pr\left[\sum_{i=1}^{i_{nr}-1} S_i < \sum_{i=1}^{i_{nr}-1} \mathrm{E}\big[S_i \mid \overline{h}\big] - \sqrt{\ln(1/p)\mu} \cdot \gamma_1 - \gamma_2 + \Delta_{nonreg}\right] < 2.96p + i_{nr} \cdot 2\Pr[\neg\mathcal{J}] \ .$$

The contribution of the non-regular layers is given by Theorem 15. If $i_{nr} = 3$, i.e., layer 3 is not regular, then note that no other higher index layer can be regular either. We then use Theorem 13 to bound the contribution of the first two layers and note that the bound is stronger than if we had used Theorem 12 (the one for the first three layers combined). We then proceed to consider the non-regular cases in a similar way as before.

Finally, by Lemma 16, with probability $\Pr[\neg\mathcal{J}_{\max}]$, we can assume that the contribution from higher layers is zero since:

$$\sum_{i=i_{\max}+1}^{\infty} \mathrm{E}\big[S_i \mid \overline{h}\big] = 0 \ ,$$

and hence

$$\sum_{i=1}^{i_{\max}} \mathrm{E}\big[S_i \mid \overline{h}\big] = \mu \ .$$

At this point, we get that:

$$\Pr\Big[X < \mu - \sqrt{\ln(1/p)\mu} \cdot \gamma_1 - \gamma_2\Big] < 3p + i_{\max} \cdot 2\Pr[\neg\mathcal{J}] + \Pr[\neg\mathcal{J}_{\max}] \ ,$$

which matches the claim. □

We now bound the $\mathcal{P}_{error}$ terms by further assuming that c and d are not too large. Note that similar bounds can be obtained even for bigger c and d.

Lemma 36. *For $|\Sigma| \geqslant 2^{11}$ and $c \leqslant \ln|\Sigma|$, the following holds:*

$$\mathcal{P}_{error} \leqslant (c + d - 2)\ln(|\Sigma|) \cdot \left(49\left(\frac{3}{|\Sigma|}\right)^{d-3} + 3\left(\frac{1}{2}\right)^{|\Sigma|/2}\right) \ .$$

Proof. Recall that $i_{\max} = \ln(|\Sigma|^{d-2} \cdot 2^s)$. We now derive expressions for $i_{\max}$, $\Pr[\neg\mathcal{J}]$ and $\Pr[\neg\mathcal{J}_{\max}]$. Since $c \leqslant |\Sigma|/(2\log|\Sigma|)$ by assumption and $2^s \leqslant |\Sigma|^c$ (the universe), we have that

$$i_{\max} \leqslant \ln(|\Sigma|^{d-2} \cdot |\Sigma|^c) \leqslant (c + d - 2) \cdot \ln|\Sigma| \ .$$

From Theorem 8, we also have that:

$$\Pr[\neg\mathcal{J}] \leqslant 24(3/|\Sigma|)^{d-3} + 1/2^{|\Sigma|/2}$$

Finally, we have that

$$\Pr[\neg\mathcal{J}_{\max}] \leqslant \left(\frac{1}{|\Sigma|}\right)^{d-3} + 3^{c+1}\left(\frac{3}{|\Sigma|}\right)^{d-1} + \left(\frac{1}{|\Sigma|}\right)^{|\Sigma|/2-1}$$

Thus

$$2i_{\max}\Pr[\neg\mathcal{J}] \leqslant 2(c+d-2)\ln(|\Sigma|)\cdot\left(24\left(\frac{3}{|\Sigma|}\right)^{d-3} + \left(\frac{1}{2}\right)^{|\Sigma|/2}\right).$$

Note that $(1/|\Sigma|)^{d-3} + 3^{c+1}/|\Sigma|(3/|\Sigma|)^{d-1} \leqslant (3/|\Sigma|)^{d-3}$ and $(1/|\Sigma|)^{|\Sigma|/2-1} < 1/2^{|\Sigma|/2}$. We have

$$\begin{aligned}\mathcal{P}_{error} &= i_{\max}\cdot 2\Pr[\neg\mathcal{J}] + \Pr[\neg\mathcal{J}_{\max}] \\ &\leqslant (c+d-2)\ln(|\Sigma|)\cdot\left(49\left(\frac{3}{|\Sigma|}\right)^{d-3} + 3\left(\frac{1}{2}\right)^{|\Sigma|/2}\right).\end{aligned}$$

□

7.2 Proof of Theorem 1

With the technical Theorem 35 in hand, we can now prove our main Theorem 1. The proof is technical but the goal is clear: To unwind the unwieldy expression expressions of Theorem 35. We restate the theorem below

Theorem 1. *For any $b \geqslant 1$ and $c \leqslant \ln s$, if $s \geqslant 2^{16}\cdot b^2$, and $\mu \in [s/4, s/2]$. For any $\delta > 0$,*

$$\Pr[|X| < (1-\delta)\mu] < 3\exp\left(\frac{-\delta^2\mu}{7}\right) + (c+b+1)\ln(s)\cdot\left(49\left(\frac{3}{s}\right)^{b} + 3\left(\frac{1}{2}\right)^{s/2}\right). \tag{8}$$

Proof. Let $b = d-3$. We will show that $\gamma_1 + \frac{\gamma_2}{\sqrt{\ln(1/p)\mu}} \leqslant \sqrt{7}$ whenever p, μ, and $|\Sigma|$ obey the stated restrictions, such that the theorem follows from Theorem 35 and Lemma 36. Before tackling γ_1

and γ_2, however, we will bound the involved symbols in terms of parameters $|\Sigma|, \mu$ and p:

$$\begin{aligned}\varepsilon_3 &= (2+\sqrt{6})\sqrt{\frac{\ln(s_{sec}/p)}{|\Sigma|}} + \frac{\frac{1}{2}\ln(1/p)+6}{\mu} \\ &\leqslant (2+\sqrt{6})\sqrt{\frac{\ln(1/p)+\ln(20)}{|\Sigma|}} + \frac{2\ln(1/p)+24}{|\Sigma|}\,,\end{aligned}$$

$$p_{reg} \geqslant \frac{p}{160\cdot\log_6(\mu)}\,,$$

$$\ln(s_{sec}/p_{reg}) \leqslant \ln(1/p) + \ln\ln(\mu) + 8.1$$

$$\begin{aligned}n_{top} &\leqslant \log_6(1/p) + \log_6(\ln(s_{sec}/p_{reg})) + \log_6(160) \\ &\leqslant \ln(1/p) + \ln\ln\ln(\mu) + 4\,,\end{aligned}$$

$$\begin{aligned}\ln(1/p_{top}) &\leqslant \ln(1/p) + \ln(160) + \max\{\ln\ln(\mu), \ln(n_{top})\} \\ &\leqslant \ln(1/p) + \ln\ln(1/p) + \ln\ln(\mu) + 6.5\,.\end{aligned}$$

Keeping μ fixed, we see that our bound on $\gamma_1 + \frac{\gamma_2}{\sqrt{\mu\ln(1/p)}}$ is maximized either when $\ln(1/p)$ goes towards zero (where the constant terms and dependencies on $\ln(\mu)$ dominate) or when $\ln(1/p)$ goes towards infinity. Further, it is clear that both γ_1 and $\gamma_2/\sqrt{\mu}$ decreases for larger μ as all terms of γ_2 are of order $O\left(\ln(1/p)\cdot(\ln\ln\ln(1/p) + \ln\ln(\mu))\right)$, with the higher-order terms found in Δ_{top}. We will thus evaluate the expression at the extremal points given by the restrictions of the theorem. As the statement is trivially true when $p > 1/3$, this is $\ln(1/p) = \ln(3) \approx 1.09$ and $p = 1/|\Sigma|^b$.

Next, we will argue that setting $b > 1$ will only lead to a stronger bound on $\gamma_2/\sqrt{\mu\ln(1/p)}$ when $p = 1/|\Sigma|^b$, due to the stronger requirement on $|\Sigma|$ that follows. To see this, let $\phi \geqslant 1$ such that $|\Sigma| = \phi\cdot 2^{16}\cdot b^2$. Then, when $p = 1/|\Sigma|^b$, the following "atomics" of γ_1 and $\gamma_2/\sqrt{\ln(1/p)\mu}$ are all maximized at $b = 1$.

$$\frac{\ln(1/p)}{\sqrt{\ln(1/p)\mu}} = \sqrt{\frac{\ln(1/p)}{\mu}} \leqslant \sqrt{\frac{b\cdot(2\ln(b)+\ln(\phi\cdot 2^{16}))}{\phi\cdot 2^{14}\cdot b^2}} \leqslant 0.0261$$

$$\frac{\ln(1/p)\cdot\ln\ln(\mu)}{\sqrt{\ln(1/p)\mu}} \leqslant \ln\ln(\phi\cdot 2^{14}\cdot b^2)\cdot\sqrt{\frac{b\cdot(2\ln(b)+\ln(\phi\cdot 2^{16}))}{\phi\cdot 2^{14}\cdot b^2}} \leqslant 0.0592$$

$$\frac{\ln(1/p)\cdot\ln\ln\ln(1/p)}{\sqrt{\ln(1/p)\mu}} \leqslant \ln\ln(b\ln(\phi\cdot 2^{16}\cdot b^2))\cdot\sqrt{\frac{b\cdot(2\ln(b)+\ln(\phi\cdot 2^{16}))}{\phi\cdot 2^{14}\cdot b^2}} \leqslant 0.0229.$$

All that's left now is to evaluate $\gamma_1+\gamma_2/\sqrt{\mu\ln(1/p)}$ using the bounds on the underlying symbols given above, setting $p = 1/|\Sigma|$ and $p = 1/3$.

At $p = 1/3$, $\gamma_1 + \gamma_2/\sqrt{\mu\ln(3)} \leqslant 2.58 \leqslant \sqrt{7}$.

At $p = 1/|\Sigma|$, $\gamma_1 + \gamma_2/\sqrt{\mu\ln(|\Sigma|)} \leqslant 2.34$. □

7.3 Subsampling and Proof of Theorem 2

In this section, we show a method for extending the bound of Theorem 1 to smaller μ while also allowing us to derive stronger concentration bounds when $\mu \ll |\Sigma|$. For these results we require X,

the set of selected keys, to be defined as the preimage of a collection of hash values that are all contained within a dyadic interval I. Then Theorem 1 bounds the number of keys hashed into I while Theorem 8 shows that all of these keys are independently and uniformly distributed within I with high probability. This allows us to apply a standard Chernoff bound to bound how many of the keys hashed into I are also in X.

For a fixed keyset S with $|S| = n$ and a subset of hash values $I \subset [2^l]$ define $X_I = \{x \in S \mid h(x) \in I\}$ to be the random variable that defines the preimage of I and $\mu_I = \mathrm{E}[|X_I|] = |I|/2^l \cdot n$.

Theorem 37. *If I is contained within a dyadic interval I' such that $\mu_{I'} \in [|\Sigma|/4, |\Sigma|/2]$ and $|\Sigma| \geqslant 2^{16} \cdot b^2$, then for any $p > 1/|\Sigma|^b$ it holds that*

$$\Pr\Big[|X_I| < \mu_I - (\sqrt{2} + \sqrt{\varepsilon})\sqrt{\ln(1/p)\mu_I} \wedge \mathcal{J}'\Big] < 4p + \mathcal{P}_{error}$$

where $\varepsilon = 7 \cdot (\mu_I/\mu_{I'})$ and $\mathcal{J}' = \mathcal{I}(\widetilde{h}_{<c+d}(X_{I'}))$ is the event that the derived keys $\widetilde{h}(X_{I'})_{c+d-1}$ are linearly independent.

Proof. Let $\mu' = \mathrm{E}[|X_I| \mid |X_{I'}|] = |X_{I'}| \cdot \mu_I/\mu_{I'}$. First, observe that

$$\big(\mu' \geqslant \mu_I - t_1\big) \wedge \left(|X_I| \geqslant \mu' \cdot \left(1 - \frac{t_2}{\mu_I - t_1}\right)\right) \implies |X_I| \geqslant \mu_I - t_1 - t_2\,,$$

hence

$$\begin{aligned}\Pr\big[|X_I| < \mu_I - t_1 - t_2 \wedge \mathcal{J}'\big] &\leqslant \Pr\big[\big(\mu' < \mu_I - t_1 \vee \big(|X| < \mu' - t_2 \wedge \mu' \geqslant \mu - t_1\big)\big) \wedge \mathcal{J}'\big] \\ &\leqslant \Pr\big[\mu' < \mu_I - t_1 \wedge \mathcal{J}'\big] + \Pr\big[|X| < \mu' - t_2 \wedge \mu' \geqslant \mu - t_1 \wedge \mathcal{J}'\big]\,.\end{aligned}$$

By Theorem 1,

$$\Pr\left[|X_{I'}| < \mu_{I'} \cdot \left(1 - \sqrt{\frac{7\ln(1/p)}{\mu_{I'}}}\right) \wedge \mathcal{J}'\right] < 3p + \mathcal{P}_{error}$$

and thus

$$\Pr\Big[\mu' < \mu_I - \sqrt{7\ln(1/p)\mu_I \cdot (\mu_I/\mu_{I'})} \wedge \mathcal{J}'\Big] < 3p + \mathcal{P}_{error}\,.$$

As $\mathcal{J}'$ implies that the elements of $X_{I'}$ are uniformly and independently distributed within I', $|X_I|$ follows a binomial distribution with mean μ'. Letting $t_1 = \sqrt{7\ln(1/p)\mu_I \cdot (\mu_I/\mu_{I'})}$ we thus have for any $\delta > 0$ that

$$\Pr\big[|X_I| < (1-\delta)\mu' \wedge \mu' \geqslant \mu_I - t_1 \wedge \mathcal{J}'\big] < \exp\left(-\frac{\delta^2(\mu_I - t_1)}{2}\right)$$

and hence

$$\Pr\left[|X_I| < \left(1 - \frac{\sqrt{2\ln(1/p)\mu_I}}{\mu_I - t_1}\right)\mu' \wedge \mu' \geqslant \mu_I - t_1 \wedge \mathcal{J}'\right] < \exp\left(-\frac{\ln(1/p)\mu_I(\mu_I - t_1)}{(\mu_I - t_1)^2}\right) \leqslant p\,.$$

Let $t_2 = \sqrt{2\ln(1/p)\mu_I}$. Then we have shown that

$$\Pr\big[|X_I| < \mu_I - t_1 - t_2 \wedge \mathcal{J}'\big] < 4p + \mathcal{P}_{error}\,.$$

Finally, the theorem follows by observing that

$$\begin{aligned} t_1 + t_2 &= \sqrt{\mu_I} \cdot \left(\sqrt{2\ln(1/p)} + \sqrt{7\ln(1/p)(\mu_I/\mu_{I'})} \right) \\ &= \sqrt{\mu_I \ln(1/p)} \cdot \left(\sqrt{2} + \sqrt{\varepsilon} \right) . \end{aligned}$$

□

Theorem 2 follows as a direct consequence of Theorem 37

Theorem 2. *Let $h : [u] \to [2^l]$ be a Tornado Tabulation hash function with $s \geqslant 2^{16}b^2$ and $c \leqslant \ln(s)$, A a set of keys, and $X = \{x \in A \mid h(x) < p\}$ for some $p \in [2^l]$. Suppose that $\mu = \mathrm{E}[X] \leqslant s/278$. Then for any $\delta < 1$, it holds that*

$$\Pr[||X| - \mu| > (1+\delta)\mu] < 5\exp\left(\frac{-\delta^2\mu}{3}\right) + (c+b+2)\ln(s) \cdot \left(49\left(\frac{3}{s}\right)^b + 3\left(\frac{1}{2}\right)^{s/2}\right).$$

Proof. Let $s = |\Sigma|$ and $b = d - 3$. Denote the value ε defined in Theorem 37 by ε_s.

Define $\hat{X}$ to be the number of keys $x \in S$ where $h(x) \leqslant \hat{t}$ for some $\hat{t} \in [2^l]$. Now, let $\hat{t}$ be the maximal power of 2 such that $\hat{\mu} = \mathrm{E}\left[\hat{X}\right] \leqslant |\Sigma|/2$. Note that $\hat{\mu} \geqslant |\Sigma|/4$.

As $\mu \leqslant |\Sigma|/4$, it follows that $\hat{t} \geqslant t$ and thus $[0, \hat{t}]$ is a dyadic interval containing $[0, t]$. Further, we see that $7 \cdot \mu/\hat{\mu} \leqslant 7\mu \cdot 4/|\Sigma| \leqslant 28(\mu/s) \leqslant 28/278$.Theorem 37 then gives

$$\Pr\left[X < \mu - (\sqrt{2} + \sqrt{\varepsilon_s})\sqrt{\ln(1/p)\mu} \wedge \mathcal{J}'\right] < 4p + \mathcal{P}_{error}$$

or, by substituting $\delta = \left(\sqrt{2} + \sqrt{\varepsilon_s}\right)\sqrt{\ln(1/p)/\mu}$,

$$\Pr\left[X < (1-\delta)\mu \wedge \mathcal{J}'\right] < 4\exp\left(\frac{-\delta^2\mu_I}{\left(\sqrt{2} + \sqrt{\varepsilon_s}\right)^2}\right) + \mathcal{P}_{error}\,.$$

Note that $(\sqrt{2} + \sqrt{\varepsilon_s})^2 \leqslant 3$.

Meanwhile, Theorem 4 bounds the upper tail. Observe that $\mathcal{J}'$ implies $\mathcal{I}(\widetilde{h}(X))$, as $\mathcal{J}'$ requires independence on a larger keyset while only inspecting the first $c + d - 1$ characters of each.

$$\Pr\left[X > \mu + \sqrt{3\ln(1/p)\mu} \wedge \mathcal{J}'\right] \leqslant p\,.$$

By Theorem 8, we know that $\Pr[\neg\mathcal{J}'] \leqslant 24(3/s)^b + 1/2^{s/2}$ and, from Lemma 36 that $\mathcal{P}_{error} \leqslant (c+b+1)\ln(s) \cdot \left(49\left(\frac{3}{s}\right)^b + 3\left(\frac{1}{2}\right)^{s/2}\right)$ and the theorem follows by adding the two probabilities together. □

8 Counting Zero Sets

In this section we prove Corollary 39, which allows for an efficient way of bounding the number of ordered zero sets that can be constructed from a set of n keys, each c characters long. Trivially, if one is looking for a zero-set of size k the first $k-1$ keys can be chosen in at most n^{k-1} ways – and at most one choice of the final key will make them form a zero set. Corollary 39 improves this bound by a factor of $n/3^c$.

Theorem 38. *Let $S \subseteq \Sigma^c$ be a set of n keys and let p be a generalized key. Then the number of $2t$-tuples $(x_1, \ldots, x_{2t}) \in S^{2t}$ such that $\Delta_{i\in[2t]} x_i = p$ is at most $((2t-1)!!)^c n^t$.*

Corollary 39. *Let $S \subseteq \Sigma^c$ with $|S| = n$ and $k \geqslant 4$. Then at most $3^c \cdot n^{k-2}$ tuples from S^k are zero-sets.*

Proof. Consider any prefix $(c_1, \ldots, c_{k-4}) \in S^{k-4}$. By Theorem 38 at most $3^c \cdot n^2$ tuples $(t_1, t_2, t_3, t_4) \in S^4$ satisfy $\Delta_{i\in[k-4]} c_i = \Delta_{i\in[4]} t_i$, making the tuple a zero-set. Summing over all n^{k-4} prefixes, the result follows. □

Theorem 38 generalizes [DKRT15, Lemma 2] which only applies to the case $p = 0$, that is, for zero sets, but this entails that we cannot use it to prove a statement like our Corollary 39 which keeps the dependency on c, the number of characters, at 3^c regardless of the size of zero sets considered. If c and n are known, one could construct a stronger version of Corollary 39 by applying Theorem 38 with a value of t minimizing $((2t-1)!!)^c/n^t$.

We prove Theorem 38 through the following, more general, lemma. Theorem 38 follows by setting all $A_k = S$.

Lemma 40. *Let $A_1, A_2, \ldots, A_{2t} \subseteq \Sigma^c$ be sets of keys and $p \subseteq [c] \times \Sigma$ a generalized key. Then the number of $2t$-tuples $(x_1, \ldots, x_{2t}) \in A_1 \times A_2 \times \cdots \times A_{2t}$ such that*

$$\Delta_{k\in[2t]} x_k = p$$

is at most $((2t-1)!!)^c \prod_{k=1}^{2t} \sqrt{|A_k|}$.

Proof of Lemma 40. The proof proceeds by induction over c. For $c = 1$ we consider all ways of partitioning the $2t$ coordinates of the tuple $\mathbf{x} = (x_1, \ldots, x_{2t})$ into an ordered list of pairs $((x_{i_k}, x_{j_k}))_{k=1}^t$ with $i_k < j_k$. The pairs can be chosen in $(2t-1)!! = (2t-1) \cdot (2t-3) \cdots 1$ ways and can be ordered in $t!$ ways. All mentions of pairs being before/after each other will be with reference to the chosen ordering, not the natural ordering of the coordinates in the pair. For $k \in [t]$ let $\mathbf{x}_{<k} = \bigcup_{l<k} \{x_{i_l}, x_{j_l}\}$ be the characters appearing in the first $k-1$ pairs of coordinates. We partition the characters of p into an arbitrary set of pairs $p = \{\{\alpha, \alpha'\}, \{\beta, \beta'\}, \ldots\}$, and we will use the notation α' to denote the "neighbour" of α in this pairing, letting $\alpha'' = \alpha$.

We fix the relationship between characters (x_{i_k}, x_{j_k}) in each pair by defining the permutation $\pi_{\mathcal{A}}$ on Σ parameterized by a set of characters $\mathcal{A} \subseteq \Sigma$. For the k'th pair (x_{i_k}, x_{j_k}) we require that $x_{j_k} = \pi_{\mathbf{x}_{<k}}(x_{i_k})$, where

$$\pi_{\mathcal{A}}(\alpha) = \begin{cases} \alpha' & \text{if } \alpha \in p \backslash \mathcal{A} \\ \alpha & \text{otherwise.} \end{cases}$$

In this way each pair of coordinates will contain two copies of the same character, except at most $|p|/2$ pairs which contain two distinct characters from p, thus ensuring that these characters appear an odd number of times overall. A pair with two copies of a character $\alpha \in p$ can only occur when a prior pair has provided the odd copy of α. Although this may appear to be a crucial limitation of the process we will later show that any tuple $\mathbf{x}$ with $\Delta\mathbf{x} = p$ can be constructed from several choices of ordered pairings. Note that all tuples $\mathbf{x}$ generated by this procedure will have a subset of p as symmetric difference, with the symmetric difference being exactly p iff all $|p|/2$ "mixed" pairs $(\alpha, \alpha') \in p$ occur.

As $\pi_{\mathcal{A}}$ is a permutation on Σ (for any fixed set $\mathcal{A}$) the number of possible assignments $(x_{i_k}, x_{j_k}) \in A_{i_k} \times A_{j_k}$ with $x_{j_k} = \pi_{\mathcal{A}}(x_{i_k})$ is at most $\min\{|A_{i_k}|, |A_{j_k}|\} \leqslant \sqrt{|A_{i_k}|\,|A_{j_k}|}$. Counting all tuples in $A_1 \times \cdots \times A_{2t}$ adhering to these restrictions, and summing over all $t! \cdot (2t-1)!!$ ordered pairings, will yield at most $t! \cdot (2t-1)!! \prod_{k=1}^{2t} \sqrt{|A_k|}$ tuples, of which many will be duplicates counted from several ordered pairings.

We now prove that each tuple with symmetric difference p can be produced from at least $t!$ distinct ordered pairings, thus proving the existence of at most $(2t-1)!! \prod_{k=1}^{2t} \sqrt{|A_k|}$ distinct such tuples. Consider a tuple $\mathbf{x}$ with $\Delta\mathbf{x} = p$, and note that $\mathbf{x}$ can be produced from an ordered pairing iff:

(1) Each pair $\{\alpha, \alpha'\} \in p$ appears in a paired set of coordinates $\{x_{i_k}, x_{j_k}\}$.

(2) The remaining $t - |p|/2$ pairs each contain two identical characters.

(3) Each pair mentioned in (1) precedes all other pairs containing α or α'.

First, we count the number of ways that positions containing a character from p can be partitioned into pairs satisfying (1). Given any such pairing, and the assumption that $\Delta\mathbf{x} = p$, at least one way of pairing the remaining coordinates of $\mathbf{x}$ to satisfy (2) exists. Second, we find the number of ways that a pairing satisfying (1) and (2) can be ordered to satisfy (3).

For $\alpha \in p$ let $\#(\alpha)$ be the number of occurrences of α in $\mathbf{x}$. For each set of neighbours $\{\alpha, \alpha'\} \in p$ the pair of coordinates mentioned in (1) can be chosen in $\#(\alpha) \cdot \#(\alpha')$ ways. There is thus at least $\prod_{\{\alpha,\alpha'\}\in p} \#(\alpha) \cdot \#(\alpha')$ valid pairings satisfying (1) and (2).

Disregarding (3), the t pairs can be ordered in $t!$ ways. We will now compute the fraction of these permutations satisfying (3). For each pair of neighbours $\{\alpha, \alpha'\} \in p$ let $\#(\alpha\alpha')$ be the number of pairs containing α or α'. Consider the following procedure for generating all permutations: First, for each pair $\{\alpha, \alpha'\} \in p$, choose $\#(\alpha\alpha')$ *priorities* from $\{1, \ldots, t\}$ which will be distributed amongst pairs of coordinates containing α and/or α'. Next, for each pair $\{\alpha, \alpha'\} \in p$, the paired coordinates containing $\{\alpha, \alpha'\}$ is assigned one of the $\#(\alpha\alpha')$ priorities reserved for α/α'. Afterwards the remaining (α, α)- and (α', α')-pairs are assigned the remaining priorities. This procedure will generate an ordering on the pairs satisfying (3) exactly when, in the second step, each mixed pair is given the highest priority amongst the $\#(\alpha\alpha')$ choices. Thus one in $\prod_{\{\alpha,\alpha'\}\in p} \#(\alpha\alpha')$ permutations satisfies (3).

Finally, observe that $\#(\alpha\alpha') = (\#(\alpha) + \#(\alpha'))/2$. Combining the two counting arguments it is seen that $\mathbf{x}$ can be created from at least

$$t! \prod_{\{\alpha,\alpha'\}\in p} \frac{\#(\alpha) \cdot \#(\alpha')}{\#(\alpha\alpha')} \geqslant t!$$

ordered pairings, showing that the lemma holds for $c = 1$.

For $c > 1$ we assume the lemma to be true for shorter keys, and proceed in a manner similar to that for $c = 1$. We will, at first, look at the *final* position of all involved keys (that is, the position characters (c, α) for $\alpha \in \Sigma$), and will consider the set p^c of characters from p appearing in the final position, i.e. $p^c = \{\alpha \in \Sigma \mid (c, \alpha) \in p\}$, which we again consider to be partitioned into pairs $p^c = \{(\alpha, \alpha'), (\beta, \beta'), \ldots\}$. Generally, we will use a^c to refer to the character at the c'th position of the key a and $\tilde{a} = a \backslash a^c$ for the preceding $c-1$ characters. We use the same notation for tuples and sets of keys where the operation is applied to each key, $A^c = \cup_{a\in A} a^c$ and $\tilde{A} = \cup_{a\in A} \tilde{a}$. For a

character $\beta \in \Sigma$ and set of keys A let $A[\beta] = \{a \in A \mid a^c = \beta\}$ be the keys of A having β as their last character.

We consider all ways of partitioning the $2t$ positions into an ordered set of pairs, this time requiring that the characters in the final position of each key satisfy $x^c_{j_k} = \pi_{\mathbf{x}^c_{<k}}(x^c_{i_k})$ where, for any $\mathcal{A} \subseteq \Sigma$,

$$\pi_{\mathcal{A}}(\alpha) = \begin{cases} \alpha' & \text{if } \alpha \in p^c \backslash \mathcal{A} \\ \alpha & \text{otherwise.} \end{cases}$$

For a given sequence $(\alpha_1, \ldots, \alpha_t) \in \Sigma^t$ of t character consider the number of $2t$-tuples $\mathbf{x} \in A_1 \times \ldots \times A_{2t}$ where $(x^c_{i_k}, x^c_{j_k}) = (\alpha_{i_k}, \pi_{\mathbf{x}^c_{<k}}(\alpha_{i_k}))$ for all $k \in [t]$ and $\Delta\mathbf{x} = p$. If $(\alpha_1, \ldots, \alpha_t)$ gives $\Delta\mathbf{x}^c = p^c$ then counting these tuples is equivalent to counting in how many ways each pair $(\tilde{x}_{i_k}, \tilde{x}_{j_k})$ can be chosen from $\tilde{A}_{i_k}[\alpha_k] \times \tilde{A}_{j_k}[\pi_{\mathbf{x}^c_{<k}}(\alpha_k)]$ such that $\Delta\tilde{\mathbf{x}} = \tilde{p}$. By the induction hypothesis the number of such $2t$-tuples is bounded by

$$((2t-1)!!)^{c-1} \prod_{k=1}^{t} \sqrt{|A_{i_k}[\alpha_k]| \cdot \left|A_{j_k}[\pi_{\mathbf{x}^c_{<k}}(\alpha_k)]\right|}\,.$$

Summing over all tuples $(\alpha_1, \ldots, \alpha_t)$ thus gives an upper bound on the number of $2t$-tuples with symmetric difference p while also adhering to the restrictions imposed by π on the characters in the last position of each key, which in turn is determined by the ordered pairing of the $2t$ coordinates. For ease of notation we use the shorthand π_k for $\pi_{\mathbf{x}^c_{<k}}$ where $\mathbf{x}^c_{<k}$ is understood as the characters $\alpha_1, \ldots, \alpha_{k-1}$ along with their neighbours as determined by π. Thus π_k is dependent on $\alpha_1, \ldots, \alpha_{k-1}$, even if this is not apparent from the notation.

To sum over all choices of α's we repeatedly apply Cauchy-Schwarz on the innermost term of the sum. For any $k \in [t]$ and fixed $\alpha_1, \ldots, \alpha_{k-1}$ we have

$$\sum_{\alpha_k \in \Sigma} \sqrt{|A_{i_k}[\alpha_k]|\,|A_{j_k}[\pi_k[\alpha_k]]|} \leqslant \sqrt{\sum_{\beta \in \Sigma} |A_{i_k}[\beta]| \sum_{\beta \in \Sigma} |A_{j_k}[\pi_k(\beta)]|} = \sqrt{|A_{i_k}|\,|A_{j_k}|}\,.$$

To see that this is an application of Cauchy-Schwarz observe that

$$\sum_{\alpha \in \Sigma} \sqrt{|A_{i_k}[\alpha]|\,|A_{j_k}[\pi_k(\alpha)]|}$$

is the inner product of the two vectors $\left(\sqrt{|A_{i_k}[\alpha]|}\right)_{\alpha \in \Sigma}$ and $\left(\sqrt{|A_{j_k}[\pi_k(\alpha)]|}\right)_{\alpha \in \Sigma}$ while

$$\sqrt{\sum_{\alpha \in \Sigma} |A_{i_k}[\alpha]| \sum_{\alpha \in \Sigma} |A_{j_k}[\pi_k(\alpha)]|}$$

is the product of their norms.

Applying the above inequality t times the total number of tuples for each ordered pairing

becomes

$$
\begin{aligned}
&((2t-1)!!)^{c-1} \sum_{(\alpha_1,\ldots,\alpha_t)\in\Sigma^t} \prod_{k=1}^{t} \sqrt{|A_{i_k}[\alpha_k]|\,|A_{j_k}[\pi_k(\alpha_k)]|} \\
&\leqslant \sqrt{|A_{i_t}|\,|A_{j_t}|} \cdot ((2t-1)!!)^{c-1} \sum_{(\alpha_1,\ldots,\alpha_{t-1})\in\Sigma^{t-1}} \prod_{k=1}^{t-1} \sqrt{|A_{i_k}[\alpha_k]|} \cdot \sqrt{|A_{j_k}[\pi_k(\alpha_k)]|} \\
&\leqslant \prod_{l=t-1}^{t} \sqrt{|A_{i_l}|\,|A_{j_l}|} \cdot ((2t-1)!!)^{c-1} \sum_{(\alpha_1,\ldots,\alpha_{t-2})\in\Sigma^{t-2}} \prod_{k=1}^{t-2} \sqrt{|A_{i_k}[\alpha_k]|} \cdot \sqrt{|A_{j_k}[\pi_k(\alpha_k)]|} \\
&\leqslant \prod_{l=t-2}^{t} \sqrt{|A_{i_l}|\,|A_{j_l}|} \cdot ((2t-1)!!)^{c-1} \sum_{(\alpha_1,\ldots,\alpha_{t-3})\in\Sigma^{t-3}} \prod_{k=1}^{t-3} \sqrt{|A_{i_k}[\alpha_k]|} \cdot \sqrt{|A_{j_k}[\pi_k(\alpha_k)]|} \\
&\vdots \\
&\leqslant ((2t-1)!!)^{c-1} \prod_{l=1}^{t} \sqrt{|A_{i_k}|\,|A_{j_k}|} \\
&= ((2t-1)!!)^{c-1} \prod_{k=1}^{2t} \sqrt{|A_k|}\,.
\end{aligned}
$$

Summing over all $t!(2t-1)!!$ ways of partitioning the coordinates into an ordered list of pairs we thus find at most $t!((2t-1)!!)^c \prod_{k=1}^{2t} \sqrt{|A_k|}$ tuples. By the same counting argument as presented for single-character keys each $2t$-tuple $\mathbf{x}^c \in \Sigma^c$ with $\Delta\mathbf{x}^c = p^c$ and which complies with π is produced from at least $t!$ distinct ordered pairings of its coordinates. Thus each sequence of applications of the induction hypothesis is repeated at least $t!$ times. Hence at most $((2t-1)!!)^c \prod_{k=1}^{2t} \sqrt{|A_k|}$ distinct $2t$-tuples $\mathbf{x}$ satisfy $\Delta\mathbf{x} = p$, proving Lemma 40. □

References

[AKK+20] Anders Aamand, Jakob Bæk Tejs Knudsen, Mathias Bæk Tejs Knudsen, Peter Michael Reichstein Rasmussen, and Mikkel Thorup. Fast hashing with strong concentration bounds. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1265–1278, 2020.

[BBK+23] Ioana O. Bercea, Lorenzo Beretta, Jonas Klausen, Jakob Bæk Tejs Houen, and Mikkel Thorup. Locally uniform hashing. In *64th FOCS*, pages 1440–1470, 2023.

[BCFM00] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000. See also STOC'98.

[BJK+02] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *Proc. 6th International Workshop on Randomization and Approximation Techniques (RANDOM)*, pages 1–10, 2002.

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

[CK07] Edith Cohen and Haim Kaplan. Summarizing data using bottom-k sketches. In *Proc. 27th PODC*, pages 225–234, 2007.

[Dav81] H.A. David. *Order Statistics*. Wiley, New York, 2 edition, 1981.

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

[DKT17] Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Mikkel Thorup. Fast similarity sketching. In *58th FOCS*, pages 663–671, 2017.

[DR98] Devdatt Dubhashi and Desh Ranjan. Balls and bins: A study in negative dependence. *Random Structures & Algorithms*, 13(5):99–124, 1998.

[DT14] Søren Dahlgaard and Mikkel Thorup. Approximately minwise independence with twisted tabulation. In *Proc. 14th Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 134–145, 2014.

[FEFGM07] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédédric Meunier. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. In *In Analysis of Algorithms (AOFA)*, 2007.

[FM85] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985. Announced at FOCS'83.

[HH08] Abdolhossein Hoorfar and Mehdi Hassani. Inequalities on the lambert w function and hyperpower function. *J. Inequal. Pure and Appl. Math*, 9(2):5–9, 2008.

[HT22] Jakob Bæk Tejs Houen and Mikkel Thorup. Understanding the moments of tabulation hashing via chaoses. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPIcs*, pages 74:1–74:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. 30th ACM Symposium on Theory of Computing (STOC)*, pages 604–613, 1998.

[JS68] Kumar Jogdeo and Stephen M Samuels. Monotone convergence of binomial probabilities and a generalization of ramanujan's equation. *The Annals of Mathematical Statistics*, 39(4):1191–1195, 1968.

[Li15] Ping Li. 0-bit consistent weighted sampling. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 665–674, 08 2015.

[LOZ12] Ping Li, Art B. Owen, and Cun-Hui Zhang. One permutation hashing. In *Proc. 26thAdvances in Neural Information Processing Systems*, pages 3122–3130, 2012.

[LSMK11] Ping Li, Anshumali Shrivastava, Joshua L. Moore, and Arnd Christian König. Hashing algorithms for large-scale learning. In *Proc. 25thAdvances in Neural Information Processing Systems*, pages 2672–2680, 2011.

[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 *Proc. 24th ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 209–228, 2013.

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

[SSS95] Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-Hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics*, 8(2):223–250, 1995. See also SODA'93.

[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 *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 classes and applications of hash functions. *Journal of Computer and System Sciences*, 22(3):265–279, 1981. See also FOCS'79.

[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.

A An Generalized Chernoff Bounds

The following is the standard Chernoff bound, here shown to apply to variables that are not independent, but whose sum is dominated by that of independent indicator variables. This result is known from [DR98] but we include a proof for completeness.

Lemma 41. *Let $X_1, \ldots, X_n$ be 0/1-variables and $p_1, \ldots, p_n$ be reals such that $\mu = \sum_i p_i$ and, for all $I \subseteq [n]$, $\Pr[\prod_{i\in I} X_i = 1] \leqslant \prod_{i\in I} p_i$ (implying $\mathrm{E}[X_i] \leqslant p_i$) then*

$$\Pr\left[\sum_{i=1}^{n} X_i > (1+\delta)\mu\right] \leqslant \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu$$

for any $\delta > 0$.

Proof. Let $a = 1 + \delta$, $X = \sum_{i=1}^n X_i$ and $s > 0$. Let $Z_1, \ldots, Z_n$ be independent 0/1-variables with $\mathrm{E}[Z_i] = p_i$ and define $Z = \sum_{i=1}^n Z_i$. For any $I \subseteq [n]$ we then have

$$\mathrm{E}\left[\prod_{i\in I} X_i\right] \leqslant \prod_{i\in I} p_i \leqslant \mathrm{E}\left[\prod_{i\in I} Z_i\right].$$

Let i be a positive integer. For $V \in [n]^i$, which may contain duplicate entries, let I be the distinct elements of V. We similarly have

$$\mathrm{E}\left[\prod_{v\in V} X_v\right] = \mathrm{E}\left[\prod_{l\in I} X_l\right] \leqslant \mathrm{E}\left[\prod_{l\in I} Z_v\right] = \mathrm{E}\left[\prod_{v\in V} Z_v\right]$$

hence

$$\mathrm{E}\big[X^i\big] = \sum_{V\in[n]^i} \mathrm{E}\left[\prod_{v\in V} X_v\right] \leqslant \sum_{V\in[n]^i} \mathrm{E}\left[\prod_{v\in V} Z_v\right] = \mathrm{E}\big[Z^i\big].$$

$$\begin{aligned}\Pr[X \geqslant a\cdot\mu] &= \Pr\big[e^{sX} \geqslant e^{sa\cdot\mu}\big] \\ &\leqslant \frac{\mathrm{E}[\exp(sX)]}{e^{sa\cdot\mu}}.\end{aligned}$$

Note that

$$\mathrm{E}[\exp(sX)] = \sum_{i=0}^{\infty} \frac{s^i \mathrm{E}\big[X^i\big]}{i!} \leqslant \sum_{i=0}^{\infty} \frac{s^i \mathrm{E}\big[Z^i\big]}{i!} = \mathrm{E}[\exp(sZ)].$$

Due to the independence of $Z_1, \ldots, Z_n$ we further have

$$\begin{aligned}\mathrm{E}[\exp(sX)] &\leqslant \prod_{i=1}^{n} \mathrm{E}[\exp(sZ_i)] \\ &= \prod_{i=1}^{n} (p_i \cdot e^s + (1-p_i)) \\ &\leqslant \prod_{i=1}^{n} (\exp(p_i(e^s-1))) \\ &\leqslant \exp(\mu(e^s-1))\end{aligned}$$

and thus

$$\begin{aligned}\Pr[X \geqslant a \cdot \mu] &\leqslant \frac{\exp(\mu(e^s - 1))}{\exp(sa\mu)} \\ &= \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu\end{aligned}$$

when setting $s = \ln(a) = \ln(1 + \delta)$. □

Appendix C

Online Sorting and Online TSP

Online sorting and online TSP: randomized, stochastic, and high-dimensional

Mikkel Abrahamsen
University of Copenhagen, Denmark

Ioana O. Bercea
KTH Royal Institute of Technology, Stockholm, Sweden

Lorenzo Beretta
University of California, Santa Cruz, USA

Jonas Klausen
University of Copenhagen, Denmark

László Kozma
Institut für Informatik, Freie Universität Berlin, Germany

Abstract

In the *online sorting problem*, n items are revealed one by one and have to be placed (immediately and irrevocably) into empty cells of a size-n array. The goal is to minimize the sum of absolute differences between items in consecutive cells. This natural problem was recently introduced by Aamand, Abrahamsen, Beretta, and Kleist (SODA 2023) as a tool in their study of online geometric packing problems. They showed that when the items are reals from the interval $[0, 1]$ a competitive ratio of $O(\sqrt{n})$ is achievable, and no deterministic algorithm can improve this ratio asymptotically.

In this paper, we extend and generalize the study of online sorting in three directions:

- *randomized*: we settle the open question of Aamand et al. by showing that the $O(\sqrt{n})$ competitive ratio for the online sorting of reals cannot be improved even with the use of randomness;
- *stochastic*: we consider inputs consisting of n samples drawn uniformly at random from an interval, and give an algorithm with an improved competitive ratio of $\widetilde{O}(n^{1/4})$. The result reveals connections between online sorting and the design of efficient hash tables;
- *high-dimensional*: we show that $\widetilde{O}(\sqrt{n})$-competitive online sorting is possible even for items from $\mathbb{R}^d$, for arbitrary fixed d, in an adversarial model. This can be viewed as an online variant of the classical TSP problem where tasks (cities to visit) are revealed one by one and the salesperson assigns each task (immediately and irrevocably) to its timeslot. Along the way, we also show a tight $O(\log n)$-competitiveness result for *uniform metrics*, i.e., where items are of different *types* and the goal is to order them so as to minimize the *number of switches* between consecutive items of different types.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases sorting, online algorithm, TSP

Funding *Ioana O. Bercea, Lorenzo Beretta and Jonas Klausen:* Supported by grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.
Mikkel Abrahamsen: Supported by Starting Grant 1054-00032B from the Independent Research Fund Denmark under the Sapere Aude research career programme and part of Basic Algorithms Research Copenhagen (BARC), supported by the VILLUM Foundation grant 16582.
László Kozma: Supported by DFG grant KO 6140/1-2.

1 Introduction

The following natural problem called *online sorting* was recently introduced by Aamand, Abrahamsen, Beretta, and Kleist [1]: Given a sequence $x_1, \ldots, x_n$ of real values, assign them bijectively to array cells $A[1], \ldots, A[n]$. Crucially, after receiving x_j, for $j = 1, 2, \ldots, n$, we must immediately and irrevocably set $A[i] = x_j$, for some previously unused array index $i \in [n]$. The goal is to minimize $\sum_{i=1}^{n-1} |A[i+1] - A[i]|$, the sum of absolute differences between items in consecutive cells.

Aamand et al. [1] study the problem as modelling certain geometric online packing problems. In particular, they use it to show lower bounds on the competitive ratio of such problems[1]. It is easy to see that the optimal (offline) solution of online sorting is to place the entries in sorted (increasing) order, and the question is how to approximate this solution in an online setting where the items are revealed one by one.

The problem evokes familiar scenarios where we must commit *step-by-step* to an ordering of items, such as when scheduling meetings in a calendar, writing recipes in a notebook, planting trees in a garden, or writing data into memory cells. In such situations, some local coherence or sortedness is often desirable, but once the location of an item has been assigned, it is expensive to change it; for instance, it may be difficult to reschedule meetings or to migrate data items once memory locations are referenced from elsewhere. One must then carefully balance between placing similar items next to each other and leaving sufficiently large gaps amid uncertainty about future arrivals[2].

Moreover, online sorting can be firmly placed among familiar and well-studied online problems; we briefly mention two. As they differ from online sorting in crucial aspects, a direct transfer of techniques appears difficult.

- *list labeling* or *order maintenance* [16, 11, 6, 26, 7]: in this problem, a sequence of values are to be assigned labels consistent with their ordering (in effect placing the values into an array). The main difference from online sorting is that the sequence must be fully sorted and the goal is to minimize *recourse*, i.e., movement of previously placed items.
- *matching on the line* [15, 24, 14, 4]: here, a sequence of clients (e.g., drawn from $[n]$) are to be matched uniquely and irrevocably to servers (say, locations in $[n]$). The problem differs from online sorting mainly in its cost function; the goal here is to minimize the sum of distances between matched client-server pairs.

One of the main results of Aamand et al. [1] is an algorithm for online sorting with competitive ratio $O(\sqrt{n})$. Aamand et al. require the input entries to come from the unit interval $[0, 1]$ and to contain the endpoints 0 and 1. Conveniently, this makes the offline cost equal to 1 and the competitive ratio equal to the online cost. We show that the same competitiveness result can be obtained even if these assumptions are relaxed.

Aamand et al. [1] also show that the $O(\sqrt{n})$ bound cannot be improved by any deterministic algorithm, and leave it as an open question whether it can be improved through randomization (assuming that the adversary is oblivious, i.e., that it does not know the "coin

[1] The *competitive ratio* of an online algorithm is the worst-case ratio of its cost to the optimum (offline) cost over inputs of a certain size n. The competitive ratio of a problem is the best competitive ratio achievable by an online algorithm for the problem.

[2] The cost measure of online sorting is also natural as a measure of the *unsortedness* of a sequence. For this, the *number of inversions* (i.e., the number of pairs $a < b$ where b appears before a) is perhaps more widely used. We argue, however, that with this latter cost, one cannot obtain a nontrivial competitive ratio. Consider an adversary that outputs $n/2$ copies of 0.5, followed by either all 0s or all 1s until the end. It is easy to see that one of these choices results in at least $n^2/8$ inversions for any online algorithm, whereas the optimal (sorted) sequence has zero inversions.

flips" of our algorithm). Note that in several online problems such as *paging* or *k-server*, randomization can lead to asymptotic improvements in competitiveness (against an oblivious adversary); e.g., see [10]. As our first main result, we show that for online sorting, randomization (essentially) does not help.

▶ **Theorem 1.** *The (deterministic and randomized) competitive ratio of online sorting is* $\Theta(\sqrt{n})$*. The lower bound* $\Omega(\sqrt{n})$ *holds even when the input numbers are from* $[0, 1]$*.*

Online TSP. Ordering real values with the above cost (sum of differences between consecutive items) can be naturally viewed as a one-dimensional variant of the Traveling Salesperson Problem (TSP). Suppose that n cities are revealed one by one (with repetitions allowed), and a salesperson must decide, for each occurrence c of a city, on a timeslot for visiting c, i.e., the position of c in the eventual tour. The cost is then the length of the fully constructed tour[3]. Formally, given a sequence $(x_1, \ldots, x_n)$ of items $x_i \in S$, for a metric space S with distance function $d(\cdot, \cdot)$, the goal is to assign the items bijectively to array cells $A[1], \ldots, A[n]$ in an online fashion such as to minimize $\sum_{i=1}^{n-1} d\left(A[i], A[i+1]\right)$. We refer to this problem[4] as *online TSP in* S.

Online TSP in $\mathbb{R}$ is exactly online sorting. A natural d-dimensional generalization is *online TSP in* $\mathbb{R}^d$, with the Euclidean distance $d(\cdot, \cdot)$ between items.[5] A difficulty arising in dimensions two and above is that the optimal cost is no longer constant; in $\mathbb{R}^d$, the (offline) optimum may reach $\Theta(n^{1-\frac{1}{d}})$ even if the input points come from a unit box. Computing the optimum exactly is NP-hard even if $d = 2$ [22]. Our second main result is an online algorithm whose cost is $O(n^{1-\frac{1}{d+1}})$ and a competitiveness guarantee close to the one-dimensional case.

▶ **Theorem 2.** *There is a deterministic algorithm for online TSP in* $\mathbb{R}^d$ *with competitive ratio* $\sqrt{d} \cdot 2^d \cdot O(\sqrt{n \log n})$*.*

As this setting includes online sorting as a special case, the lower bound of $\Omega(\sqrt{n})$ applies. A key step in obtaining Theorem 2 is the study of the *uniform metric* variant of the problem, i.e., the case of distance function d with $d(x, y) = 1$ if and only if $x \neq y$. This captures the natural problem where items (or tasks) fall into a certain number of *types*, and we wish to order them such as to minimize the number of *switches*, i.e., consecutive pairs of items of different types. For this case we prove a tight (deterministic and randomized) competitive ratio, independent of the number of different types, which may be of independent interest. Our algorithm is a natural greedy strategy; we analyze it by modelling the evolution of contiguous runs of empty cells as a coin-removal game between the algorithm and adversary.

▶ **Theorem 3.** *The competitive ratio of online sorting of* n *items under the uniform metric is* $\Theta(\log n)$*. The upper bound* $O(\log n)$ *is achieved by a deterministic algorithm and the lower bound* $\Omega(\log n)$ *also holds for randomized algorithms.*

Aamand et al. also consider the setting where the array is only partially filled (placing n items into $m > n$ cells). In this case, the cost is understood as the sum of distances $d(x, y)$

[3] A small technicality is whether the salesperson must return to the starting point or not. The effect of this in our cost regime, however, is negligible.

[4] This model is sometimes referred to as the *online-list model*; it has been considered mostly in the context of scheduling problems [23, 13]. To our knowledge, TSP has not been studied in this setting before. Note however, that a different online model, called the *online-time model* has been used to study TSP [3, 20, 9]. In that model new cities can be revealed while the salesperson is already executing the tour; this may require changing the tour on the fly. Results in the two models are not comparable.

[5] One may also view this task as a form of *dimensionality reduction*: we seek to embed a d-dimensional data set in a one-dimensional space (the array), while preserving some distance information.

over pairs of items x, y with no other item placed between them. We thus extend our previous result to arrays of a larger size, obtaining a tight characterization.

▶ **Theorem 4.** *The competitive ratio (deterministic and randomized) of online sorting of n items with an array of size $\lceil \gamma n \rceil$, with $\gamma > 1$, under the uniform metric is $\Theta(1 + \log \frac{\gamma}{\gamma - 1})$.*

Stochastic input. Given the (rather large) $\Omega(\sqrt{n})$ lower bound on the competitive ratio of online sorting, it is natural to ask whether we can overcome this barrier by relaxing the worst-case assumption on the input. Such a viewpoint has become influential recently in an attempt to obtain more refined and more realistic guarantees in online settings (e.g., see [25, 14]). A natural model is to view each item as drawn independently from some distribution, e.g., uniformly at random from a fixed interval. Our next main result shows an improved competitive ratio for such stochastic inputs:

▶ **Theorem 5.** *There is an algorithm for online sorting of n items drawn independently and uniformly at random from $(0, 1]$ that achieves competitive ratio $O((n \log n)^{1/4})$ with probability at least $1 - 2/n$.*

The algorithm illuminates a connection between online sorting and hash-based dictionaries [19, § 6.4]. In the latter, the task is to place a sequence of n keys in an array of size $O(n)$, with the goal of minimizing *search time*. Fast searches are achieved by hashing keys to locations in the array. Due to hash collisions, not all elements can be stored exactly at their hashed location, and various paradigms have been employed to ensure that they are stored "nearby" (for the search to be fast). We adapt two such paradigms to online sorting. The first is to hash elements into buckets and solve the problem separately in each bucket. Each bucket has a fixed capacity, and so an additional *backyard* is used to store keys that do not fit in their hashed bucket [2, 8]. In Theorem 5, we use the values of the entries to assign them to buckets and employ a similar backyard design. We note some critical technical differences: in our design, all buckets must be full and we solve the problem recursively in each bucket; we also operate in a much tighter balls-into-bins regime, as we are hashing n elements into exactly n locations.

When the array size is allowed to be bigger than n, we employ yet another way for resolving hash collisions: the linear probing approach of Knuth [18]. Here, the value $\lceil \alpha n \rceil$ serves as the hash location of an entry $\alpha \in (0, 1)$. Intuitively, this is a good approximation for where the entry would appear in the sorted order. If we make sure that we place the entry close enough to this intended location, we can hope for a small overall cost. That is, we use the fact that linear probing places similar values close to each other. We get the following:

▶ **Theorem 6.** *For any $\gamma > 1$, there is an algorithm for online sorting of n items drawn independently and uniformly from $(0, 1)$ into an array of size $\lceil \gamma n \rceil$ that achieves expected competitive ratio $O\left(1 + \frac{1}{\gamma - 1}\right)$.*

Paper structure. In § 2 we present our results for the standard (one-dimensional) online sorting, in particular the lower bound for the randomized competitive ratio (Theorem 1). Results for online TSP in $\mathbb{R}^d$ and results for the uniform metric (Theorems 2, 3, 4) are in § 3. Our results for online sorting with stochastic input (Theorems 5, 6) are in § 4. We conclude with a list of open questions in § 5. We defer some proofs and remarks to the Appendix.

2 Competitiveness for online sorting

Given a sequence $X = (x_1, \ldots, x_n) \in \mathbb{R}^n$ and a bijection $f \colon [n] \to [n]$ assigning "array cells" $A[i] = x_{f(i)}$, let $\mathtt{D}_f(X) = \sum_{i=1}^{n-1} |A[i+1] - A[i]|$. Let $\mathtt{OPT}(X)$ denote the quantity

$\min_f \mathtt{D}_f(X)$, i.e., the *offline optimum*.

An *online* algorithm is one that constructs the mapping f incrementally. Upon receiving x_j, for $j = 1, \ldots, n$, the algorithm immediately and irrevocably assigns $f(i) = j$, for some previously unassigned $i \in [n]$. For an online algorithm $\mathcal{A}$, we denote by $\mathcal{A}(X)$ the cost $\mathtt{D}_f(X)$ for the function f constructed by algorithm $\mathcal{A}$ on input sequence X. We are interested in the competitive ratio $\mathtt{C}_\mathcal{A}(n)$ of an algorithm $\mathcal{A}$, i.e., the supremum of $\mathcal{A}(X)/\mathtt{OPT}(X)$ over all inputs X of a certain size n, and in the best possible competitive ratio $\mathtt{C} = \mathtt{C}(n) = \inf_\mathcal{A} \mathtt{C}_\mathcal{A}(n)$ obtainable by any online algorithm $\mathcal{A}$ for a given problem.

Upper bound. Aamand et al. [1] show that $\mathtt{C} \in \Theta(\sqrt{n})$, under the additional restrictions that $x_i \in [0, 1]$ for all $i \in [n]$, and that $\{0, 1\} \subseteq \{x_1, \ldots, x_n\}$. As our first result, we employ a careful doubling strategy to show the same upper bound without the two restrictions, for general sequences of n reals (the lower bound clearly continues to hold).

▷ Claim 7 (Proof in Appendix A). There is a deterministic online algorithm $\mathcal{A}$ for online sorting of an arbitrary sequence of n reals, with $\mathtt{C}_\mathcal{A} \in O(\sqrt{n})$.

Lower bound. The competitive ratio $\mathtt{C}_\mathcal{A}$ of a *randomized* algorithm $\mathcal{A}$ is the supremum of $\mathbb{E}[\mathcal{A}(X)]/\mathtt{OPT}(X)$ over all inputs X, where the expectation is over the random choices of $\mathcal{A}$. Aamand et al. [1] leave open the question of whether a lower bound of $\Omega(\sqrt{n})$ on the competitive ratio also holds for randomized algorithms. We settle this question in the affirmative. It is important to emphasize that the random choices of $\mathcal{A}$ are not known to the adversary, i.e., we assume the *oblivious* model [10].

▷ Claim 8. $\mathtt{C}_\mathcal{A} \in \Omega(\sqrt{n})$ for every (possibly randomized) online algorithm $\mathcal{A}$.

In the remainder of the section we prove Claim 8, which implies Theorem 1. As usual, to lower bound the performance of a randomized algorithm (a distribution over deterministic algorithms), we lower bound instead the performance of a deterministic algorithm on an (adversarially chosen) distribution over input sequences.

Input distribution. Assume for simplicity that $\sqrt{n}$ is an integer. Repeat the following until a sequence of length n is obtained. With probability $\frac{1}{2\sqrt{n}}$ set the remainder of the sequence to 0s. With probability $\frac{1}{2\sqrt{n}}$ set the remainder of the sequence to 1s. With probability $1 - \frac{1}{\sqrt{n}}$ offer $\frac{k}{\sqrt{n}}$ for $k = 0, \ldots, \sqrt{n} - 1$ as the next $\sqrt{n}$ elements of the sequence, and flip a new coin. We refer to the $\sqrt{n}$ elements produced by the third option as an *epoch*. Notice that $\mathtt{OPT} \leq 1$, thus it is enough to lower bound the expected cost of the algorithm.

Notation. Let T be a partially filled array, $|T|$ be the number of stored elements and $G(T)$ be the number of "gaps", i.e., maximally contiguous groups of empty cells.

Let $f(T)$ be the minimum cost of *filling* T by an online algorithm, when the input sequence is chosen according to the distribution described above, where the cost of two neighboring elements $T[i], T[i+1]$ are accounted for when $T[i]$ or $T[i+1]$ is filled, whichever happens the latest.

More formally, define the cost of a partially filled array T to be

$$c(T) = \sum_{\substack{i=0 \\ T[i],T[i+1] \text{ are nonempty}}}^{n-1} |T[i] - T[i+1]|,$$

and let $\mathcal{A}(T)$ be the final array produced by an online algorithm $\mathcal{A}$ when the remaining $n - |T|$ elements of the input sequence are generated as described above. Then $f(T) = c(\mathcal{A}(T)) - c(T)$

is the difference between the final cost and the cost of the partially filled array T. Thus $f(T) = 0$ if $|T| = n$, and $\mathbb{E}[f(\emptyset)]$ is the value we wish to bound, using $\emptyset$ for the empty array.

For mappings T_1, T_2 where $T_2[i] = T_1[i]$ for all indices where $T_1[i]$ is nonempty we say that T_2 can be obtained from T_1. For such a pair of mappings let $c(T_1, T_2) = c(T_2) - c(T_1)$ be the cost of transforming T_1 into T_2 according to the way of accounting given above.

Let $\mathcal{T}(T)$ be the set of mappings that can be obtained by inserting an epoch into T. That is, $\mathcal{T}(T)$ contains all mappings T' which can be obtained from T where the difference between T' and T corresponds to the elements of an epoch. Note that $|T'| = |T| + \sqrt{n}$ for $T' \in \mathcal{T}(T)$.

Let $\mathcal{L} = \{T \colon G(T) \leq \frac{\sqrt{n}}{8}\}$ and $\mathcal{H} = \{T \colon G(T) > \frac{\sqrt{n}}{8}\}$ be the sets of all partially filled arrays with a low/high number of gaps.

▶ **Lemma 9.** *Let $T_1, T_2 \in \mathcal{L}$ with $T_2 \in \mathcal{T}(T_1)$. Then $c(T_1, T_2) \geq \frac{3}{16}$.*

Proof. If an element is placed between two empty cells, the number of gaps will increase by one. If it is placed next to one or two occupied cells, the number of gaps stays constant or is reduced by one – call such an insertion an *attachment*. As $G(T_2) \leq G(T_1) + \frac{\sqrt{n}}{8}$ at least $\frac{7\sqrt{n}}{16}$ of the $\sqrt{n}$ insertions of the epoch must be attachments.

An attachment incurs cost at least $\frac{1}{\sqrt{n}}$ unless the value(s) in the neighboring cell(s) and the newly inserted value are identical. As all values within an epoch are distinct, an attachment can only be without cost if the neighboring cell was occupied in T_1 (that is, before the epoch). As $G(T_1) \leq \frac{\sqrt{n}}{8}$ at most $\frac{\sqrt{n}}{4}$ occupied cells border an empty cell. The remaining $\sqrt{n}(\frac{7}{16} - \frac{1}{4}) = \frac{3\sqrt{n}}{16}$ attachments will incur non-zero cost. ◀

As a shorthand, let $(T + 0)$ and $(T + 1)$ be the mappings obtained by filling all gaps in T with 0s/1s, respectively.

▶ **Lemma 10.** $c(T, (T + 0)) + c(T, (T + 1)) \geq G(T)$.

Proof. Each gap in T is bordered by at least one non-empty cell $A[i]$. We have $|A[i] - 0| + |A[i] - 1| = 1$. ◀

For a mapping T the expected remaining cost can be bounded from below by

$$\mathbb{E}[f(T)] \geq \frac{1}{2\sqrt{n}} \cdot c(T, (T + 0)) + \frac{1}{2\sqrt{n}} \cdot c(T, (T + 1)) + \left(1 - \frac{1}{\sqrt{n}}\right) \cdot \min_{T' \in \mathcal{T}(T)} \{c(T, T') + \mathbb{E}[f(T')]\}$$

and by Lemma 10 we thus have

$$\mathbb{E}[f(T)] \geq \frac{G(T)}{2\sqrt{n}} + \left(1 - \frac{1}{\sqrt{n}}\right) \cdot \min_{T' \in \mathcal{T}(T)} \{c(T, T') + \mathbb{E}[f(T')]\}\,. \tag{1}$$

$$\text{Let } \mathcal{L}(i) = \min_{\substack{T \in \mathcal{L} \\ |T| = n - i \cdot \sqrt{n}}} \mathbb{E}[f(T)], \quad \text{and} \quad \mathcal{H}(i) = \min_{\substack{T \in \mathcal{H} \\ |T| = n - i \cdot \sqrt{n}}} \mathbb{E}[f(T)]$$

be the minimum expected cost of filling *any* array with $i \cdot \sqrt{n}$ empty cells, and which contains a low/high number of gaps, for $i \in \{0, 1, \ldots, \sqrt{n}\}$, respectively $i \in \{0, 1, \ldots, \sqrt{n} - 1\}$. (Note that $\mathcal{H}(\sqrt{n})$ is undefined as an empty array cannot have a high number of gaps.)

Combining Equation (1) with Lemma 9 we obtain

$$\mathcal{L}(i) \geq \left(1 - \frac{1}{\sqrt{n}}\right) \cdot \min\{3/16 + \mathcal{L}(i-1),\, \mathcal{H}(i-1)\},$$
$$\mathcal{H}(i) \geq 1/16 + \left(1 - \frac{1}{\sqrt{n}}\right) \cdot \min\left\{\mathcal{L}(i-1),\, \mathcal{H}(i-1)\right\},$$

with $\mathcal{L}(0) = \mathcal{H}(0) = 0$. Our next lemma, proved by induction (see Appendix A), leads to the lower bound.

▶ **Lemma 11.** $\mathcal{L}(\sqrt{n}) \in \Omega(\sqrt{n})$.

As the empty mapping $\emptyset$ is contained in $\mathcal{L}$ we have $\mathbb{E}[f(\emptyset)] \geq \mathcal{L}(\sqrt{n}) \in \Omega(\sqrt{n})$. Yao's minmax principle [21, Prop. 2.6] implies the lower bound on the expected cost of randomized algorithms for a worst-case input.

3 Competitiveness for online TSP

We now consider the generalization of online sorting that we call *online TSP*. Given a sequence $X = (x_1, \ldots, x_n) \in S^n$ for some metric space S, and a bijection $f\colon [n] \to [n]$, and $A[i] = x_{f(i)}$, let $\mathtt{D}_f(X) = \sum_{i=1}^{n-1} d(A[i+1], A[i])$. Here, $d(\cdot,\cdot)$ is a metric over S. As before, $\mathtt{OPT}(X) = \min_f \mathtt{D}_f(X)$, i.e., the *offline optimum*, and $\mathcal{A}(X)$ is the cost $\mathtt{D}_f(X)$ for a function f constructed by an online algorithm $\mathcal{A}$ on input sequence X. We define $\mathtt{C}_{\mathcal{A}}$ and $\mathtt{C}$ as before.

Our main interest is in the case $S = \mathbb{R}^d$, and particularly $d = 2$, with $d(\cdot,\cdot)$ the Euclidean distance. As a tool in the study of the Euclidean case, we first look at a simpler, *uniform metric* problem in § 3.1, showing a tight $\Theta(\log n)$ bound on the competitive ratio. Then, in § 3.2 we study the Euclidean $\mathbb{R}^2$ and $\mathbb{R}^d$ cases. As the uniform metric case is natural in itself, we revisit it in § 3.3 in the setting where the array size is larger than n.

3.1 Uniform metric

Let S be an arbitrary discrete set and consider the distance function $d(x, y) = 0$ if $x = y$ and $d(x, y) = 1$ otherwise. Let $K = K(X)$ denote the number of distinct entries in the input sequence X, i.e., $K = |\{x_1, \ldots, x_n\}|$. We give instance-specific bounds on the cost in terms of K and n. The following claim is obvious.

▷ Claim 12. $\mathtt{OPT}(X) = K - 1$.

Next, we give a bound on the online cost and show that it is asymptotically optimal.

▷ Claim 13. There is an online algorithm $\mathcal{A}$ with cost $\mathcal{A}(X) \leq K \log_2 n$.

▷ Claim 14. For every (possibly randomized) algorithm $\mathcal{A}$ and all $K \geq 3$, there is an input distribution X with $K = K(X)$ such that $\mathbb{E}[\mathcal{A}(X)] \in \Omega(K \log n)$.

Note that the condition $K \geq 3$ is essential; if $K = 1$, then $\mathcal{A}(X) = \mathtt{OPT}(X) = 0$ for every algorithm $\mathcal{A}$, and if $K = 2$, then there is an online algorithm $\mathcal{A}$ that achieves $\mathcal{A}(X) = \mathtt{OPT}(X) = 1$. (Place the two types of elements at the opposite ends of the array.)

Claims 12, 13, 14 together yield the main result of this subsection.

▶ **Theorem 3.** *The competitive ratio of online sorting of n items under the uniform metric is $\Theta(\log n)$. The upper bound $O(\log n)$ is achieved by a deterministic algorithm and the lower bound $\Omega(\log n)$ also holds for randomized algorithms.*

Proof of Claim 13. We describe algorithm $\mathcal{A}$ (see Appendix B for an alternative), noting that it is not required to know K in advance. Assume without loss of generality that $\{x_1, \ldots, x_n\} = \{1, 2, \ldots, K\}$. For each j, with $1 \leq j \leq K$, maintain a *cursor* $c_j \in [n]$ indicating the array cell where the next item x_i is placed if it equals j. More precisely, if $x_i = j$, then let $f(c_j) = i$, and move the cursor to the right: $c_j = \min\{c_j + 1, n\}$.

If $f(c_j)$ is already assigned (i.e., the cell $A[c_j]$ is already written), set c_j to the mid-point of the largest empty contiguous interval. Similarly, when j is encountered the first time, then initialize c_j at the mid-point of the largest empty contiguous interval. Thus, initially, $c_{x_1} = \lfloor n/2 \rfloor$, i.e., place the first element at the middle of the array.

Clearly, $\mathcal{A}$ can always place x_i *somewhere*, so $\mathcal{A}$ correctly terminates. It remains to prove the upper bound on the number of unequal neighbors at the end of the process.

We model the execution of $\mathcal{A}$ as a **coin-game**. Consider a number of up to K *piles* of coins. The game starts with a single pile of n coins. An adversary repeatedly performs one of two possible operations: (1) remove one coin from an arbitrary pile, (2) split the *largest* pile into two equal parts. Operation (2) is only allowed when the number of piles is less than K, and only for a pile of at least two coins. The game ends when all coins have been removed.

It is easy to see that this game models the execution of $\mathcal{A}$ in the sense that for any execution of $\mathcal{A}$ on X there is an execution of the coin-game in which the sizes of the piles correspond at each step to the lengths of the contiguous empty intervals in the array. Moreover, the number of consecutive unequal pairs at the end of algorithm $\mathcal{A}$ (i.e., the cost $\mathcal{A}(X)$) is at most the number of splits (i.e., operations (2)) of the coin-game execution. It is thus sufficient to upper bound the number of splits in *any* execution of the coin game.

Let n_i denote the size of a pile before its split, for the i-th split operation. As n_i is the size of the largest pile and piles can only get smaller, the sequence n_i is non-increasing. Suppose the i-th split replaces a pile of size t with two piles of size $t/2$. Then, after the split there are at most $K - 2$ piles with sizes in $[t/2, t]$ and no pile greater than t. Thus, after at most $K - 2$ further splits, we split a pile of size at most $t/2$. (Possible operations (1) can only strengthen this claim, as they make some piles smaller.) It follows that $n_{i+K} \leq n_i/2$ for all i. As $n_1 \leq n$, and $n_i \geq 1$ for all i, the number of splits is at most $K \log_2 n$. ◀

Proof of Claim 14. Let $\mathcal{A}$ be an algorithm filling the items into an array A of size n. We present a distribution over inputs X incurring cost $\mathbb{E}[X] \geq \Omega(K \log n)$. By Yao's minmax principle [21, Prop. 2.6] we thus get that every randomized algorithm has a worst-case input of cost $\Omega(K \log n)$.

For each free cell $A[i]$ we say that $A[i]$ is *friendly* for the two (possibly identical) elements y_k and y_l that are placed closest to the left and right of $A[i]$ in A. When asked to place an element y into A, $\mathcal{A}$ will increase the cost of the partial solution *unless* y is placed in a cell that is friendly for y. When placing y in a cell that is friendly for y, the number of friendly cells for y will decrease by one. The number of friendly cells for other elements may or may not decrease, but no element will have more friendly cells than before the insertion of y.

Let $z_1 \leq z_2 \leq \cdots \leq z_K$ be the number of friendly cells for each of the K values, sorted nondecreasingly. With n' free cells in A, $\sum_{i=1}^{K} z_i \leq 2n'$, and hence $z_1 \leq 2n'/K$. Consider the median $z_{\lfloor K/2 \rfloor}$. As $\sum_{i=\lfloor K/2 \rfloor}^{K} z_i \leq 2n'$, we have $z_1 \leq \cdots \leq z_{\lfloor K/2 \rfloor} \leq \frac{2n'}{\frac{1}{2}K} = 4n'/K$.

We now describe our input distribution, proceeding in *epochs*: At the start of each epoch a value y is chosen uniformly at random from $\{1, \ldots, K\}$, and this element is presented all through the epoch. If $K \in \{3, 4\}$, the epoch will consist of $2n'/K$ copies of y, where n' is the number of unoccupied cells at the start of the epoch. By the first observation above, $\Pr[z_y \leq 2n'/K] \geq 1/K \geq 1/4$, in which case the input forces $\mathcal{A}$ to increase the cost of the partial solution.

If $K > 4$, we instead let the epoch be of length $4n'/K$, and by the second observation above we obtain that $\Pr[z_y \leq 4n'/K] \geq 1/2$, again increasing the cost of the partial solution with constant probability.

To establish a lower bound for the expected cost produced by this input, it remains to lower bound the number of epochs processed. For small K, the epoch leaves $n' \cdot \frac{K-2}{K}$ free cells for the coming epochs. For $K > 4$ the epoch leaves $n' \cdot \frac{K-4}{K}$ free cells. Thus, in both cases, the number of epochs is at least

$$\log_{\frac{K}{K-4}}(n) = \frac{\log_2(n)}{\log_2\left(1 + \frac{4}{K-4}\right)} \geq \frac{K-4}{4}\log_2(n)\,.$$

Hence, the input will force $\mathcal{A}$ to produce a solution of expected cost $\Omega(K \log n)$. ◀

3.2 Online TSP in $\mathbb{R}^d$

We now proceed to the case where $S = \mathbb{R}^d$ and $d(\cdot,\cdot)$ is the Euclidean distance. We start with the first new case, $d = 2$. For ease of presentation, we omit floors and ceilings in the analysis. We assume that the input points are from the unit box $[0,1]^2$ and that the optimum length is at least 1. These assumptions can be relaxed by a similar doubling-approach as in the proof of Claim 7.

The following result is well known [5, 17, 12] (consider, e.g., a $\sqrt{n} \times \sqrt{n}$ uniform grid).

▷ Claim 15. For all sequences X of n points in $[0,1]^2$, we have $\texttt{OPT}(X) \in O(\sqrt{n})$. Moreover, there exists a sequence X of n poins in $[0,1]^2$ such that $\texttt{OPT}(X) \in \Omega(\sqrt{n})$.

As a warm-up before our competitiveness result, we give an upper bound on the cost of an online algorithm and we show the tightness of this bound. The arguments extend the one-dimensional ones in a straightforward way. The proofs of the following can be found in Appendix B.

▷ Claim 16. There is an online algorithm $\mathcal{A}$ such that $\mathcal{A}(X) \in O(n^{2/3})$ for all $X \in [0,1]^2$.

▷ Claim 17. For every deterministic $\mathcal{A}$ there is an input X such that $\mathcal{A}(X) \in \Omega(n^{2/3})$.

Now we move to the study of the competitive ratio. Note that the lower bound of Claim 8 immediately applies to our setting. Combined with Claim 16, it follows that $\texttt{C} \in O(n^{2/3}) \cap \Omega(n^{1/2})$. In the following we (almost) close this gap, proving the following.

▷ Claim 18. The competitive ratio of online TSP in $\mathbb{R}^2$ is $O(\sqrt{n \log n})$.

Partition the box $[0,1]^2$ into $t \times t$ boxes of sizes $1/t \times 1/t$, for t to be set later. Let K be the number of boxes that are *touched*, i.e., that contain some input point x_i. We first need to lower bound the optimum.

▶ **Lemma 19.** $\texttt{OPT}(X) \geq K/4t$.

Proof. If $K/4t < 1$, we are done, as $\texttt{OPT}(X) \geq 1$ by assumption. Assume therefore $K \geq 4t$. Choose an arbitrary representative point of X from each touched box. Observe that the optimum cannot be shorter than the optimal tour of the representatives (by triangle inequality). Among any five consecutive vertices of the representatives-tour, two must be from non-neighboring boxes, thus the total length of the four edges connecting these points is at least $1/t$. It follows that at least $K/4$ edges have this length, yielding the claim. ◀

Now we need an algorithm that does well in terms of K. For any two neighboring entries in the array we consider only whether they come from the same box. This allows us to reduce our problem to the uniform metric case (Claim 13).

▷ Claim 20. There is an online algorithm $\mathcal{A}$ such that $\mathcal{A}(X) \in O(K \log n + n/t)$.

Proof. We treat points from the same box as having the *same value*. We run the algorithm for the uniform metric given in Claim 13, incurring a total number of $O(K \log n)$ differing pairs of neighbors. For these pairs we account for a maximum possible cost of $\sqrt{2}$. All other neighboring pairs have the same value (= come from the same box), incurring a cost of at most $O(1/t)$ each, for a total of $O(n/t)$. This completes the proof. ◀

Together with Lemma 19, this yields Claim 18. Indeed, set $t = \sqrt{\frac{n}{\log n}}$. If $K \leq t$, then using $\texttt{OPT} \geq 1$, we have $\mathcal{A}/\texttt{OPT} \leq \frac{K \log n + n/t}{1} \in O(\sqrt{n \log n})$. If $K > t$, then using Lemma 19, $\mathcal{A}/\texttt{OPT} \leq \frac{K \log n + n/t}{K/4t} = 4t \log n + 4n/K \in O(\sqrt{n \log n})$.

Online TSP in $\mathbb{R}^d$. We now extend the bound on the competitive ratio (Claim 18) to the higher dimensional case, also considering the dependence on d, leading to the claimed result.

▶ **Theorem 2.** *There is a deterministic algorithm for online TSP in $\mathbb{R}^d$ with competitive ratio $\sqrt{d} \cdot 2^d \cdot O(\sqrt{n \log n})$.*

Analogously to Claim 15, we first state an absolute bound on the optimal TSP cost in d dimensions, treating d as a constant [5, 17, 12].

▷ Claim 21. For all sets X of n points in $[0,1]^d$, we have $\texttt{OPT}(X) \in O(n^{1-1/d})$. Moreover, there exists a set X of n poins in $[0,1]^d$ such that $\texttt{OPT}(X) \in \Omega(n^{1-1/d})$.

A straightforward generalization of Claims 16 and 17 yields (for all constant d):

▷ Claim 22. There is an online algorithm $\mathcal{A}$ for online TSP in $\mathbb{R}^d$ such that $\mathcal{A}(X) \in O(n^{1-\frac{1}{d+1}})$ for all X. For every deterministic algorithm $\mathcal{A}$ there is an input X such that $\mathcal{A}(X) \in \Omega(n^{1-\frac{1}{d+1}})$.

Proof of Theorem 2. We follow the proof of Claim 18, with minor changes. We assume the input to come from the d-dimensional unit box $[0,1]^d$. We partition this box into t^d boxes of sizes $(1/t)^d$. When adapting Lemma 19, we have to consider 2^d (instead of 4) consecutive tour edges, thus obtaining $\texttt{OPT}(X) \geq \frac{K}{2^d t}$. When adapting Claim 20, our upper bound increases by a factor of $\sqrt{d}$, the distance between antipodal vertices of a unit d-cube, replacing the implicit $\sqrt{2}$ in the earlier bound. Thus, when bounding the competitive ratio, we incur an overall factor of $\sqrt{d} \cdot 2^d$, yielding the result. ◀

3.3 Uniform metric with a larger array

We study the problem of placing n elements $\{1, \ldots, K\}$ into an array of size $\lceil \gamma n \rceil$ for a fixed $\gamma > 1$. The algorithm and distribution from Claims 13 and 14 can be reused in this setting, stopping either process after the insertion of the first n elements.

▷ Claim 23. For every algorithm $\mathcal{A}$ and any number $K \geq 3$ of input values, there is an input distribution X such that $\mathbb{E}[\mathcal{A}(X)] \in \Omega(K \cdot (1 + \log(\gamma/(\gamma - 1))))$.

▷ Claim 24. There is an online algorithm $\mathcal{A}$ with cost $\mathcal{A}(X) \in O(K \cdot (1 + \log(\gamma/(\gamma - 1))))$.

The two claims together imply Theorem 4. We defer their proofs to Appendix B.1. They rely on similar arguments as the proofs of the claims mentioned above, with a refined argument for counting the number of epochs/moves performed.

4 Competitiveness for stochastic online sorting

In this section, we prove Theorem 5 and Theorem 6 (§ 4.1).

▶ **Theorem 5.** *There is an algorithm for online sorting of n items drawn independently and uniformly at random from $(0,1]$ that achieves competitive ratio $O((n\log n)^{1/4})$ with probability at least $1-2/n$.*

We start by describing the general design of the algorithm from Theorem 5 and showing some fundamental properties. We then describe a recursive algorithm and give the parameterizations that achieve the desired competitive cost.

The general design. We let α and β denote two parameters in $(0,1)$ which we will set later. We decompose the array A of n cells as such: the first $N = n - n^\beta$ cells are divided into $M = n^\alpha$ consecutive sub-arrays $A_1, \ldots, A_{n^\alpha}$, and the remaining n^β cells (at the end of the array) form one single subarray denoted by B. We refer to each of the sub-arrays A_i as a *bucket* and to B as the *backyard*. We note that each bucket A_i can hold $C = N/M$ elements, which we refer to as the *capacity* of the bucket.

We use the values of the elements to hash them into the array. Namely, for an element $x \in (0,1)$, we define $h\colon (0,1] \to \{1,\ldots,M\}$ by setting $h(x) = \lceil x \cdot M \rceil$. In other words, the elements in the interval $(0, 1/M]$ will all hash to bucket 1, elements in the interval $(1/M, 2/M]$ will hash to bucket 2, etc. Since the elements are chosen independently and uniformly at random from $(0,1]$, we get that h assigns the elements independently and uniformly at random into the M buckets.[6]

$\mathsf{SortUnif}_1(A,n)$: the first algorithm. Let $\mathsf{SortDet}(A,n)$ denote the deterministic algorithm from [1]. We now define the algorithm $\mathsf{SortUnif}_1(A,n)$ as such: upon receiving x, it checks if the bucket $A_{h(x)}$ has any empty cells. If so, it forwards x to $\mathsf{SortDet}(A_{h(x)}, C)$. Otherwise, it forwards x to $\mathsf{SortDet}(B, n^\beta)$ (and we say that the corresponding bucket is *full*).

We first prove that $\mathsf{SortUnif}_1(A,n)$ successfully places all items with high probability. Note that this is not always guaranteed: it could happen that both the bucket and the backyard are full (before we have managed to place all the n elements). If this happens, then among all n elements, strictly less than C hash into some bucket. We call this event a *failure* and show the following by employing a Chernoff bound (Appendix C):

▷ Claim 25. Given any $c > 0$, if $\beta \geq \frac{1}{2} \cdot \left(1 + \alpha + \frac{\ln\ln n + \ln(2(c+1))}{\ln n}\right)$, then $\mathsf{SortUnif}_1(A,n)$ fails with probability at most $1/n^c$.

We now bound the cost of $\mathsf{SortUnif}_1(A,n)$:

▷ Claim 26. If $\mathsf{SortUnif}(A,n)$ does not fail, then its cost is at most $O(\sqrt{C} + n^{\beta/2})$.

Proof. Since each bucket receives at least C elements, the cost of placing elements inside bucket A_1 is given by the cost of $\mathsf{SortDet}(A_1, C)$ on elements from $(0, 1/M)$. We bound this by $O(\sqrt{C} \cdot 1/M]$. For the remaining buckets, the elements are drawn from $(i/M, (i+1)/M)]$. This is equivalent to sorting elements from $(0, 1/M]$, and so their cost is the same as that of A_1. Therefore, in total, the cost from each individual bucket is $O(\sqrt{C})$. In addition, we also have the cost from crossing from one bucket to the next. This is at most $2/M$, since the maximum difference between elements from consecutive buckets is at most $2/M$. Since there are M buckets, this amounts to a cost of at most 2. The cost of crossing from bucket

[6] We assume that there is no element of value 0. The probability of this happening is 0.

A_M to the backyard is 1. Finally, the cost in the backyard is $O(n^{\beta/2})$, since it employs $\mathsf{SortDet}(B, n^\beta)$ on elements from $(0,1)$. ◀

We instantiate α and β such that we get the following (proof in Appendix C):

▶ **Lemma 27.** *Given any $c > 0$, $\mathsf{SortUnif}_1(A, n)$ has cost at most $O(n^{1/3} \cdot \ln^{1/6} n \cdot (2(c+1))^{1/6})$ with probability at least $1 - 1/n^c$.*

$\mathsf{SortUnif}_k$: recursing on the buckets. We take advantage of the fact that within each bucket, the elements are chosen uniformly at random. That is, we can apply the same strategy recursively inside the bucket. We get a series of algorithms $\mathsf{SortUnif}_k$ for $k \geq 2$. In $\mathsf{SortUnif}_k$, we let α and β be defined as in $\mathsf{SortUnif}_1$. When we see an element x with $h(x) = i$, if its bucket A_i is not full, we place $(xM - i + 1)$ using $\mathsf{SortUnif}_{k-1}(A_{h(x)}, C)$. Note that we have already conditioned on the fact that $\lceil x \cdot M \rceil = i$. In this case, $(xM - i + 1)$ becomes uniformly distributed in $(0,1)$. If A_i is full, we place x in the backyard according to $\mathsf{SortDet}(B, n^\beta)$.

Note that there are now several causes for failure: either some bucket A_i is not full, or one of the algorithms inside the bucket fails. We bound the probability of either of these happening as follows, proving the claim by induction over k (Appendix C):

▶ **Lemma 28.** *Given any $c > 0$, $\mathsf{SortUnif}_k(A, n)$ has cost at most*

$$O\left(n^{1/f_k} \cdot \ln^{1/4} n \cdot (2(c+1))^{B_k}\right)$$

with probability at least $1 - 2/n^c$, where $f_k = 4 - 1/2^{k-1}$ and $B_k = k/4$.

Setting $c = 1$ and $k = \log_2\left(2/(4 + \ln n - \sqrt{\ln^2 n + 8 \ln n})\right)$ yields the bounds claimed in Theorem 5; the details are given in Appendix C.

4.1 Stochastic online sorting in larger arrays

▶ **Theorem 6.** *For any $\gamma > 1$, there is an algorithm for online sorting of n items drawn independently and uniformly from $(0,1)$ into an array of size $\lceil \gamma n \rceil$ that achieves expected competitive ratio $O\left(1 + \frac{1}{\gamma - 1}\right)$.*

The algorithm. Define $\alpha = (\gamma - 1)/10$ and $\beta = \gamma - \alpha$ such that array A has size $(\beta + \alpha)n$. We will refer to the final αn cells of A as the *buffer*. The first βn cells we consider to correspond to the interval $[0,1)$. More specifically, cell $A[i]$ represents the interval $\left[\frac{i}{\beta n}, \frac{i+1}{\beta n}\right)$.

Let $h\colon [0,1) \to \{0, \beta n - 1\}$ be the function which maps values $x \in [0,1)$ to the index of the corresponding cell: $h(x) = \left\lfloor \frac{x}{\beta n} \right\rfloor$. Our algorithm $\mathcal{A}$ inserts each value x into $A[h(x)]$ if the cell is available, and otherwise into the *first* available cell following $h(x)$, possibly wrapping around from $A[\gamma n - 1]$ to $A[0]$ (although, as we argue in Appendix C.1, this is unlikely). In other words, x is inserted into the cell $A[h(x) + i \mod \gamma n]$ where i is the smallest non-negative integer such that the specified cell is available.

Counting steps. As the values are drawn independently and uniformly from $[0,1)$, the indices $h(x)$ are also independently and uniformly distributed in $\{0, \ldots, \beta n - 1\}$. The algorithm thus mirrors the scheme of linear probing commonly used for implementing hash tables. In linear probing each key x that is to be added to the table (and which is generally not assumed to come from a known distribution) is hashed to an index of the array, and x is inserted into the first following free cell.

When implementing a hash table, one is interested in analyzing the *speed* of insertions, which for linear probing corresponds to the number of cells that are probed before a free cell is found. Although speed is not a concern for our online algorithm, we will nonetheless show that the number of steps performed is an important measure for bounding the cost of our solution. Formally, let $s(k)$ be the number of steps performed when inserting the k-th value x_k of the input. If $A[h(x_k)]$ is free, then $s(k) = 1$; otherwise $s(k) = i + 1$, when x_k is inserted into cell $A[h(x_k) + i \mod \gamma n]$.

Now consider the linear probing process $\mathcal{A}_{LP}$ inserting n elements into a table T of size βn (that is, without the buffer space). Analogously to $s(i)$ we let $s_{LP}(i)$ be the number of steps performed by $\mathcal{A}_{LP}$ when inserting the i-th element. We couple the processes $\mathcal{A}$ and $\mathcal{A}_{LP}$ so that they encounter the same stream of indices $h(x_1), h(x_2), \ldots, h(x_n)$ during execution. We then have $s(i) \leq s_{LP}(i)$. Indeed, the only difference between the processes is that $\mathcal{A}$ has an extra αn cells of buffer space to prevent wraparound at the end of A. More generally, we thus have $\mathbb{E}[s(i)] \leq \mathbb{E}[s_{LP}(i)]$. Combining the pieces we get that:

▷ Claim 29. $\mathbb{E}[\mathcal{A}(X)] \leq O\left(1 + \frac{1}{\beta n} \sum_{i=1}^{n} \mathbb{E}[s(i)]\right)$.

The proof can be found in Appendix C.1. We then invoke the following classic result by Knuth [18] to bound $\sum_i \mathbb{E}[s_{LP}(i)]$:

▶ **Theorem 30** ([18]). *Consider the process of inserting $(1-\varepsilon)m$ elements into an array of size m by linear probing. When hash values are assigned uniformly and independently,*

$$\sum_{i=1}^{(1-\varepsilon)m} \mathbb{E}[s_{LP}(i)] \in O\left((1-\varepsilon)m \cdot \frac{1}{\varepsilon}\right).$$

To apply Theorem 30, set $m = \beta n$ and $\varepsilon = (\beta - 1)/\beta$ such that $(1-\varepsilon)m = n$, and thus $\sum_{i=1}^{n} \mathbb{E}[s(i)] \leq \sum_{i=1}^{n} \mathbb{E}[s_{LP}(i)] \in O\left(n \cdot \left(\frac{\beta}{\beta-1}\right)\right)$. As $\beta - 1 = \frac{9}{10} \cdot (\gamma - 1)$, we have $\mathbb{E}[\mathcal{A}(X)] \in O\left(1 + \frac{1}{\beta} \cdot \frac{\beta}{\beta-1}\right) \subseteq O\left(1 + \frac{1}{\gamma-1}\right)$, proving Theorem 6. ◀

5 Conclusion and open questions

In this paper, we studied the online sorting problem in some of its variants. Several questions and directions to further study remain, we mention those that we find most interesting.

Online sorting of reals. For large arrays ($m = \lceil \gamma n \rceil$, for $\gamma > 1$) significant gaps remain between the upper and lower bounds on the competitive ratio of online sorting (see [1]).

In a different direction, consider online sorting and *online list labeling*, mentioned in § 1. These can be seen as two extremes of an "error vs. recourse" trade-off: online sorting allows no recourse, while list labeling allows no error. Achieving a smooth trade-off between the two optimization problems by some hybrid approach is an intriguing possibility.

Stochastic and other models. In the stochastic setting of online sorting, we could improve upon the worst-case bound, but the obtained ratio (Theorem 5) is likely not optimal; we are not aware of nontrivial lower bounds. For stochastic online sorting with large arrays (Theorem 6), a matching lower bound is likewise missing. We restricted our study to uniform distribution over an interval; extending the results to other distributions remains open.

Other models that go beyond the worst-case assumption could yield further insight; we mention two possible models: (1) online sorting *with advice* (e.g., from a machine learning model): suppose that each input item comes with a possibly unreliable estimate of its rank

in the optimal (sorted) sequence; what is the best way to make use of this advice? (2) online sorting *with partially sorted input*: for instance, suppose that the *input cost* $\sum_{i=1}^{n-1} |x_{i+1} - x_i|$ is small. What is the best guarantee for the online sorting cost in this case?

Online TSP in various metrics. For online TSP in $\mathbb{R}^d$, for fixed d, a small gap remains between the lower bound $\Omega(\sqrt{n})$ and upper bound $O(\sqrt{n \log n})$ on the competitive ratio (Theorem 2). The dependence on d (when $d \in \omega(1)$) is not known to be optimal. Online TSP in $\mathbb{R}^d$ with large array, and/or with stochastic input are also interesting directions.

Some non-Euclidean metrics pose natural questions, e.g., L_0 or L_∞ in $\mathbb{R}^d$, or tree- or doubling metrics. More broadly, is the optimal competitive ratio $O(\sqrt{n})$ for arbitrary metrics?

References

1. Anders Aamand, Mikkel Abrahamsen, Lorenzo Beretta, and Linda Kleist. Online sorting and translational packing of convex polygons. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 1806–1833. SIAM, 2023. URL: `https://doi.org/10.1137/1.9781611977554.ch69`, `doi:10.1137/1.9781611977554.CH69`.
2. Yuriy Arbitman, Moni Naor, and Gil Segev. Backyard cuckoo hashing: Constant worst-case operations with a succinct representation. In *2010 IEEE 51st Annual symposium on foundations of computer science*, pages 787–796. IEEE, 2010.
3. Giorgio Ausiello, Esteban Feuerstein, Stefano Leonardi, Leen Stougie, and Maurizio Talamo. Algorithms for the on-line travelling salesman 1. *Algorithmica*, 29:560–581, 2001.
4. Eric Balkanski, Yuri Faenza, and Noémie Périvier. The power of greedy for online minimum cost matching on the line. In *Proceedings of the 24th ACM Conference on Economics and Computation*, pages 185–205, 2023.
5. Jillian Beardwood, J. H. Halton, and J. M. Hammersley. The shortest path through many points. *Mathematical Proceedings of the Cambridge Philosophical Society*, 55(4):299–327, 1959. `doi:10.1017/S0305004100034095`.
6. Michael A Bender, Richard Cole, Erik D Demaine, Martin Farach-Colton, and Jack Zito. Two simplified algorithms for maintaining order in a list. In *European Symposium on Algorithms*, pages 152–164. Springer, 2002.
7. Michael A Bender, Alex Conway, Martín Farach-Colton, Hanna Komlós, William Kuszmaul, and Nicole Wein. Online list labeling: Breaking the log 2 n barrier. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 980–990. IEEE, 2022.
8. Michael A. Bender, Martin Farach-Colton, John Kuszmaul, William Kuszmaul, and Mingmou Liu. On the optimal time/space tradeoff for hash tables. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1284–1297. ACM, 2022. `doi:10.1145/3519935.3519969`.
9. Antje Bjelde, Jan Hackfeld, Yann Disser, Christoph Hansknecht, Maarten Lipmann, Julie Meißner, Miriam Schlöter, Kevin Schewior, and Leen Stougie. Tight bounds for online tsp on the line. *ACM Transactions on Algorithms (TALG)*, 17(1):1–58, 2020.
10. Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.
11. Paul F Dietz. Maintaining order in a linked list. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 122–127, 1982.
12. L. Few. The shortest path and the shortest road through n points. *Mathematika*, 2(2):141–144, 1955. `doi:10.1112/S0025579300000784`.
13. Amos Fiat and Gerhard J. Woeginger. On-line scheduling on a single machine: Minimizing the total completion time. *Acta Informatica*, 36(4):287–293, 1999. URL: `https://doi.org/10.1007/s002360050162`, `doi:10.1007/S002360050162`.

14 Anupam Gupta, Guru Guruganesh, Binghui Peng, and David Wajc. Stochastic online metric matching. In *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2019.

15 Anupam Gupta and Kevin Lewi. The online metric matching problem for doubling metrics. In *Automata, Languages, and Programming: 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I 39*, pages 424–435. Springer, 2012.

16 Alon Itai, Alan G Konheim, and Michael Rodeh. A sparse table implementation of priority queues. In *International Colloquium on Automata, Languages, and Programming*, pages 417–431. Springer, 1981.

17 Howard J. Karloff. How long can a euclidean traveling salesman tour be? *SIAM J. Discret. Math.*, 2(1):91–99, 1989. doi:10.1137/0402010.

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

19 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.

20 Nicole Megow, Martin Skutella, José Verschae, and Andreas Wiese. The power of recourse for online mst and tsp. In *Automata, Languages, and Programming: 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I 39*, pages 689–700. Springer, 2012.

21 Rajeev Motwani and Prabhakar Raghavan. Randomized algorithms. In Mikhail J. Atallah, editor, *Algorithms and Theory of Computation Handbook*, Chapman & Hall/CRC Applied Algorithms and Data Structures series. CRC Press, 1999. URL: https://doi.org/10.1201/9781420049503-c16, doi:10.1201/9781420049503-C16.

22 Christos H Papadimitriou. The euclidean travelling salesman problem is np-complete. *Theoretical computer science*, 4(3):237–244, 1977.

23 Kirk Pruhs, Jirí Sgall, and Eric Torng. Online scheduling. In Joseph Y.-T. Leung, editor, *Handbook of Scheduling - Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004. URL: http://www.crcnetbase.com/doi/abs/10.1201/9780203489802.ch15, doi:10.1201/9780203489802.CH15.

24 Sharath Raghvendra. Optimal analysis of an online algorithm for the bipartite matching problem on a line. In *34th International Symposium on Computational Geometry (SoCG 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

25 Tim Roughgarden, editor. *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press, 2020. doi:10.1017/9781108637435.

26 Michael Saks. Online labeling: Algorithms, lower bounds and open questions. In *International Computer Science Symposium in Russia*, pages 23–28. Springer, 2018.

A Proofs from § 2

▷ Claim 7 (Proof in Appendix A). There is a deterministic online algorithm $\mathcal{A}$ for online sorting of an arbitrary sequence of n reals, with $\mathtt{C}_{\mathcal{A}} \in O(\sqrt{n})$.

Proof. Our task is to place n values $x_1, \ldots, x_n \in \mathbb{R}$ bijectively into the array cells $A[1], \ldots, A[n]$. We revisit and extend the algorithm from [1]. The main obstacle is that in [1] it can be assumed that $\mathtt{OPT} = 1$. In our case, $\mathtt{OPT} = \max_i \{x_i\} - \min_i \{x_i\}$; as this quantity is not known until the end, we estimate it based on the available data.

For ease of presentation assume that $x_1 = 0$. This is without loss of generality as we can simply transform x_i to $x_i - x_1$ for all $i = 1, \ldots, n$ and the costs of both offline and online solutions are unaffected.

Start by reading x_1, x_2 and placing them arbitrarily (this incurs a cost of at most $2\mathtt{OPT}$), and set $q = \lceil \log_2 |x_2 - x_1| \rceil = \lceil \log_2 |x_2| \rceil$ (as we assumed $x_1 = 0$). Throughout the algorithm we maintain q so that the input values seen so far are in the interval $J = [-2^q, 2^q]$. This clearly holds after reading x_2 . For simplicity, assume first that $x_3, \ldots, x_n \in J$, and q does not need to be updated.

Let $N_1 = \lfloor \sqrt{n} \rfloor$ and $N_2 = 2N_1$. Partition the interval J into N_1 equal-size subintervals $J_1, \ldots, J_{N_1}$, called "boxes", and partition the array A into N_2 equal-size subarrays $A_1, \ldots, A_{N_2}$, called "blocks" (as n is generally not divisible by N_2, we allow one block to be smaller).

Whenever a value $x_i \in J_k$ is received, if it is the first value from box J_k, then we assign J_k to a yet unassigned (empty) block A_t; we place x_i in the leftmost cell of A_t and continue placing future values from J_k into A_t (left-to-right). If the block A_t of J_k is full, we assign a new empty block to J_k.

If we run out of empty blocks, we must have processed at least $n/2$ values; to see this, observe that of the $N_2 = 2N_1$ blocks, at most $N_1 - 1$ are non-full, the rest have been filled. From this point on, we treat the remaining $n' \leq n/2$ empty cells of A as a virtual (contiguous) array A', recursively placing the remaining n' values into A' by the same strategy (we restart $\mathcal{A}$ on A', without the intitialization of q, i.e., we keep the current values of q and J).

The total cost $t(n)$ of $\mathcal{A}$ is at most

$$t(n) \leq t(n') + 2^{q+1} \cdot \frac{n}{N_1} + 2^{q^*+1} \cdot 2N_2.$$

The first term is the remaining cost of placing n' values into a contiguous array of size n'.

The second term is the cost due to at most n values already placed within full blocks. (As these come from one of the N_1 boxes of J and $|J| \leq 2^{q+1}$, the bound follows.)

The third term is the cost due to boundaries between blocks, and "at the interface" with the recursive call, i.e., at the margins of incomplete blocks. (This term thus captures the cost of "pretending" A' to be contiguous.) Here 2^{q^*+1} is an upper bound on the size of J throughout the entire execution of $\mathcal{A}$ (including future points), i.e., $q^* = \lceil \log_2 (\max_i (x_i) - \min_i (x_i)) \rceil = \lceil \log_2 \mathtt{OPT} \rceil$, and $q \leq q^*$.

We have $t(n) \leq t(n/2) + \mathtt{OPT} \cdot O(\sqrt{n})$, which resolves to $t(n) \in \mathtt{OPT} \cdot O(\sqrt{n})$. Notice that $\mathtt{OPT}$ is the *global optimum* of the problem, not the optimum in the current recursive call.

Suppose now that we receive a point $x_i \notin J$. We update

$$q = \lceil \log_2 (\max_{k \leq i} (x_k) - \min_{k \leq i} (x_k)) \rceil$$

and $J = [-2^q, 2^q]$. Note that the value of q increased, and all values $x_1, \ldots, x_i$ seen so far are also contained in the current J. We then split J into new boxes $J'_1, \ldots, J'_{N_1}$ (without

changing N_1 or N_2). The blocks $A_1, \dots, A_{N_2}$ remain, and if some old box J_i was assigned to a block A_j, then we assign the corresponding new box J'_i to A_j. With these changes, we continue with the algorithm.

Observe that at the margin of every non-filled block we may end up with neighbors not from the same box (i.e., one is from the old, one is from the new). Thus, their distance may be up to $|J| = 2^{q+1}$, so we incur an additional cost of at most $2^{q+1} \cdot N_1 \le 2^{q+1} \cdot \sqrt{n}$.

We may incur such an extra cost several times during the execution, but always for a different integer $q \le q^\star$. As $2^{q^\star-1} \le \texttt{OPT} \le 2^{q^\star+1}$, the total extra cost is at most $\sum_{q \le q^\star} 2^{q+1} \cdot \sqrt{n} \in \texttt{OPT} \cdot O(\sqrt{n})$, and the overall bound is unaffected. ◀

▶ **Lemma 11.** $\mathcal{L}(\sqrt{n}) \in \Omega(\sqrt{n})$.

Proof. Proceed by induction on i. For all $i' \in \{0, \dots, i-1\}$ assume

$$\mathcal{H}(i') \ge \left(1 - \frac{1}{\sqrt{n}}\right)^{i'} \cdot \frac{i'}{32},$$
$$\mathcal{L}(i') \ge \left(1 - \frac{1}{\sqrt{n}}\right)^{i'} \cdot \frac{i'-1}{32}.$$

Then

$$\begin{aligned}
\mathcal{H}(i) &\ge \frac{1}{16} + \left(1 - \frac{1}{\sqrt{n}}\right) \cdot \min\{\mathcal{L}(i-1), \mathcal{H}(i-1)\} \\
&\ge \frac{1}{16} + \left(1 - \frac{1}{\sqrt{n}}\right) \cdot \min\left\{\left(1 - \frac{1}{\sqrt{n}}\right)^{i-1} \cdot \frac{i-2}{32}, \left(1 - \frac{1}{\sqrt{n}}\right)^{i-1} \cdot \frac{i-1}{32}\right\} \\
&\ge \left(1 - \frac{1}{\sqrt{n}}\right)^{i} \cdot \frac{i}{32},
\end{aligned}$$

and

$$\begin{aligned}
\mathcal{L}(i) &\ge \left(1 - \frac{1}{\sqrt{n}}\right) \cdot \min\left\{\frac{3}{16} + \mathcal{L}(i-1), \mathcal{H}(i-1)\right\} \\
&\ge \left(1 - \frac{1}{\sqrt{n}}\right) \cdot \min\left\{\frac{3}{16} + \left(1 - \frac{1}{\sqrt{n}}\right)^{i-1} \cdot \frac{i-2}{32}, \left(1 - \frac{1}{\sqrt{n}}\right)^{i-1} \cdot \frac{i-1}{32}\right\} \\
&= \min\left\{\left(1 - \frac{1}{\sqrt{n}}\right) \cdot \frac{3}{16} + \left(1 - \frac{1}{\sqrt{n}}\right)^{i} \cdot \frac{i-2}{32}, \left(1 - \frac{1}{\sqrt{n}}\right)^{i} \cdot \frac{i-1}{32}\right\} \\
&\ge \left(1 - \frac{1}{\sqrt{n}}\right)^{i} \cdot \frac{i-1}{32}.
\end{aligned}$$

Finally, note that $(1 - \frac{1}{\sqrt{n}})^{\sqrt{n}} \ge 0.3$ for $n \ge 10$, proving the claim. ◀

B Proofs from § 3

An alternative algorithm for Claim 13. We now present a different algorithm, resembling the one from [1] and the one from the proof of Claim 7. If K is known, then we can proceed in a straightforward way: Partition the array into $2K$ blocks, assign individual values to blocks, and when we run out of free blocks, recurse on the remaining free space. We get a cost of $t(n) = O(K) + t(n/2)$, yielding $t(n) \in O(K \log n)$. Here the $O(K)$ term accounts for both the boundaries between blocks and the interface with the recursive call.

We can handle the case of unknown K by a doubling strategy, similar to the proof of Claim 7. Namely, start with $K = 2$, and run the above algorithm. When a new element arrives that increases the number of different values seen to $K + 1$, then set K to $2K$, and split each block in the current recursive level into two equal parts. A full block will be replaced by two full blocks, an empty block will be replaced by two empty blocks. In both cases, there is no additional cost.

A partially filled block will be replaced either by (1) a full block and an empty block, or (2) a full block and a partially filled block, or (3) a partially filled block and an empty block. In either case, if the block was assigned to a value t, then, if there is a resulting partially filled block, then it will continue to be filled with values t. Thus there is no incurred cost that is not accounted for already elsewhere, and the bound of $O(K \log n)$ holds.

▷ Claim 16. There is an online algorithm $\mathcal{A}$ such that $\mathcal{A}(X) \in O(n^{2/3})$ for all $X \in [0,1]^2$.

Proof. The algorithm closely follows the online sorting algorithm of [1] and the extension in Claim 7.

Divide the box $[0,1]^2$ into $n^{1/3} \times n^{1/3}$ *boxes* of sizes $n^{-1/3} \times n^{-1/3}$. Divide the array of size n into $2n^{2/3}$ equal size *blocks*. When a point arrives in a given box B, assign B to a previously unassigned block J, and place points from B into J (in arrival order, left-to-right). If J is full, assign a new block to B. If no more new blocks are available, re-use the empty space by treating it as if it were contiguous, and recursively assign blocks for the remaining (at most) $n/2$ input points.

The total cost can be bounded as $t(n) \leq O(n^{2/3}) + t(n/2) \in O(n^{2/3})$, where we accounted for the at most $O(n^{-1/3})$ distance between any two neighbors, and the at most $O(1)$ distance between the $O(n^{2/3})$ block-boundaries and boundaries at the interface between the first call and the recursive call. ◀

▷ Claim 17. For every deterministic $\mathcal{A}$ there is an input X such that $\mathcal{A}(X) \in \Omega(n^{2/3})$.

Proof. We adapt the lower bound strategy for online sorting from [1], with the $n^{2/3}$ allowed input points being the grid points of the $n^{1/3} \times n^{1/3}$ uniform grid spanning $[0,1]^2$. We present the adversary strategy against deterministic algorithms; a randomized oblivious adversary can be obtained using techniques similar to those in § 2.

In the first phase, as long as there is a grid point that does not appear in the array with an empty neighbor, output that point as the next input item. If all grid points appear with an empty neighbor, then move to phase two, and output $(0,0)$ points until the end. In the first phase each input item incurs cost at least $\Omega(n^{-1/3})$, thus, if we remain in the first phase until the end, then the total cost is $\Omega(n \cdot n^{-1/3}) = \Omega(n^{2/3})$. If we move to the second phase, then the cost is at least

$$\sum_{1 \leq i,j \leq n^{1/3}} \sqrt{i \cdot n^{-1/3} + j \cdot n^{-1/3}},$$

likewise yielding $\Omega(n^{2/3})$. ◀

B.1 Proofs from § 3.3

▷ Claim 23. For every algorithm $\mathcal{A}$ and any number $K \geq 3$ of input values, there is an input distribution X such that $\mathbb{E}[\mathcal{A}(X)] \in \Omega(K \cdot (1 + \log(\gamma/(\gamma - 1))))$.

Proof. We start by introducing one copy of each element $\{1, \ldots, K\}$, immediately causing a cost of $K - 1$. Then, we proceed as in the proof of Claim 14, choosing a type uniformly at random. Assume $K \geq 5$. With m' free cells left in the array we repeat the chosen element

$4m'/K$ times. By the arguments presented in the proof of Claim 14, this increases the cost of the partial solution with probability at least $1/2$. Next, a new element is chosen, and the process is repeated with $m'(1-4/K)$ free cells left in the array.

The question now is how many times we can repeat this procedure before having inserted n elements. Watching the process in reverse, it is seen that with m' cells currently free, we have caused an increase in cost (with probability $1/2$) during the previous $Km'/(K-4)$ insertions. The process is stopped when $m' = (\gamma-1)n$, and started at $m' = \gamma n - K$. Hence we can repeat the process

$$\begin{aligned}\log_{\frac{K}{K-4}}\left(\frac{\gamma n-K}{(\gamma-1)n}\right) &= \frac{\log_2\left(\frac{\gamma}{(\gamma-1)}-\frac{K}{(\gamma-1)n}\right)}{\log_2\left(\frac{K}{K-4}\right)}\\ &\geq \frac{K-4}{4}\cdot\log_2\left(\frac{\gamma}{(\gamma-1)}-\frac{K}{(\gamma-1)n}\right)\\ &\geq \Omega\left(K\cdot\log\left(\frac{\gamma}{\gamma-1}\right)\right)\end{aligned}$$

times. Thus the expected cost of $\mathcal{A}(X)$ will be $K-1+1/2\cdot\Omega(K\cdot\log(\gamma/(\gamma-1)))$.

The case $K\in\{3,4\}$ is lifted from the proof of Claim 14 in the same way: Letting each round be of length m'/K, rounds will cause an increase in cost with probability at least $1/K\geq 1/4$, and we can execute $\Omega(K\cdot\log(\gamma/(\gamma-1)))$ rounds. ◀

▷ Claim 24. There is an online algorithm $\mathcal{A}$ with cost $\mathcal{A}(X)\in O(K\cdot(1+\log(\gamma/(\gamma-1))))$.

Proof. We reuse the algorithm and coin-game from the proof of Claim 13, with the modification that the game starts with γn coins in a single pile and ends when $(\gamma-1)n$ coins are left. Generalizing the observation from the proof of Claim 13, we observe that if we at some point have c piles each of size at most t, then all piles will be of size at most $t/2$ after another c splits.

Applying this observation to the very start of the game, we initially have a single pile of γn coins, then two piles of at most $\gamma n/2$ coins each after the first split, then four piles of at most $\gamma n/4$ coins after another two splits, and so forth. Generally, $n_i\leq \gamma n/2^{\lfloor\log_2 i+1\rfloor}$.

From the above observation, $n_{2K}\leq\gamma n/K$, and from the proof of Claim 13 we have $n_{i+K}\leq n_i/2$. The early termination of the game means that $n_i\geq(\gamma-1)n/K$ for all i, thus

$$i\leq 2K+K\log_2\left(\frac{\gamma n}{K}\cdot\frac{K}{(\gamma-1)n}\right)=K\cdot\left(2+\log_2\left(\frac{\gamma}{\gamma-1}\right)\right).$$ ◀

C Proofs from § 4

▷ Claim 25. Given any $c>0$, if $\beta\geq\frac{1}{2}\cdot\left(1+\alpha+\frac{\ln\ln n+\ln(2(c+1))}{\ln n}\right)$, then $\mathsf{SortUnif}_1(A,n)$ fails with probability at most $1/n^c$.

Proof. Let $|A_i|$ denote the number of elements that hash (according to h) into a fixed bucket A_i, for some $i\in\{1,\ldots,M\}$. Then $\mathbb{E}[|A_i|]=n^{1-\alpha}$, since we have n elements hashing into $M=n^\alpha$ buckets. Recall that the capacity of each bucket is $C=N/M$, where $N=n-n^\beta$, and so $C=n^{1-\alpha}-n^{\beta-\alpha}$. We now apply a Chernoff bound for the lower tail of $|A_i|$:

$$\begin{aligned}\Pr[|A_i|<C]=\Pr\left[|A_i|<\left(1-1/n^{1-\beta}\right)\cdot n^{1-\alpha}\right]&\leq\exp\left(-1/2\cdot n^{1-\alpha}\cdot 1/n^{2-2\beta}\right)\\&=\exp\left(-1/2\cdot n^{2\beta-\alpha-1}\right).\end{aligned}$$

The claim follows by setting α and β such that the above probability is at most $1/n^{c+\alpha}$ and then doing a union bound over all buckets. ◀

▶ **Lemma 27.** *Given any $c > 0$, $\mathsf{SortUnif}_1(A, n)$ has cost at most $O(n^{1/3} \cdot \ln^{1/6} n \cdot (2(c+1))^{1/6})$ with probability at least $1 - 1/n^c$.*

Proof. We first optimize for the cost of $\mathsf{SortUnif}_1(A, n)$ from Claim 26. Note that the cost is (asymptotically) minimized when $n^\beta = \Theta(C)$, that is, when $n^\beta = \Theta(n^{1-\alpha})$. We thus set $\beta = 1 - \alpha$. We now verify the conditions from Claim 25 and set

$$\alpha = \frac{1}{3} - \frac{\ln\ln n + \ln(2(c+1))}{3\ln n}\,.$$

Then $n^{\frac{1-\alpha}{2}} = n^{\frac{1}{3}} \cdot n^{\frac{\ln\ln n + \ln(2(c+1))}{6\ln n}} = n^{\frac{1}{3}} \cdot \exp\left(\frac{\ln\ln n + \ln(2(c+1))}{6}\right)$ and the claim follows. ◀

▶ **Lemma 28.** *Given any $c > 0$, $\mathsf{SortUnif}_k(A, n)$ has cost at most*

$$O\left(n^{1/f_k} \cdot \ln^{1/4} n \cdot (2(c+1))^{B_k}\right)$$

with probability at least $1 - 2/n^c$, where $f_k = 4 - 1/2^{k-1}$ and $B_k = k/4$.

Proof. We prove the claim by induction on k. We assume that the statement is true for k and we then prove it for $k+1$. From the proof of Claim 25, we have that the buckets in $\mathsf{SortUnif}_{k+1}$ are not full with probability at most $1/n^c$ as long as

$$\beta \geq \frac{1}{2} \cdot \left(1 + \alpha + \frac{\ln\ln n + \ln(2(c+1))}{\ln n}\right)\,. \tag{2}$$

We now would like to have the same upper bound for the probability that $\mathsf{SortUnif}_k$ fails in some bucket. For that, we invoke the inductive hypothesis with failure probability at most $1/n^{c+1}$, and do a union bound over the n^α buckets. We implicitly assume here that $\alpha < 1$, so that we have $2n^\alpha \leq n$. We upper bound C by $n^{1-\alpha}$ and get that the cost for all the buckets is

$$O\left(n^{(1-\alpha)/f_k} \cdot \exp\left(1/4 \cdot \ln\ln n + B_k \cdot \ln(2(c+2))\right)\right)\,.$$

The cost across the buckets and the backyard remains $O(1)$, and the cost of the backyard is $O(n^{\beta/2})$ as before. We now equate

$$\frac{1-\alpha}{f_k} + \frac{1/4 \cdot \ln\ln n + B_k \ln(2(c+2))}{\ln n} = \frac{\beta}{2}$$

and then set

$$\alpha = \frac{4 - f_k}{4 + f_k} + \frac{f_k}{f_k + 4} \cdot \frac{4B_k \cdot \ln(2c+4) - \ln(2c+2)}{\ln n}$$

to satisfy Eq.(2). We would then get that

$$\frac{1-\alpha}{f_k} = \frac{2}{4+f_k} - \frac{1}{4+f_k} \cdot \frac{4B_k \cdot \ln(2c+4) - \ln(2c+2)}{\ln n}$$

and the exponent in the overall cost would be

$$\frac{2}{4+f_k} + \frac{1/4 \cdot \ln\ln n + B_k \cdot \ln(2c+4) - \frac{4B_k}{4+f_k} \cdot \ln(2c+4) + \frac{1}{4+f_k}\ln(2c+2)}{\ln n}\,.$$

Note that $2/(4+f_k) = 1/f_{k+1}$ by definition, so the only thing left to verify is that:

$$B_k \cdot \ln(2c+4) - \frac{4B_k}{4+f_k} \cdot \ln(2c+4) + \frac{1}{4+f_k} \ln(2c+2) \leq B_{k+1} \cdot \ln(2c+2) \; .$$

We upper bound $\ln(2c+4) \leq 2\ln(2c+2)$ and get that it is enough to show that

$$2B_k \cdot \left(1 - \frac{4}{4+f_k}\right) + \frac{1}{4+f_k} \leq B_{k+1} \; .$$

Finally, we use the fact that $1/8 \leq 1/(4+f_k) \leq 1/4$ to show that this inequality holds when we set $B_k = k/4$. ◀

▶ **Theorem 5.** *There is an algorithm for online sorting of n items drawn independently and uniformly at random from $(0,1]$ that achieves competitive ratio $O((n\log n)^{1/4})$ with probability at least $1-2/n$.*

Proof. To get an overall bound, we invoke Lemma 28 for $c=1$ and a value k that leads to a small cost. We set $x = 1/2^{k-1}$ and rewrite $k = \log_2(2/x)$ in the main term in the cost as follows:

$$n^{1/f_k} \cdot 4^{k/4} = n^{1/(4-x)} \cdot \sqrt{2/x} = \exp(\ln n/(4-x) + 1/2 \cdot \ln(2/x)) = \exp(\ln(2)/2 + f(x)) \; ,$$

where we defined $f(x) = \ln n/(4-x) - 1/2 \cdot \ln(x)$. This function is minimized at $x_0 = 4 + \ln n - \sqrt{\ln^2 n + 8\ln n}$, so we invoke $\mathsf{SortUnif}_{k_0}(A,n)$ for $k_0 = \log_2(2/x_0)$, yielding:

$$f(x_0) = \frac{\ln n}{\sqrt{\ln^2 n + 8\ln n} - \ln n} - \frac{1}{2} \cdot \ln(4 + \ln n - \sqrt{\ln^2 n + 8\ln n}) \leq \frac{\ln n}{4} + \frac{1}{2} \; .$$

Plugging $f(x_0)$ and k_0 back, we get an overall cost of $O((n\ln n)^{1/4})$. As the optimal (offline) cost is clearly $\Omega(1)$ with high probability, the claim follows. ◀

C.1 Proofs from § 4.1

In this subsection we prove the following claim characterizing the cost of the algorithm presented in § 4.1.

▷ Claim 29. $\mathbb{E}[\mathcal{A}(X)] \leq O\left(1 + \frac{1}{\beta n}\sum_{i=1}^{n} \mathbb{E}[s(i)]\right)$.

A central concept for our analysis is that of the *run*. A run denotes a maximal contiguous subarray of filled cells $A[i]$ through $A[j]$ such that an element $x \in \left[\frac{i}{\beta n}, \frac{j+2}{\beta n}\right)$ will be placed in cell $A[j+1]$. Generally, long runs are to be avoided as they make it more likely that an element x is placed far away from $A[h(x)]$, which makes it more difficult to bound the cost incurred by neighboring elements.

The issue of wraparound. When analyzing linear probing one can often disregard the issue of wraparound as a technical corner case which has no impact on the procedure at hand other than to complicate notation and definitions. For our use case however, wraparound would mean that we place a large value at the start of the array – a region that we, intuitively, have otherwise been filling with small values.

Wraparound could thus incur quite a penalty, and we will prove that, w.h.p., no wraparound will happen. In this way, wraparounds can be disregarded in the following arguments. Formally, let $\mathcal{W}$ denote the event that $\mathcal{A}$ experiences wraparound during execution, so that some element x is placed in a cell of index smaller than $h(x)$. We then compute the expected cost of $\mathcal{A}(X)$ as $\mathbb{E}[\mathcal{A}(X)] = \mathbb{E}[\mathcal{A}(X) \cdot [\mathcal{W}]] + \mathbb{E}[\mathcal{A}(X) \cdot [\neg\mathcal{W}]]$.

▷ Claim 31.

$$\mathbb{E}[\mathcal{A}(X) \cdot [\mathcal{W}]] \leq O\left(\frac{1}{n}\right).$$

Proof. First, note that the cost of *any* solution will be at most n. Hence the claim follows when we have shown that $\Pr[\mathcal{W}] \leq O\left(\frac{1}{n^2}\right)$.

As each run starts in one of the first βn cells, some run needs to be of length at least αn to cover the buffer at the end of A, which is a necessary condition for wraparound. For such a long run to occur, there must in turn exist an interval $I \subset [0, \beta n]$ of length αn such that there are at least αn values $x \in X$ with $h(x) \in I$. Denote the number of such elements by X_I.

As X_I is the sum of independent trials, each with success probability α/β, we can bound the probability of X_I exceeding αn by a Chernoff bound. Setting $\delta = \beta - 1$ and $\mu = \mathbb{E}[X_I] = n \cdot \alpha/\beta$ we have, for any fixed α, β,

$$\Pr[X_I > \alpha n] = \Pr[X_I > (1+\delta)\mu] \leq \exp\left(\frac{-\delta^2\mu}{2+\delta}\right) = \exp\left(\frac{-(\beta-1)^2\frac{\alpha}{\beta}n}{1+\beta}\right) \leq O\left(\frac{1}{n^4}\right).$$

By a union bound over all βn intervals we get $\Pr[\mathcal{W}] \leq O(\beta/n^3) \leq O(1/n^2)$. ◀

Bounding the cost of the solution. We account for three types of cost in $\mathcal{A}(X)$: If the i-th element is inserted in a cell right before the start of a run, the i-th step is said to incur a *merge* cost, denoted $mer(i)$, as it potentially merges two runs. If the i-th element is inserted in the cell following a run, the i-th step is said to incur an *extend* cost, denoted $ext(i)$, as it extends the run. If the i-th insertion does not give rise to a merge- (resp. extend-) cost, we set $mer(i) = 0$ (resp. $ext(i) = 0$). Finally we have to account for the cost between values $A[i]$ and $A[j]$ separated by one or more empty cells. This cost is not charged to a specific insertion performed by the algorithm but is instead viewed as a property of the solution, $sep(\mathcal{A}(X))$.

We bound the expected size of each type of cost in the following three claims, and together they account for the value of the solution produced by $\mathcal{A}$.

▷ Claim 32.

$$\sum_{i=1}^{n} \mathbb{E}[mer(i) \cdot [\neg\mathcal{W}]] \leq \frac{1}{\beta n}\sum_{i=1}^{n}\mathbb{E}[s(i)] + \frac{1}{\beta}.$$

Proof. Let x_i be the i-th element of X and let k be the index where x_i is placed. Then $s(i) = 1 + k - h(x_i)$, due to our assumption that no wraparound occurs. Let $y = A[k+1]$ be the value in the neighbouring cell, which was inserted before x_i. Then $mer(i) = |y - x_i|$.

As cell $A[k]$ was empty at the time of y's insertion we must have $y \in \left[\frac{k+1}{\beta n}, \frac{k+2}{\beta n}\right)$. Meanwhile, $x_i \in \left[\frac{h(x_i)}{\beta n}, \frac{h(x_i)+1}{\beta n}\right)$ and thus

$$|y - x_i| \leq \frac{k + 2 - h(x_i)}{\beta n} = \frac{s(i)+1}{\beta n}.$$

It follows that $\sum_{i=1}^{n}\mathbb{E}[mer(i)\cdot[\neg\mathcal{W}]] \leq \frac{1}{\beta n}\sum_{i=1}^{n}\mathbb{E}[s(i)] + \frac{1}{\beta}$. ◀

▷ Claim 33.

$$\sum_{i=1}^{n} \mathbb{E}[ext(i) \cdot [\neg\mathcal{W}]] \leq \frac{1}{\beta n}\sum_{i=1}^{n}\mathbb{E}[s(i)].$$

Proof. Denote by I_k the k-th run in A which spans cells $A[a_k]$ through $A[b_k]$ before the insertion of x_i, with value $y_k = A[b_k]$ in the final position. Assuming no wraparound occurs, we have $a_k \leq b_k$. In a slight abuse of notation we denote by $h(x_i) \in I_k$ the event that x_i is appended to I_k, which is equivalent to the event $h(x_i) \in [a_k, b_k + 1]$. Assuming $h(x_i) \in I_k$, we then have $s(i) = 1 + (b_k + 1) - h(x_i) = 2 + b_k - h(x_i)$. Then

$$ext(i) = \sum_{k=1} |x_i - y_k| \cdot [h(x_i) \in I_k] \leq \sum_{k=1} \frac{1}{\beta n} \left(|h(x_i) - h(y_k)| + 1\right) \cdot [h(x_i) \in I_k] \ .$$

As we are interested in $\mathbb{E}[ext(i)]$ we wish to bound $\mathbb{E}[|h(x_i) - h(y_k)| \cdot [h(x_i) \in I_k]]$ for some fixed k. We will not delve into the distribution of $h(y_k)$ but note that the distribution of $h(x_i)$ is independent of $h(y_k)$. Conditioning on $h(y_k)$, we see that the expectation is maximized when $h(y_k)$ is at one of the "extremes" $h(y_k) = a_k$ or $h(y_k) = b_k$. For ease of computation we substitute $h(y_k) = b_k + 1$ for our upper bound:

$$\begin{aligned}\mathbb{E}[|h(x_i) - h(y_k)| \cdot [h(x_i) \in I_k] \cdot [\neg\mathcal{W}]] &\leq \mathbb{E}[|(b_k + 1) - h(x_i)| \cdot [h(x_i) \in I_k]] \\ &= \mathbb{E}[(s(i) - 1) \cdot [h(x_i) \in I_k]] \ .\end{aligned}$$

Plugging into our previous equation we obtain

$$\mathbb{E}[ext(i) \cdot [\neg\mathcal{W}]] \leq \frac{1}{\beta n} \sum_{k=1} \mathbb{E}[s(i) \cdot [h(x_i) \in I_k]] \leq \frac{1}{\beta n} \cdot \mathbb{E}[s(i)] \ .$$

◀

▷ Claim 34.

$$\mathbb{E}[sep(\mathcal{A}(X)) \cdot [\neg\mathcal{W}]] \leq 2 \, .$$

Proof. Let the runs be numbered $1, \ldots, r$ according to their order in A, and say that the i-th run covers cells $A[a_i]$ through $A[b_i]$ such that $a_i \leq b_i < a_{i+1} \leq b_{i+1}$. With this notation $sep(\mathcal{A}(X)) = \sum_{i=1}^{r} |A[a_{i+1}] - A[b_i]|$.

Fix some i, let $x = A[a_{i+1}]$ and $y = A[b_i]$. First, observe that $h(x) = a_{i+1}$. Had $h(x)$ been lower, x would have been placed in a cell of smaller index (and we know that $A[a_{i+1} - 1]$ is empty). Had $h(x)$ been higher, x would have been placed in a cell of index at least a_{i+1} due to our assumption that no wraparound occurs. Hence $A[a_{i+1}] \in \left[\frac{a_{i+1}}{\beta n}, \frac{a_{i+1}+1}{\beta n}\right)$.

Similarly, we have $a_i \leq h(y) \leq b_i$ and thus $A[b_i] \in \left[\frac{a_i}{\beta n}, \frac{b_i+1}{\beta n}\right)$, as the element y would otherwise have been placed outside of the run. Thus

$$\sum_{i=1}^{r} |A[a_{i+1}] - A[b_i]| \leq \sum_{i=1}^{r} \frac{a_{i+1} - a_i + 1}{\beta n} = \frac{a_r - a_1 + r}{\beta n} \leq 2$$

as $r \leq a_r \leq \beta n$. ◀

Summing over Claims 31–34, we obtain the bound stated in Claim 29. ◀

Appendix D

A Faster Algorithm for Constrained Correlation Clustering

A Faster Algorithm for Constrained Correlation Clustering

Nick Fischer
INSAIT, Sofia University "St. Kliment Ohridski", Bulgaria

Evangelos Kipouridis
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Jonas Klausen
BARC, University of Copenhagen, Denmark

Mikkel Thorup
BARC, University of Copenhagen, Denmark

Abstract

In the Correlation Clustering problem we are given n nodes, and a preference for each pair of nodes indicating whether we prefer the two endpoints to be in the same cluster or not. The output is a clustering inducing the minimum number of violated preferences. In certain cases, however, the preference between some pairs may be too important to be violated. The constrained version of this problem specifies pairs of nodes that *must* be in the same cluster as well as pairs that *must not* be in the same cluster (hard constraints). The output clustering has to satisfy all hard constraints while minimizing the number of violated preferences.

Constrained Correlation Clustering is APX-Hard and has been approximated within a factor 3 by van Zuylen *et al.* [SODA '07]. Their algorithm is based on rounding an LP with $\Theta(n^3)$ constraints, resulting in an $\Omega(n^{3\omega})$ running time. In this work, using a more combinatorial approach, we show how to approximate this problem significantly faster at the cost of a slightly weaker approximation factor. In particular, our algorithm runs in $\widetilde{O}(n^3)$ time (notice that the input size is $\Theta(n^2)$) and approximates Constrained Correlation Clustering within a factor 16.

To achieve our result we need properties guaranteed by a particular influential algorithm for (unconstrained) Correlation Clustering, the CC-PIVOT algorithm. This algorithm chooses a *pivot* node u, creates a cluster containing u and all its preferred nodes, and recursively solves the rest of the problem. It is known that selecting pivots at random gives a 3-approximation. As a byproduct of our work, we provide a derandomization of the CC-PIVOT algorithm that still achieves the 3-approximation; furthermore, we show that there exist instances where no ordering of the pivots can give a $(3-\varepsilon)$-approximation, for any constant ε.

Finally, we introduce a node-weighted version of Correlation Clustering, which can be approximated within factor 3 using our insights on Constrained Correlation Clustering. As the general weighted version of Correlation Clustering would require a major breakthrough to approximate within a factor $o(\log n)$, Node-Weighted Correlation Clustering may be a practical alternative.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases Clustering, Constrained Correlation Clustering, Approximation

Funding *Jonas Klausen* and *Mikkel Thorup* are part of BARC, Basic Algorithms Research Copenhagen, supported by VILLUM Foundation grants 16582 and 54451. This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979).
Nick Fischer: Partially funded by the Ministry of Education and Science of Bulgaria's support for INSAIT, Sofia University "St. Kliment Ohridski" as part of the Bulgarian National Roadmap for Research Infrastructure. Parts of this work were done while the author was at Saarland University.

Acknowledgements We thank Lorenzo Beretta for his valuable suggestions on weighted sampling.

1 Introduction

Clustering is a fundamental task related to unsupervised learning, with many applications in machine learning and data mining. The goal of clustering is to partition a set of nodes into disjoint clusters, such that (ideally) all nodes within a cluster are similar, and nodes in different clusters are dissimilar. As no single definition best captures this abstract goal, a lot of different clustering objectives have been suggested.

Correlation Clustering is one of the most well studied such formulations for a multitude of reasons: Its definition is simple and natural, it does not need the number of clusters to be part of the input, and it has found success in many applications. Some few examples include automated labeling [1, 16], clustering ensembles [11], community detection [20, 39], disambiguation tasks [32], duplicate detection [6] and image segmentation [33, 43].

In Correlation Clustering we are given a graph $G = (V, E)$, and the output is a partition (clustering) $C = \{C_1, \ldots, C_k\}$ of the vertex set V. We refer to the sets C_i of C as *clusters*. The goal is to minimize the number of edges between different clusters plus the number of non-edges inside of clusters. More formally, the goal is to minimize $|E \triangle E_C|$, the cardinality of the symmetric difference between E and E_C, where we define $E_C = \bigcup_{i=1}^{k} \binom{C_i}{2}$. In other words, the goal is to transform the input graph into a collection of cliques with the minimal number of edge insertions and deletions. An alternative description used by some authors is that we are given a *complete* graph where the edges are labeled either "+" (corresponding to the edges in our graph G) or "−" (corresponding to the non-edges in our graph G).

The problem is typically motivated as follows: Suppose that the input graph models the relationships between different entities which shall be grouped. An edge describes that we prefer its two endpoints to be clustered together, whereas a non-edge describes that we prefer them to be separated. In this formulation the cost of a correlation clustering is the number of violated preferences.

1.1 Previous Results

Correlation Clustering was initially introduced by Bansal, Blum, and Chawla [8], who proved that it is NP-Hard, and provided a deterministic constant-factor approximation, the constant being larger than 15,000. Subsequent improvements were based on rounding the natural LP: Charikar, Guruswami and Wirt gave a deterministic 4-approximation [18], Ailon, Charikar and Newman gave a randomized 2.5-approximation and proved that the problem is APX-Hard [3], while a deterministic 2.06-approximation was given by Chawla, Makarychev, Schramm and Yaroslavtsev [19]. The last result is near-optimal among algorithms rounding the natural LP, as its integrality gap is at least 2. In a breakthrough result by Cohen-Addad, Lee and Newman [25] a $(1.994 + \epsilon)$-approximation using the Sherali-Adams relaxation broke the 2 barrier. It was later improved to $1.73 + \epsilon$ [24] by Cohen-Addad, Lee, Li and Newman, and even to 1.437 by Cao *et al.* [14]. There is also a combinatorial 1.847-approximation (Cohen-Addad *et al.* [26]).

Given the importance of Correlation Clustering, research does not only focus on improving its approximation factor. Another important goal is efficient running times without big sacrifices on the approximation factor. As the natural LP has $\Theta(n^3)$ constraints, using a state-of-the-art LP solver requires time $\Omega(n^{3\omega}) = \Omega(n^{7.113})$. In order to achieve efficient running times, an algorithm thus has to avoid solving the LP using an all-purpose LP-solver, or the even more expensive Sherali-Adams relaxation; such algorithms are usually called

combinatorial algorithms[1]. Examples of such a direction can be seen in [3] where, along with their LP-based 2.5-approximation, the authors also design a combinatorial 3-approximation (the CC-PIVOT algorithm); despite its worse approximation, it enjoys the benefit of being faster. Similarly, much later than the 2.06-approximation [19], Veldt devised a faster combinatorial 6-approximation and a 4-approximation solving a less expensive LP [38].

Another important direction is the design of *deterministic* algorithms. For example, [3] posed as an open question the derandomization of CC-PIVOT. The question was (partially) answered affirmatively by [37]. Deterministic algorithms were also explicitly pursued in [38], and are a significant part of the technical contribution of [19].

Correlation Clustering has also been studied in different settings such as parameterized algorithms [29], sublinear and streaming algorithms [7, 13, 9, 10, 13, 17], massively parallel computation (MPC) algorithms [23, 15], and differentially private algorithms [12].

PIVOT. The CC-PIVOT algorithm [3] is a very influential algorithm for Correlation Clustering. It provides a 3-approximation and is arguably the simplest constant factor approximation algorithm for Correlation Clustering. It simply selects a node uniformly at random, and creates a cluster $\mathcal{C}$ with this node and its neighbors in the (remaining) input graph. It then removes $\mathcal{C}$'s nodes and recurses on the remaining graph. Due to its simplicity, CC-PIVOT has inspired several other algorithms, such as algorithms for Correlation Clustering in the streaming model [7, 13, 9, 10, 13, 17] and algorithms for the more general Fitting Utrametrics problem [2, 22].

One can define a meta-algorithm based on the above, where we do not necessarily pick the pivots uniformly at random. Throughout this paper, we use the term PIVOT algorithm to refer to an instantiation of the (Meta-)Algorithm 1[2]. Obviously CC-PIVOT is an instantiation of PIVOT, where the pivots are selected uniformly at random.

Algorithm 1 The PIVOT meta-algorithm. CC-PIVOT is an instantiation of PIVOT where pivots are selected uniformly at random.

```
   procedure Pivot(G = (V, E))
1  |  C ← ∅
2  |  while V ≠ ∅ do
3  |  |  Pick a pivot node u
   |  |  /* an instantiation of Pivot() only needs to specify how the
   |  |     pivot is selected in each iteration                      */
4  |  |  Add a cluster containing u and all its neighbors to C
5  |  |  Remove u, its neighbors and all their incident edges from G
6  |  return C
```

The paper that introduced CC-PIVOT [3] posed as an open question the derandomization of the algorithm. The question was partially answered in the affirmative by [37]. Unfortunately

[1] On a more informal note, combinatorial algorithms are often not only faster, but also provide deeper insights on a problem, compared to LP-based ones.

[2] This is not to be confused with the more general pivoting paradigm for Correlation Clustering algorithms. In that design paradigm, the cluster we create for each pivot is not necessarily the full set of remaining nodes with which the pivot prefers to be clustered, but can be decided in any other way (e.g. randomly, based on a probability distribution related to an LP or more general hierarchies such as the Sherali-Adams hierarchy).

there are two drawbacks with this algorithm. First, it requires solving the natural LP, which makes its running time equal to the pre-existing (better) 2.5-approximation. Second, this algorithm does not only derandomize the order in which pivots are selected, but also decides the cluster of each pivot based on an auxiliary graph (dictated by the LP) rather than based on the original graph. Therefore it is not an instantiation of PIVOT.

Weighted Correlation Clustering. In the weighted version of Correlation Clustering, we are also given a weight for each preference. The final cost is then the sum of weights of the violated preferences. An $O(\log n)$-approximation for weighted Correlation Clustering is known by Demaine, Emanuel, Fiat and Immorlica [27]. In the same paper they show that the problem is equivalent to the Multicut problem, meaning that an $o(\log n)$-approximation would require a major breakthrough. As efficiently approximating the general weighted version seems out of reach, research has focused on special cases for which constant-factor approximations are possible [34, 35].

Constrained Correlation Clustering. Constrained Correlation Clustering is an interesting variant of Correlation Clustering capturing the idea of critical pairs of nodes. To address these situations, Constrained Correlation Clustering introduces hard constraints in addition to the pairwise preferences. A clustering is valid if it satisfies all hard constraints, and the goal is to find a valid clustering of minimal cost. We can phrase Constrained Correlation Clustering as a weighted instance of Correlation Clustering: Simply give infinite weight to pairs associated with a hard constraint and weight 1 to all other pairs.

To the best of our knowledge, the only known solution to Constrained Correlation Clustering is given in the work of van Zuylen and Williamson who designed a deterministic 3-approximation [37]. The running time of this algorithm is $O(n^{3\omega})$, where $\omega < 2.3719$ is the matrix-multiplication exponent. Using the current best bound for ω, this is $\Omega(n^{7.113})$.

1.2 Our Contribution

Our main result is the following theorem. It improves the $\Omega(n^{7.113})$ running time of the state-of-the-art algorithm for Constrained Correlation Clustering while still providing a constant (but larger than 3) approximation factor[3].

▶ **Theorem 1** (Constrained Correlation Clustering). *There is a deterministic algorithm for Constrained Correlation Clustering computing a* 16*-approximation in time* $\widetilde{O}(n^3)$.

We first show how to obtain this result, but with a randomized algorithm that holds with high probability, instead of a deterministic one. In order to do so, we perform a (deterministic) preprocessing step and then use the CC-PIVOT algorithm. Of course CC-PIVOT alone, without the preprocessing step, would not output a clustering respecting the hard constraints. Its properties however (and more generally the properties of PIVOT algorithms) are crucial; we are not aware of any other algorithm that we could use instead and still satisfy all the hard constraints of Constrained Correlation Clustering after our preprocessing step.

To obtain our deterministic algorithm we derandomize the CC-PIVOT algorithm.

▶ **Theorem 2** (Deterministic PIVOT). *There are the following deterministic PIVOT algorithms for Correlation Clustering:*

[3] We write $\widetilde{O}(T)$ to suppress polylogarithmic factors, i.e., $\widetilde{O}(T) = T(\log T)^{O(1)}$.

- *A combinatorial* $(3+\epsilon)$*-approximation, for any constant* $\epsilon > 0$*, in time* $\widetilde{O}(n^3)$*.*
- *A non-combinatorial* 3*-approximation in time* $\widetilde{O}(n^5)$*.*

We note that the final approximation of our algorithm for Constrained Correlation Clustering depends on the approximation of the applied PIVOT algorithm. If it was possible to select the order of the pivots in a way that guarantees a better approximation, this would immediately improve the approximation of our Constrained Correlation Clustering algorithm. For this reason, we study lower bounds for PIVOT; currently, we know of instances for which selecting the pivots at random doesn't give a better-than-3-approximation in expectation [3]; however, for these particular instances there *does* exist a way to choose the pivots that gives better approximations. Ideally, we want a lower bound applying for any order of the pivots (such as the lower bound for the generalized PIVOT solving the Ultrametric Violation Distance problem in [22]). We show that our algorithm is optimal, as there exist instances where no ordering of the pivots will yield a better-than-3-approximation.

▶ **Theorem 3** (PIVOT Lower Bound)**.** *There is no constant* $\epsilon > 0$ *for which there exists a PIVOT algorithm for Correlation Clustering with approximation factor* $3 - \epsilon$*.*

We also introduce the Node-Weighted Correlation Clustering problem, which is related to (but incomparable, due to their asymmetric assignment of weights) a family of problems introduced in [39]. As weighted Correlation Clustering is equivalent to Multicut, improving over the current $\Theta(\log n)$-approximation seems out of reach. The advantage of our alternative type of weighted Correlation Clustering is that it is natural and approximable within a constant factor.

In Node-Weighted Correlation Clustering we assign weights to the nodes, rather than to pairs of nodes. Violating the preference between nodes u, v with weights ω_u and ω_v incurs cost $\omega_u \cdot \omega_v$. We provide three algorithms computing (almost-)3-approximations for Node-Weighted Correlation Clustering:

▶ **Theorem 4** (Node-Weighted Correlation Clustering, Deterministic)**.** *There are the following deterministic algorithms for Node-Weighted Correlation Clustering:*

- *A combinatorial* $(3+\epsilon)$*-approximation, for any constant* $\epsilon > 0$*, in time* $\widetilde{O}(n^3)$*.*
- *A non-combinatorial* 3*-approximation in time* $O(n^{7.116})$*.*

▶ **Theorem 5** (Node-Weighted Correlation Clustering, Randomized)**.** *There is a randomized combinatorial algorithm for Node-Weighted Correlation Clustering computing an expected* 3*-approximation in time* $O(n+m)$ *with high probability* $1 - 1/\operatorname{poly}(n)$*.*

1.3 Overview of Our Techniques

Constrained Correlation Clustering. We obtain a faster algorithm for Constrained Correlation Clustering by

1. modifying the input graph using a subroutine aware of the hard-constraints, and
2. applying a PIVOT algorithm on this modified graph.

In fact, no matter what PIVOT algorithm is used, the output clustering respects all hard constraints when the algorithm is applied on the modified graph.

To motivate this two-step procedure, we note that inputs exist where *no* PIVOT algorithm, if applied to the unmodified graph, would respect the hard constraints. One such example is the cycle on four vertices, with two vertex-disjoint edges made into hard constraints.

The solution of [37] is similar to ours, as it also modifies the graph before applying a Correlation Clustering algorithm. However, both their initial modification and the following Correlation Clustering algorithm require solving the standard LP, which is expensive

($\Omega(n^{7.113})$ time). In our case both steps are implemented with deterministic and combinatorial algorithms which brings the running time down to $\widetilde{O}(n^3)$.

For the first step, our algorithm carefully modifies the input graph so that on one hand the optimal cost is not significantly changed, and on the other hand any PIVOT algorithm on the transformed graph returns a clustering that respects all hard constraints. For the second step, we use a deterministic combinatorial PIVOT algorithm.

Concerning the effect of modifying the graph, roughly speaking we get that the final approximation factor is $(2+\sqrt{5})\cdot\alpha+3$, where α is the approximation factor of the PIVOT algorithm we use. Plugging in $\alpha = 3+\epsilon$ from Theorem 2 we get the first combinatorial constant-factor approximation for Constrained Correlation Clustering in $\widetilde{O}(n^3)$ time.

Node-Weighted Correlation Clustering. We generalize the deterministic combinatorial techniques from before to the Node-Weighted Correlation Clustering problem. In addition, we also provide a very efficient randomized algorithm for the problem. It relies on a weighted random sampling technique.

One way to view the algorithm is to reduce Node-Weighted Correlation Clustering to an instance of Constrained Correlation Clustering, with the caveat that the new instance's size depends on the weights (and can thus even be exponential). Each node u is replaced by a set of nodes of size related to u's weight and these nodes have constraints forcing them to be in the same cluster.

We show that we can simulate a simple randomized PIVOT algorithm on that instance, where instead of sampling uniformly at random, we sample with probabilities proportional to the weights. Assuming polynomial weights, we can achieve this in linear time. To do so, we design an efficient data structure supporting such sampling and removal of elements.

It is easy to implement such a data structure using any balanced binary search tree, but the time for constructing it and applying all operations would be $O(n\log n)$. Using a non-trivial combination of the Alias Method [41, 40] and Rejection Sampling, we achieve a linear bound.

Due to space constraints the presentation of our algorithms for Node-Weighted Correlation Clustering is deferred to Appendix D.

Deterministic PIVOT algorithms. Our algorithms are based on a simple framework by van Zuylen and Williamson [37]. In this framework we assign a nonnegative "charge" to each pair of nodes. Using these charges, a PIVOT algorithm decides which pivot to choose next. The approximation factor depends on the total charge (as compared with the cost of an optimal clustering), and the minimum charge assigned to any bad triplet (an induced subgraph $K_{1,2}$).

The reason why these bad triplets play an important role is that for any bad triplet, any clustering needs to pay at least 1. To see this, let uvw be a bad triplet with uv being the only missing edge. For a clustering to pay 0, it must be the case that both uw and vw are together. However, this would imply that uv are also together although they prefer not to.

Our combinatorial $(3+\epsilon)$-approximation uses the multiplicative weights update method, which can be intuitively described as follows: We start with a tiny charge on all pairs. Then we repeatedly find a bad triplet uvw with currently minimal charge (more precisely: for which the sum of the charges of uv, vw, wu is minimal), and scale the involved charges by $1+\epsilon$. One can prove that this eventually results in an almost-optimal distribution of charges, up to rescaling.

For this purpose it suffices to show that the total assigned charge is not large compared to the cost of the optimal correlation clustering. We do so by observing that our algorithm $(1+\epsilon)$-approximates the covering LP of Figure 1, which we refer to as the *charging LP*.

Our faster deterministic non-combinatorial algorithm solves the charging LP using an LP solver tailored to covering LPs [4, 42]. An improved solver for covering LPs would directly improve the running time of this algorithm.

Figure 1 The primal and dual LP relaxations for Correlation Clustering, which we refer to as the *charging LP*. $T(G)$ is the set of all bad triplets in G.

$$\begin{aligned} \min \quad & \sum_{uv \in \binom{V}{2}} x_{uv} \\ \text{s.t.} \quad & x_{uv} + x_{vw} + x_{wu} \geq 1 \quad \forall uvw \in T(G), \\ & x_{uv} \geq 0 \quad \forall uv \in \binom{V}{2} \end{aligned}$$

$$\begin{aligned} \max \quad & \sum_{uvw \in T(G)} y_{uvw} \\ \text{s.t.} \quad & \sum_{w : uvw \in T(G)} y_{uvw} \leq 1 \quad \forall uv \in \binom{V}{2}, \\ & y_{uvw} \geq 0 \quad \forall uvw \in T(G) \end{aligned}$$

Lower Bound. Our lower bound is obtained by taking a complete graph K_n for some even number of vertices n, and removing a perfect matching. Each vertex in the resulting graph is adjacent to all but one other vertex and so *any* PIVOT algorithm will partition the vertices into a large cluster of $n-1$ vertices and a singleton cluster. A non PIVOT algorithm, however, is free to create just a single cluster of size n, at much lower cost. The ratio between these solutions tends to 3 with increasing n.

We note that in [3] the authors proved that CC-PIVOT's analysis is tight. That is, its expected approximation factor is not better than 3. However, their lower bound construction (a complete graph K_n minus one edge) only works for CC-PIVOT, not for PIVOT algorithms in general.

1.4 Open Problems

We finally raise some open questions.

1. Can we improve the approximation factor of Constrained Correlation Clustering from 16 to 3 while keeping the running time at $\widetilde{O}(n^3)$?
2. We measure the performance of a PIVOT algorithm by comparing it to the best correlation clustering obtained by *any* algorithm. But as Theorem 3 proves, there is no PIVOT algorithm with an approximation factor better than 3. If we instead compare the output to the best correlation clustering obtained by a *PIVOT algorithm*, can we get better guarantees (perhaps even an exact algorithm in polynomial time)?
3. In the Node-Weighted Correlation Clustering problem, we studied the natural objective of minimizing the total cost $\omega_v \cdot \omega_u$ of all violated preferences uv. Are there specific

applications of this problem? Can we achieve similar for other cost functions such as $\omega_v + \omega_u$?

2 Preliminaries

We denote the set $\{1, \ldots, n\}$ by $[n]$. We denote all subsets of size k of a set A by $\binom{A}{k}$. The symmetric difference between two sets A, B is denoted by $A \triangle B$. We write $\text{poly}(n) = n^{O(1)}$ and $\widetilde{O}(n) = n(\log n)^{O(1)}$.

In this paper all graphs $G = (V, E)$ are undirected and unweighted. We typically set $n = |V|$ and $m = |E|$. For two disjoint subsets $U_1, U_2 \subseteq V$, we denote the set of edges with one endpoint in U_1 and the other in U_2 by $E(U_1, U_2)$. The subgraph of G induced by vertex-set U_1 is denoted by $G[U_1]$. For vertices u, v, w we often abbreviate the (unordered) set $\{u, v\}$ by uv and similarly write uvw for $\{u, v, w\}$. We say that uvw is a *bad triplet* in G if the induced subgraph $G[uvw]$ contains exactly two edges (i.e., is isomorphic to $K_{1,2}$). Let $T(G)$ denote the set of bad triplets in G. We say that the edge set E_C of a clustering $C = \{C_1, \ldots, C_k\}$ of V is the set of pairs with both endpoints in the same set in C. More formally, $E_C = \bigcup_{i=1}^{k} \binom{C_i}{2}$.

We now formally define the problems of interest.

▶ **Definition 6** (Correlation Clustering). *Given a graph $G = (V, E)$, output a clustering $C = \{C_1, \ldots, C_k\}$ of V with edge set E_C minimizing $|E \triangle E_C|$.*

An algorithm for Correlation Clustering is said to be a PIVOT algorithm if it is an instantiation of Algorithm 1 (Page 2). That is, an algorithm which, based on some criterion, picks an unclustered node u (the *pivot*), creates a cluster containing u and its unclustered neighbors in (V, E), and repeats the process until all nodes are clustered. In particular, the algorithm may not modify the graph in other ways before choosing a pivot.

The constrained version of Correlation Clustering is defined as follows.

▶ **Definition 7** (Constrained Correlation Clustering). *Given a graph $G = (V, E)$, a set of friendly pairs $F \subseteq \binom{V}{2}$ and a set of hostile pairs $H \subseteq \binom{V}{2}$, compute a clustering $C = \{C_1, \ldots, C_k\}$ of V with edge set E_C such that no pair $uv \in F$ has u, v in different clusters and no pair $uv \in H$ has u, v in the same cluster. The clustering C shall minimize $|E \triangle E_C|$.*

We also introduce Node-Weighted Correlation Clustering, a new related problem that may be of independent interest.

▶ **Definition 8** (Node-Weighted Correlation Clustering). *Given a graph $G = (V, E)$ and positive weights $\{\omega_u\}_{u \in V}$ on the nodes, compute a clustering $C = \{C_1, \ldots, C_k\}$ of V with edge set E_C minimizing*

$$\sum_{uv \in E \triangle E_C} \omega_u \cdot \omega_v \,.$$

For simplicity, we assume that the weights are bounded by $\text{poly}(n)$, and thereby fit into a constant number of word RAM cells of size $w = \Theta(\log n)$. We remark that our randomized algorithm would be a polynomial (but not linear) time one if we allowed the weights to be of exponential size.

The Node-Weighted Correlation Clustering problem clearly generalizes Correlation Clustering since we pay $w(u) \cdot w(v)$ (instead of 1) for each pair uv violating a preference.

3 Combinatorial Algorithms for Constrained Correlation Clustering

Let us fix the following notation: A connected component in (V, F) is a *supernode*. The set of supernodes partitions V and is denoted by SN. Given a node u, we let $s(u)$ be the unique supernode containing u. Two supernodes U, W are *hostile* if there exists a hostile pair uw with $u \in U, w \in W$. Two supernodes U, W are *connected* if $|E(U, W)| \geq 1$. Two supernodes U, W are β-*connected* if $|E(U, W)| \geq \beta \cdot |U| \cdot |W|$.

The first step of our combinatorial approach is to transform the graph G into a more manageable form G', see procedure Transform of Algorithm 2. The high-level idea is that in G':

1. If uv is a friendly pair, then u and v are connected and have the same neighborhood.
2. If uv is a hostile pair, then u and v are not connected and have no common neighbor.
3. An $O(1)$-approximation of the G' instance is also an $O(1)$-approximation of the G instance.

As was already noticed in [37], Properties 1 and 2 imply that a PIVOT algorithm on G' gives a clustering satisfying the hard constraints. Along with Property 3 and our deterministic combinatorial PIVOT algorithm for Correlation Clustering in Theorem 2, we prove Theorem 1. Properties 1 and 2 (related to correctness) and the running time ($\widetilde{O}(n^3)$) of our algorithm are relatively straightforward to prove. Due to space constraints, their proofs can be found in Appendix C. In this section we instead focus on the most technically challenging part, the approximation guarantee.

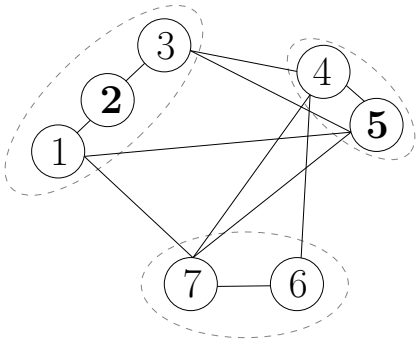
Our algorithm works as follows (see also Figure 2): If some supernode is hostile to itself, then it outputs that no clustering satisfies the hard constraints. Else, starting from the edge set E, it adds all edges within each supernode. Then it drops all edges between hostile supernodes. Subsequently, it repeatedly detects hostile supernodes that are connected with the same supernode, and drops one edge from each such connection. Finally, for each β-connected pair of supernodes, it connects all their nodes if $\beta > \frac{3-\sqrt{5}}{2}$, and disconnects them otherwise[4].

From a high-level view, the first two modifications are directly related to the hard constraints: If u_1, u_2 are friendly and u_2, u_3 are friendly, then any valid clustering has u_1, u_3 in the same cluster, even if a preference discourages it. Similarly, if u_1, u_2 are friendly, u_3, u_4 are friendly, but u_1, u_3 are hostile, then any valid clustering has u_2, u_4 in different clusters, even if a preference discourages it. Our first two modifications simply make the preferences consistent with the hard constraints.

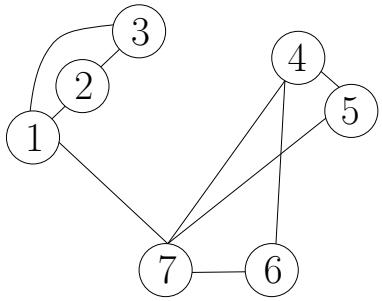
The third modification guarantees that hostile supernodes share no common neighbor. A PIVOT algorithm will thus never put their nodes in the same cluster, as the hostility constraints require. Concerning the cost, notice that if hostile supernodes U_1, U_2 are connected with supernode U_3, then no valid clustering can put all three of them in the same cluster. Therefore we always need to pay either for the connections between U_1 and U_3, or for the connections between U_2 and U_3.

Finally, after the rounding step, for each pair of supernodes U_1, U_2, the edge set $E(U_1, U_2)$ is either empty or the full set of size $|U_1| \cdot |U_2|$. This ensures that a PIVOT algorithm always puts all nodes of a supernode in the same cluster, thus also obeying the friendliness constraints. Concerning the cost of the rounded instance, a case analysis shows that it is always within a constant factor of the cost of the instance before rounding.

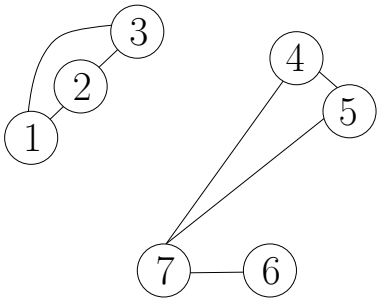
[4] The constant $\frac{3-\sqrt{5}}{2}$ optimizes the approximation factor. The natural choice of 0.5 would still give a constant approximation factor, albeit slightly worse.



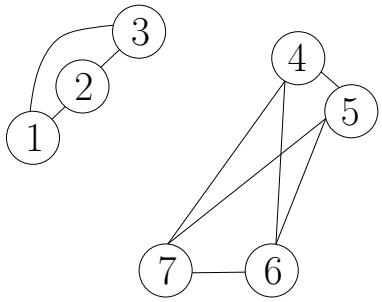
(a) The original graph. The set of friendly pairs is $F = \{\{1,2\},\{2,3\},\{4,5\},\{6,7\}\}$, and the only hostile pair in H is $\{2,5\}$.



(b) Line 3 introduces edge $\{1,3\}$, and Line 4 disconnects the supernodes containing 2 and 5.



(c) Line 6 removes the pair of edges $\{1,7\}$ and $\{4,6\}$ because $1,4$ are in hostile supernodes while $6,7$ are in the same supernode.



(d) Line 9 introduces all edges connecting supernodes $\{4,5\}$ and $\{6,7\}$ because there were enough edges between them already.

Figure 2 Illustrates an application of TRANSFORM(G,F,H) (Algorithm 2). In the transformed graph, for any two supernodes U_1, U_2, either all pairs with an endpoint in U_1 and an endpoint in U_2 share an edge, or none of them do. Furthermore, all pairs within a supernode are connected and no hostile supernodes are connected.

Formally, let E' be the edge set of the transformed graph G', let E_3 be the edge set at Line 8 of Algorithm 2 (exactly before the rounding step), OPT be the edge set of an optimal clustering for E satisfying the hard constraints described by F and H, OPT$'$ be the edge set of an optimal clustering for the preferences defined by E', and E_C be the edge set of the clustering returned by our algorithm. Finally, let α be the approximation factor of the PIVOT algorithm used.

▶ **Lemma 9.** *Given an instance (V, E, F, H) of Constrained Correlation Clustering, if two nodes u_1, u_2 are in the same supernode, then they must be in the same cluster.*

Proof. The proof follows by "in the same cluster" being a transitive property.

More formally, u_1, u_2 are in the same connected component in (V, F), as $s(u_1) = s(u_2)$. Thus, there exists a path from u_1 to u_2. We claim that all nodes in a path must be in the same cluster. This is trivial if the path is of length 0 ($u_1 = u_2$) or of length 1 ($u_1u_2 \in F$). Else, the path is $u_1, w_1, \ldots, w_k, u_2$ for some $k \geq 1$. We inductively have that all of $w_1, \ldots, w_k, u_2$ must be in the same cluster, and u_1 must be in the same cluster with w_1 because $u_1w_1 \in F$. Therefore, all nodes in the path must be in the same cluster with w_1. ◀

We now show that it is enough to bound the symmetric difference between E and E'.

Algorithm 2 The procedure ConstrainedCluster is given a graph $G = (V, E)$ describing the preferences, a set of friendly pairs F and a set of hostile pairs H. It creates a new graph G' using the procedure Transform and uses any PIVOT algorithm on G' to return a clustering.

procedure Transform$(G = (V, E), F, H)$

Compute the connected components of (V, F)

`// Impossible iff some pair must both be and not be in the same cluster.`

if $\exists U \in SN$ *hostile to itself* **then return** $G' = (\emptyset, \emptyset)$

`// Connect nodes in the same supernode.`

$E_1 \leftarrow E \cup \{ uv \in \binom{V}{2} \mid s(u) = s(v) \}$

`// Disconnect pairs in hostile supernodes.`

$E_2 \leftarrow E_1 \setminus \{ uv \in \binom{V}{2} \mid s(u)$ and $s(v)$ are hostile $\}$

`// While hostile supernodes` U_1, U_2 `are both connected with super-`
`// node` U_3`, drop an edge between` U_1, U_3 `and an edge between` U_2, U_3

$E_3 \leftarrow E_2$

while $\exists U_1, U_2, U_3 \in \binom{SN}{3}$ *such that* U_1, U_2 *are hostile and* $\exists u_1 \in U_1, u_2 \in U_2, u_3 \in U_3, u_3' \in U_3$ *such that* $u_1u_3 \in E_3, u_2u_3' \in E_3$ **do**

  $E_3 \leftarrow E_3 \setminus \{ u_1u_3, u_2u_3' \}$

`// Round connections between pairs of supernodes`

$E_4 \leftarrow E_3$

foreach $\{U_1, U_2\} \in \binom{SN}{2}$ **do**

  $E_{U_1,U_2} \leftarrow \{u_1u_2 \mid u_1 \in U_1, u_2 \in U_2\}$

  if $|E_{U_1,U_2} \cap E_4| > \frac{3-\sqrt{5}}{2}|U_1| \cdot |U_2|$ **then** $E_4 \leftarrow E_4 \cup E_{U_1,U_2}$

  else $E_4 \leftarrow E_4 \setminus E_{U_1,U_2}$

return $G' = (V, E_4)$

procedure ConstrainedCluster$(G = (V, E), F, H)$

$G' \leftarrow$ Transform$(G=(V,E),\ F,\ H)$

if $G' = (\emptyset, \emptyset)$ **then return** "Impossible"

return PIVOT(G')

▶ **Lemma 10.** *The cost of our clustering C is $|E \triangle E_C| \leq (\alpha + 1)|E \triangle E'| + \alpha|E \triangle \mathrm{OPT}|$.*

Proof. The symmetric difference of sets satisfies the triangle inequality; we therefore have

$$|E \triangle E_C| \leq |E \triangle E'| + |E' \triangle E_C|.$$

C is an α-approximation for $G' = (V, E')$ and thus $|E' \triangle E_C| \leq \alpha|E' \triangle \mathrm{OPT}'| \leq \alpha|E' \triangle \mathrm{OPT}|$. Therefore:

$$|E \triangle E_C| \leq |E \triangle E'| + \alpha|E' \triangle \mathrm{OPT}| \leq |E \triangle E'| + \alpha|E' \triangle E| + \alpha|E \triangle \mathrm{OPT}|.$$

with the second inequality following by applying the triangle inequality again. ◀

In order to upper bound $|E \triangle E'|$ by the cost of the optimal clustering $|E \triangle \mathrm{OPT}|$, we first need to lower bound the cost of the optimal clustering.

▶ **Lemma 11.** *Let S be the set of all pairs of distinct supernodes U, W that are in the same cluster in* OPT. *Then* $|E \triangle \mathrm{OPT}| \geq \sum_{\{U,W\}\in S} |E(U,W) \triangle E_3(U,W)|$.

Proof. The high-level idea is that when a node is connected to two hostile nodes, then any valid clustering needs to pay for at least one of these edges. Extending this fact to supernodes, we construct an edge set of size $\sum_{\{U,W\}\in S} |E(U,W) \triangle E_3(U,W)|$ such that the optimal clustering needs to pay for each edge in this set.

First, for any $\{U,W\} \in S$ it holds that $E(U,W) \triangle E_3(U,W) = E(U,W) \setminus E_3(U,W)$ because Line 3 (Algorithm 2) does not modify edges between pairs of distinct supernodes, and Lines 4 and 6 only remove edges.

Each edge of $E(U,W) \setminus E_3(U,W)$ is the result of applying Line 6, seeing as Line 4 only removes edges from hostile pairs of supernodes. Thus each edge $uw \in E(U,W) \setminus E_3(U,W)$ can be paired up with a unique edge $xy \in E$ which is removed together with uw. Without loss of generality it holds that $x \in U, y \in Z$ for some supernode Z different from U and W. Due to the way Line 6 chooses edges it must be the case that Z and W are hostile, hence $xy \in E \triangle \mathrm{OPT}$.

Summing over all pairs of clustered supernodes gives the result stated in the lemma. ◀

We are now ready to bound $|E \triangle E'|$.

▶ **Lemma 12.** $|E \triangle E'| \leq (1+\sqrt{5})|E \triangle \mathrm{OPT}|$

Proof. To prove this, we first charge each pair of nodes in a way such that the total charge is at most $2|E \triangle \mathrm{OPT}|$. Then we partition the pairs of nodes into 5 different sets, and show that the size of the intersection between $E \triangle E'$ and each of the 5 sets is at most $\frac{1+\sqrt{5}}{2}$ times the total charge given to the pairs in the given set.

The first three sets contain the pairs across non-hostile supernodes; out of them the first one is the most technically challenging, requiring a combination of Lemma 11 (related to Line 6 of Algorithm 2) and a direct analysis on $E \triangle \mathrm{OPT}$, as neither of them would suffice on their own. The analysis of the second and third sets relate to the rounding in Line 9. The fourth set contains pairs across hostile supernodes, while the fifth set contains pairs within supernodes. Their analysis is directly based on the hard constraints.

Let us define our charging scheme: first, each pair of nodes is charged if the optimal clustering pays for it, i.e. if this pair is in $E \triangle \mathrm{OPT}$. We further put a charge on the pairs $uw \in E \triangle E_3$ which connect supernodes that are clustered together in OPT. Notice that the number of such edges is a lower bound on $|E \triangle \mathrm{OPT}|$ by Lemma 11. Therefore the total charge over all pairs of nodes is at most $2|E \triangle \mathrm{OPT}|$ and no pair is charged twice.

Case 1: Consider two distinct supernodes U, W that are not hostile, which have more than $\frac{3-\sqrt{5}}{2}|U|\cdot|W|$ edges between them in E, and have at most $\frac{3-\sqrt{5}}{2}|U|\cdot|W|$ edges in E_3. Then the rounding of Line 9 removes all edges between them. Therefore $|E(U,W) \triangle E'(U,W)| = |E(U,W)| \leq |U| \cdot |W|$. If OPT separates U and W, then the pairs are charged $|E(U,W)|$; else they are charged $|U| \cdot |W| - |E(U,W)|$ due to the part of the charging scheme related to $E \triangle \mathrm{OPT}$. In the latter case, they are also charged $|E(U,W)| - |E_3(U,W)|$ due to the part of the charging scheme related to Lemma 11. Therefore they are charged at least

$$\begin{aligned} |U| \cdot |W| - |E(U,W)| + |E(U,W)| - |E_3(U,W)| &= |U| \cdot |W| - |E_3(U,W)| \\ &\geq |U| \cdot |W| - \tfrac{3-\sqrt{5}}{2}|U| \cdot |W|. \end{aligned}$$

Thus, in the worst case, these pairs contribute

$$\max\left\{\frac{|E(U,W)|}{|E(U,W)|}, \frac{|E(U,W)|}{|U| \cdot |W| - \frac{3-\sqrt{5}}{2}|U| \cdot |W|}\right\} \leq \frac{1}{1 - \frac{3-\sqrt{5}}{2}} = \frac{1+\sqrt{5}}{2}$$

times more in $|E \triangle E'|$ compared to their charge.

Case 2: Consider two distinct supernodes U, W that are not hostile, which have more than $\frac{3-\sqrt{5}}{2}|U| \cdot |W|$ edges between them in E, and more than $\frac{3-\sqrt{5}}{2}|U| \cdot |W|$ edges in E_3. Then the rounding of Line 9 will include all $|U| \cdot |W|$ edges between them. Thus we have $|E(U,W) \triangle E'(U,W)| = |U| \cdot |W| - |E(U,W)| < (1 - \frac{3-\sqrt{5}}{2})|U| \cdot |W|$. If OPT separates U and W it pays for $|E(U,W)| > \frac{3-\sqrt{5}}{2}|U| \cdot |W|$ pairs. Otherwise it pays $|U| \cdot |W| - |E(U,W)|$. Thus, in the worst case, these pairs contribute $\frac{1-\frac{3-\sqrt{5}}{2}}{\frac{3-\sqrt{5}}{2}} = \frac{1+\sqrt{5}}{2}$ times more in $|E \triangle E'|$ compared to their charge.

Case 3: If two distinct supernodes U, W are not hostile and have at most $\frac{3-\sqrt{5}}{2}|U| \cdot |W|$ edges between them in E, then they also have at most that many edges in E_3 as we only remove edges between such supernodes. There are thus no edges between them in E', meaning that $|E(U,W) \triangle E'(U,W)| = |E(U,W)| \leq \frac{3-\sqrt{5}}{2}|U| \cdot |W|$. If OPT separates U, W it pays for $|E(U,W)|$ pairs related to the connection between U, W; else it pays for $|U| \cdot |W| - |E(U,W)| \geq (1 - \frac{3-\sqrt{5}}{2})|U| \cdot |W| > \frac{3-\sqrt{5}}{2}|U| \cdot |W|$. Thus these pairs' contribution in $|E \triangle E'|$ is at most as much as their charge.

Case 4: Pairs uv with $s(u) \neq s(v)$ and $s(u)$ hostile with $s(v)$ are not present in E'. That is because by Line 4 no pair of hostile supernodes is connected; then Line 6 only removes edges, and Line 9 does not add any edge between $s(u)$ and $s(v)$ as they had $0 \leq \frac{3-\sqrt{5}}{2}|s(u)| \cdot |s(v)|$ edges between them. The edge uv is also not present in OPT as $s(u)$ and $s(v)$ are not in the same cluster because they are hostile. These pairs' contribution in $|E \triangle E'|$ is exactly equal to their charge.

Case 5: Pairs uv with $s(u) = s(v)$ are present in E' by Line 3 and the fact that all subsequent steps only modify edges whose endpoints are in different supernodes. The pair uv is also present in OPT, by Lemma 9. Therefore these pairs' contribution in $|E \triangle E'|$ is exactly equal to their charge.

In the worst case, the pairs of each of the five sets contribute at most $\frac{1+\sqrt{5}}{2}$ times more in $|E \triangle E'|$ compared to their charge, which proves our lemma. ◀

We are now ready to prove the main theorem.

Proof of Theorem 1. In Theorem 2 we established that there is a deterministic combinatorial PIVOT algorithm computing a Correlation Clustering with approximation factor $\alpha = 3 + \epsilon$ in time $\widetilde{O}(n^3)$, for any constant $\epsilon > 0$. Using this algorithm in Algorithm 2 gives a valid clustering. By Lemmas 10 and 12, its approximation factor is bounded by $(\alpha+1) \cdot (1+\sqrt{5}) + \alpha$. This is less than 16 for $\epsilon = 0.01$. ◀

4 PIVOT Algorithms for Correlation Clustering

4.1 Lower Bound

First we prove Theorem 3 which states that there is no PIVOT algorithm for Correlation Clustering with approximation factor better than 3.

Proof of Theorem 3. Let $G = ([2n], E)$ for some integer n, where the edge set E contains all pairs of nodes except for pairs of the form $(2k+1, 2k+2)$. In other words, the edge set of G contains all edges except for a perfect matching.

Note that if we create a single cluster containing all nodes, then the cost is exactly n. On the other hand, let u be the first choice that a PIVOT algorithm makes. If u is even, let $v = u - 1$, otherwise let $v = u + 1$. By definition of G, v is the only node not adjacent to u. Therefore, the algorithm creates two clusters—one containing all nodes except for v, and one containing only v. There are $2n - 2$ edges across the two clusters, and $n - 1$ missing edges in the big cluster, meaning that the cost is $3n - 3$.

Therefore, the approximation factor of any PIVOT algorithm is at least $(3n-3)/n = 3-\frac{3}{n}$. This proves the theorem, as for any constant less than 3, there exists a sufficiently large n such that $3 - \frac{3}{n}$ is larger than that constant. ◀

4.2 Optimal Deterministic PIVOT: 3-Approximation

A *covering* LP is a linear program of the form $\min_x\{\, cx \mid Ax \geq b \,\}$ where A, b, c, and x are restricted to vectors and matrices of non-negative entries. Covering LPs can be solved more efficiently than LPs in general and we rely on the following known machinery to prove Theorem 2:

▶ **Theorem 13** (Covering LPs, Combinatorial [30, 28])**.** *Any covering LP with at most N nonzero entries in the constraint matrix can be $(1+\epsilon)$-approximated by a combinatorial algorithm in time $\widetilde{O}(N\epsilon^{-3})$.*[5]

▶ **Theorem 14** (Covering LPs, Non-Combinatorial [4, 42])**.** *Any covering LP with at most N nonzero entries in the constraint matrix can be $(1+\epsilon)$-approximated in time $\widetilde{O}(N\epsilon^{-1})$.*

Of the two theorems, the time complexity of the algorithm promised by Theorem 14 is obviously better. However, the algorithm of Theorem 13 is remarkably simple in our setting and could thus prove to be faster in practice. Note that either theorem suffices to obtain a $(3+\epsilon)$-approximation for Correlation Clustering in $\widetilde{O}(n^3)$ time, for constant $\epsilon > 0$.

For completeness, and in order to demonstrate how simple the algorithm from Theorem 13 is in our setting, we include the pseudocode as Algorithm 3. In Appendix A we formally prove that Algorithm 3 indeed has the properties promised by Theorem 13.

The solution found by Algorithm 3 is used together with the framework by van Zuylen and Williamson [37], see Cluster in Algorithm 4. Cluster is discussed further in Appendix B, where the following lemmas are proven.

▶ **Lemma 15** (Correctness of Cluster)**.** *Assume that $x = \{\, x_{uv} \,\}_{uv}$ is a feasible solution to the LP in Figure 1. Then* Cluster$(G,\, x)$ *computes a correlation clustering of cost $3\sum_{uv} x_{uv}$. In particular, if x is an α-approximate solution to the LP (for some $\alpha \geq 1$), then* Cluster$(G,\, x)$ *returns a 3α-approximate correlation clustering.*

▶ **Lemma 16** (Running Time of Cluster, [37])**.** Cluster$(G,\, x)$ *runs in time $O(n^3)$.*

Given Theorems 13 and 14 we quickly prove Theorem 2.

Proof of Theorem 2. We compute a $(1+\epsilon/3)$-approximate solution x of the charging LP using Theorem 13 (that is, using the procedure Charge(G)). Plugging this solution x into

[5] The running time we state seems worse by a factor of ϵ^{-1} as compared to the theorems in [30, 28]. This is because the authors assume access to a machine model with exact arithmetic of numbers of size exponential in ϵ^{-1}. We can simulate this model using fixed-point arithmetic with a running time overhead of $\widetilde{O}(\epsilon^{-1})$.

Algorithm 3 The combinatorial algorithm to $(1 + O(\epsilon))$-approximate the LP in Figure 1 (Page 6) using the multiplicative weights update method. The general method was given by Garg and Könemann [30] and later refined by Fleischer [28]. We here use the notation $m(x) = \min_{uvw \in T(G)} x_{uv} + x_{vw} + x_{wu}$.

procedure CHARGE($G = (V, E)$)
1. Initialize x_{uv}, $x^*_{uv} \leftarrow 1$ for all $uv \in \binom{V}{2}$
2. **while** $\sum_{uv} x_{uv} < B := \left(\binom{n}{2}(1+\epsilon)\right)^{1/\epsilon} / (1+\epsilon)$ **do**
3. Find a bad triplet uvw minimizing $x_{uv} + x_{vw} + x_{wu}$
4. $x_{uv} \leftarrow (1+\epsilon) \cdot x_{uv}$
5. $x_{vw} \leftarrow (1+\epsilon) \cdot x_{vw}$
6. $x_{wu} \leftarrow (1+\epsilon) \cdot x_{wu}$
7. **if** $\left(\sum_{uv} x_{uv}\right) / m(x) < \left(\sum_{uv} x^*_{uv}\right) / m(x^*)$ **then**
8. **foreach** $uv \in \binom{V}{2}$ **do** $x^*_{uv} \leftarrow x_{uv}$
9. **return** $\{ x^*_{uv} / m(x^*) \}_{uv}$

Algorithm 4 The PIVOT algorithm by van Zuylen and Williamson [37]. Given a graph G and a good charging $\{ x_{uv} \}_{uv}$ (in the sense of Lemma 15), it computes a correlation clustering.

procedure CLUSTER($G = (V, E), x = \{ x_{uv} \}_{uv \in \binom{V}{2}}$)
1. $C \leftarrow \emptyset$
2. **while** $V \neq \emptyset$ **do**
3. Pick a pivot node $u \in V$ minimizing
$$\frac{\sum_{vw : uvw \in T(G)} 1}{\sum_{vw : uvw \in T(G)} x_{vw}}$$
4. Add a cluster containing u and all its neighbors to C
5. Remove u, its neighbors and all their incident edges from G
6. **return** C

CLUSTER(G, x) returns a $(3+\epsilon)$-approximate correlation clustering by Lemma 15. The total running time is bounded by $O(n^3)$ by Lemma 16 plus $\widetilde{O}(n^3\epsilon^{-3})$ by Theorem 13 (note that there are n^3 constraints, each affecting only a constant number of variables, hence the number of nonzeros in the constraint matrix is $N \leq O(n^3)$). For constant $\epsilon > 0$, this becomes $\widetilde{O}(n^3)$.

To obtain a 3-approximation, we observe that any correlation clustering has cost less than $\binom{n}{2}$. Hence, we can run the previous algorithm with $\epsilon = 1/\binom{n}{2}$ and the $(3+\epsilon)$-approximate solution is guaranteed to also be 3-approximate. The running time would be bounded by $\widetilde{O}(n^9)$. To improve upon this, we use the covering LP solver in Theorem 14 which runs in time $\widetilde{O}(n^3\epsilon^{-1})$. By again setting $\epsilon = 1/\binom{n}{2}$, the running time becomes $\widetilde{O}(n^5)$. ◀

References

1 Rakesh Agrawal, Alan Halverson, Krishnaram Kenthapadi, Nina Mishra, and Panayiotis Tsaparas. Generating labels from clicks. In Ricardo Baeza-Yates, Paolo Boldi, Berthier A. Ribeiro-Neto, and Berkant Barla Cambazoglu, editors, *Proceedings of the Second International Conference on Web Search and Web Data Mining, WSDM 2009, Barcelona, Spain, February 9-11, 2009*, pages 172–181. ACM, 2009. doi:10.1145/1498759.1498824.

2 Nir Ailon and Moses Charikar. Fitting tree metrics: Hierarchical clustering and phylogeny. *SIAM J. Comput.*, 40(5):1275–1291, 2011. Announced at FOCS'05. doi:10.1137/100806886.

3 Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 55(5):23:1–23:27, 2008. Announced in STOC 2005. doi:10.1145/1411509.1411513.

4 Zeyuan Allen-Zhu and Lorenzo Orecchia. Nearly linear-time packing and covering LP solvers – achieving width-independence and -convergence. *Math. Program.*, 175(1-2):307–353, 2019. doi:10.1007/s10107-018-1244-x.

5 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 522–539. SIAM, 2021. doi:10.1137/1.9781611976465.32.

6 Arvind Arasu, Christopher Ré, and Dan Suciu. Large-scale deduplication with constraints using dedupalog. In Yannis E. Ioannidis, Dik Lun Lee, and Raymond T. Ng, editors, *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, pages 952–963. IEEE Computer Society, 2009. doi:10.1109/ICDE.2009.43.

7 Sepehr Assadi and Chen Wang. Sublinear time and space algorithms for correlation clustering via sparse-dense decompositions. *CoRR*, abs/2109.14528, 2021. URL: https://arxiv.org/abs/2109.14528, arXiv:2109.14528.

8 Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Mach. Learn.*, 56(1-3):89–113, 2004. doi:10.1023/B:MACH.0000033116.57574.95.

9 Soheil Behnezhad, Moses Charikar, Weiyun Ma, and Li-Yang Tan. Almost 3-approximate correlation clustering in constant rounds. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 720–731. IEEE, 2022. doi:10.1109/FOCS54457.2022.00074.

10 Soheil Behnezhad, Moses Charikar, Weiyun Ma, and Li-Yang Tan. Single-pass streaming algorithms for correlation clustering. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 819–849. SIAM, 2023. URL: https://doi.org/10.1137/1.9781611977554.ch33, doi:10.1137/1.9781611977554.CH33.

11 Francesco Bonchi, Aristides Gionis, and Antti Ukkonen. Overlapping correlation clustering. *Knowl. Inf. Syst.*, 35(1):1–32, 2013. doi:10.1007/s10115-012-0522-9.

12 Mark Bun, Marek Eliás, and Janardhan Kulkarni. Differentially private correlation clustering. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 1136–1146. PMLR, 2021. URL: http://proceedings.mlr.press/v139/bun21a.html.

13 Melanie Cambus, Fabian Kuhn, Etna Lindy, Shreyas Pai, and Jara Uitto. *A (3+ε)-Approximate Correlation Clustering Algorithm in Dynamic Streams*, pages 2861–2880. SIAM, 2024. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611977912.101, arXiv:https://epubs.siam.org/doi/pdf/10.1137/1.9781611977912.101, doi:10.1137/1.9781611977912.101.

14 Nairen Cao, Vincent Cohen-Addad, Euiwoong Lee, Shi Li, Alantha Newman, and Lukas Vogl. Understanding the cluster linear program for correlation clustering. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 1605–1616. ACM, 2024. doi:10.1145/3618260.3649749.

15 Nairen Cao, Shang-En Huang, and Hsin-Hao SU. *Breaking 3-Factor Approximation for Correlation Clustering in Polylogarithmic Rounds*, pages 4124–4154. SIAM, 2024. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611977912.143, arXiv:https://epubs.siam.org/doi/pdf/10.1137/1.9781611977912.143, doi:10.1137/1.9781611977912.143.

16 Deepayan Chakrabarti, Ravi Kumar, and Kunal Punera. A graph-theoretic approach to webpage segmentation. In Jinpeng Huai, Robin Chen, Hsiao-Wuen Hon, Yunhao Liu, Wei-Ying Ma, Andrew Tomkins, and Xiaodong Zhang, editors, *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008*, pages 377–386. ACM, 2008. doi:10.1145/1367497.1367549.

17 Sayak Chakrabarty and Konstantin Makarychev. Single-pass pivot algorithm for correlation clustering. keep it simple! *CoRR*, abs/2305.13560, 2023. URL: https://doi.org/10.48550/arXiv.2305.13560, arXiv:2305.13560, doi:10.48550/ARXIV.2305.13560.

18 Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. *J. Comput. Syst. Sci.*, 71(3):360–383, 2005. Announced in FOCS 2003. doi:10.1016/j.jcss.2004.10.012.

19 Shuchi Chawla, Konstantin Makarychev, Tselil Schramm, and Grigory Yaroslavtsev. Near optimal LP rounding algorithm for correlation clustering on complete and complete k-partite graphs. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 219–228. ACM, 2015. doi:10.1145/2746539.2746604.

20 Yudong Chen, Sujay Sanghavi, and Huan Xu. Clustering sparse graphs. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 2213–2221, 2012. URL: https://proceedings.neurips.cc/paper/2012/hash/1e6e0a04d20f50967c64dac2d639a577-Abstract.html.

21 Michael B. Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 938–942. ACM, 2019. doi:10.1145/3313276.3316303.

22 Vincent Cohen-Addad, Chenglin Fan, Euiwoong Lee, and Arnaud de Mesmay. Fitting metrics and ultrametrics with minimum disagreements. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 301–311. IEEE, 2022. doi:10.1109/FOCS54457.2022.00035.

23 Vincent Cohen-Addad, Silvio Lattanzi, Slobodan Mitrovic, Ashkan Norouzi-Fard, Nikos Parotsidis, and Jakub Tarnawski. Correlation clustering in constant many parallel rounds. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 2069–2078. PMLR, 2021. URL: http://proceedings.mlr.press/v139/cohen-addad21b.html.

24 Vincent Cohen-Addad, Euiwoong Lee, Shi Li, and Alantha Newman. Handling correlated rounding error via preclustering: A 1.73-approximation for correlation clustering. In *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*, pages 1082–1104. IEEE, 2023. doi:10.1109/FOCS57990.2023.00065.

25 Vincent Cohen-Addad, Euiwoong Lee, and Alantha Newman. Correlation clustering with sherali-adams. *CoRR*, abs/2207.10889, 2022. arXiv:2207.10889, doi:10.48550/arXiv.2207.10889.

26 Vincent Cohen-Addad, David Rasmussen Lolck, Marcin Pilipczuk, Mikkel Thorup, Shuyi Yan, and Hanwen Zhang. Combinatorial correlation clustering. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 1617–1628. ACM, 2024. doi:10.1145/3618260.3649712.

27 Erik D. Demaine, Dotan Emanuel, Amos Fiat, and Nicole Immorlica. Correlation clustering in general weighted graphs. *Theor. Comput. Sci.*, 361(2-3):172–187, 2006. doi:10.1016/j.tcs.2006.05.008.

28 Lisa Fleischer. A fast approximation scheme for fractional covering problems with variable upper bounds. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 1001–1010. SIAM, 2004. URL: http://dl.acm.org/citation.cfm?id=982792.982942.

29 Fedor V. Fomin, Stefan Kratsch, Marcin Pilipczuk, Michal Pilipczuk, and Yngve Villanger. Tight bounds for parameterized complexity of cluster editing with a small number of clusters. *J. Comput. Syst. Sci.*, 80(7):1430–1447, 2014. doi:10.1016/j.jcss.2014.04.015.

30 Naveen Garg and Jochen Koenemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, FOCS '98, page 300, USA, 1998. IEEE Computer Society.

31 Svante Janson. Tail bounds for sums of geometric and exponential variables. *Statistics & Probability Letters*, 135(C):1–6, 2018. doi:10.1016/j.spl.2017.11.017.

32 Dmitri V. Kalashnikov, Zhaoqi Chen, Sharad Mehrotra, and Rabia Nuray-Turan. Web people search via connection analysis. *IEEE Trans. Knowl. Data Eng.*, 20(11):1550–1565, 2008. doi:10.1109/TKDE.2008.78.

33 Sungwoong Kim, Sebastian Nowozin, Pushmeet Kohli, and Chang Dong Yoo. Higher-order correlation clustering for image segmentation. In John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*, pages 1530–1538, 2011. URL: https://proceedings.neurips.cc/paper/2011/hash/98d6f58ab0dafbb86b083a001561bb34-Abstract.html.

34 Domenico Mandaglio, Andrea Tagarelli, and Francesco Gullo. Correlation clustering with global weight bounds. In Nuria Oliver, Fernando Pérez-Cruz, Stefan Kramer, Jesse Read, and José Antonio Lozano, editors, *Machine Learning and Knowledge Discovery in Databases. Research Track - European Conference, ECML PKDD 2021, Bilbao, Spain, September 13-17, 2021, Proceedings, Part II*, volume 12976 of *Lecture Notes in Computer Science*, pages 499–515. Springer, 2021. doi:10.1007/978-3-030-86520-7_31.

35 Gregory J. Puleo and Olgica Milenkovic. Correlation clustering with constrained cluster sizes and extended weights bounds. *SIAM J. Optim.*, 25(3):1857–1872, 2015. doi:10.1137/140994198.

36 Jan van den Brand. A deterministic linear program solver in current matrix multiplication time. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 259–278. SIAM, 2020. doi:10.1137/1.9781611975994.16.

37 Anke van Zuylen and David P. Williamson. Deterministic pivoting algorithms for constrained ranking and clustering problems. *Math. Oper. Res.*, 34(3):594–620, 2009. Announced in SODA 2007. doi:10.1287/moor.1090.0385.

38 Nate Veldt. Correlation clustering via strong triadic closure labeling: Fast approximation algorithms and practical lower bounds. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 22060–22083. PMLR, 2022. URL: https://proceedings.mlr.press/v162/veldt22a.html.

39 Nate Veldt, David F. Gleich, and Anthony Wirth. A correlation clustering framework for community detection. In Pierre-Antoine Champin, Fabien Gandon, Mounia Lalmas, and Panagiotis G. Ipeirotis, editors, *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pages 439–448. ACM, 2018. doi:10.1145/3178876.3186110.

40 Michael D. Vose. A linear algorithm for generating random numbers with a given distribution. *IEEE Trans. Software Eng.*, 17(9):972–975, 1991. doi:10.1109/32.92917.

41 Alastair J. Walker. New fast method for generating discrete random numbers with arbitrary frequency distributions. *Electronics Letters*, 10:127–128(1), April 1974.

42 Di Wang, Satish Rao, and Michael W. Mahoney. Unified acceleration method for packing and covering problems via diameter reduction. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 50:1–50:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.50.

43 Julian Yarkony, Alexander T. Ihler, and Charless C. Fowlkes. Fast planar correlation clustering for image segmentation. In Andrew W. Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *Computer Vision - ECCV 2012 - 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part VI*, volume 7577 of *Lecture Notes in Computer Science*, pages 568–581. Springer, 2012. doi:10.1007/978-3-642-33783-3_41.

A Analysis of CHARGE

In this appendix we prove that CHARGE (Algorithm 3, Page 14) computes a $(1 + O(\epsilon))$-approximation to the charging LP of Figure 1 in time $O(n^3 \operatorname{poly}(\log n/\epsilon))$, thus matching the guarantees of Theorem 13. This algorithm was first developed by Garg and Könemann [30] and later refined by Fleischer [28].

We analyze the algorithm using a primal–dual approach: We first argue that CHARGE(G) constructs a good solution (up to rescaling) to the primal LP, and then compare this to an optimal dual solution. The gap between primal and dual value is bounded by $1 + O(\epsilon)$, and by the weak duality theorem it follows that the primal solution computed by the algorithm is $(1 + O(\epsilon))$-approximately optimal.

Let us introduce some notation: Let t denote the total number of iterations of CHARGE(G). For an iteration k, let $x_{uv}^{(k)}$ denote the current value of x_{uv}. We also write $s^{(k)} = \sum_{uv} x_{uv}^{(k)}$ and $m^{(k)} = m(x^{(k)}) = \min_{uvw \in T(G)} x_{uv}^{(k)} + x_{vw}^{(k)} + x_{wu}^{(k)}$. Finally, we set $s = \sum_{uv} x_{uv}^*$ and $m = m(x^*)$. We have $s/m \leq s^{(k)}/m^{(k)}$ for all iterations k by the way that x^* is constructed.

▶ **Observation 17** (Primal Solution). *$\{ x_{uv}^*/m \}_{uv}$ is a feasible primal solution with value s/m.*

Proof. For any bad triplet uvw we have that $x_{uv}^* + x_{vw}^* + x_{wu}^* \geq m$ by definition. Hence $\{ x_{uv}^*/m \}$ satisfies all primal constraints and is feasible. Its value is s/m by definition. ◀

▶ **Observation 18** (Dual Solution). *Let y_{uvw} be the number of iterations in which uvw was picked in Line 3, scaled by $(\log_{1+\epsilon}(B) + 1)^{-1}$. Then $\{ y_{uvw} \}_{uvw}$ is a feasible dual solution with value $t/(\log_{1+\epsilon}(B) + 1)$.*

Proof. In order to prove feasibility, we need to argue that uvw is selected in at most $\log_{1+\epsilon}(B) + 1$ iterations. Indeed, in every iteration where uvw is picked we multiplicatively increase x_{uv} by $1+\epsilon$. This can happen at most $\log_{1+\epsilon}(B)+1$ times before the loop terminates. The value of $\{ y_{uvw} \}_{uvw}$ is $\sum_{uvw} y_{uvw} = t/(\log_{1+\epsilon}(B) + 1)$. ◀

We additionally need the following technical lemma.

▶ **Lemma 19.** *For any iteration k, it holds that $s^{(k)} \leq \binom{n}{2} \cdot \exp(\epsilon k m/s)$.*

Proof. The proof is by induction. For $k = 0$ the statement is clear since we initially assign $x_{uv} \leftarrow 1$ for all pairs uv. For $k > 0$ we have

$$
\begin{aligned}
s^{(k)} &= s^{(k-1)} + \epsilon m^{(k-1)} && (1)\\
&\leq s^{(k-1)} \cdot (1 + \epsilon m/s) && (2)\\
&\leq \tbinom{n}{2} \cdot \exp(\epsilon(k-1)m/s) \cdot (1 + \epsilon m/s) && (3)\\
&\leq \tbinom{n}{2} \cdot \exp(\epsilon k m/s), && (4)
\end{aligned}
$$

where we used (1) the update rule in Lines 4–6, (2) the fact that $s/m \leq s^{(k)}/m^{(k)}$, (3) the induction hypothesis and (4) the fact that $1 + x \leq \exp(x)$ for all real x. ◀

In combination we obtain the correctness of CHARGE(G):

▶ **Lemma 20** (Correctness of CHARGE). *The algorithm CHARGE(G) correctly computes a $(1 + O(\epsilon))$-approximate solution $\{ x_{uv}^*/m \}_{uv}$ to the charging LP.*

Proof. In order to argue that the primal solution $\{ x^*_{uv}/m \}_{uv}$ from Observation 17 is $(1 + O(\epsilon))$-approximate, it suffices to bound the gap to its corresponding dual solution from Observation 18. Their gap is bounded by

$$\frac{s/m}{t/(\log_{1+\epsilon}(B) + 1)} = \frac{s(\log_{1+\epsilon}(B) + 1)}{mt}$$

Recall that the algorithm terminates with $s \geq B$, thus by Lemma 19 we obtain that $B \leq \binom{n}{2} \cdot \exp(\epsilon tm/s)$, or equivalently $tm/s \geq \epsilon^{-1} \ln(B/\binom{n}{2})$. It follows that the gap is bounded by

$$\begin{aligned} &\leq \frac{\epsilon(\log_{1+\epsilon}(B) + 1)}{\ln(B/\binom{n}{2})} \\ &= \frac{\ln(B(1+\epsilon))}{\ln(B/\binom{n}{2})} \cdot \frac{\epsilon}{\ln(1+\epsilon)} \end{aligned}$$

By setting $B = (\binom{n}{2}(1+\epsilon))^{1/\epsilon}/(1+\epsilon)$ as in the algorithm, the first term becomes $1/(1-\epsilon)$ and thus

$$\begin{aligned} &= \frac{1}{1-\epsilon} \cdot \frac{\epsilon}{\ln(1+\epsilon)} \\ &\leq \frac{1}{1-\epsilon} \cdot \frac{\epsilon}{\epsilon - \epsilon^2/2} \\ &\leq 1 + O(\epsilon). \end{aligned}$$

◀

▶ **Lemma 21** (Running Time of Charge). *The running time of* Charge(G) *is bounded by* $O(n^3 \operatorname{poly}(\log n/\epsilon))$.

Proof. Any variable x_{uv} can be increased at most $\log_{1+\epsilon}(B) + 1$ times. Hence, the total number of iterations is bounded by $O(n^2 \log_{1+\epsilon}(B)) = O(n^2 \operatorname{poly}(\log n/\epsilon))$. To efficiently implement the loop, we use a priority queue to maintain $\{ x_{uv} + x_{vw} + x_{wu} \}_{uvw \in T(G)}$. The initialization takes time $O(n^3 \log n)$. In each iteration we can select the minimum-weight bad triplet in time $O(\log n)$ by a single query. Changing the three variables x_{uv}, x_{vw}, x_{wu} affects at most $O(n)$ entries in the queue and therefore takes time $O(n \log n)$.

In the previous paragraph we assumed that arithmetic operations run in unit time. However, observe that we work with numbers of magnitude up to B and precision ϵ. We can perform arithmetic operations on numbers of that size in time $\operatorname{poly}(\log(B/\epsilon)) = \operatorname{poly}(\log n/\epsilon)$. Therefore, the total running time increases by a factor $\operatorname{poly}(\log n/\epsilon)$ and is still bounded by $O(n^3 \operatorname{poly}(\log n/\epsilon))$ as claimed. ◀

B Correctness of CLUSTER

We borrow the algorithm from van Zuylen and Williamson [37], see Cluster in Algorithm 4. We present our analysis using the charging LP relaxation in Lemma 15. To obtain a best-possible PIVOT algorithm, think of the input x as an exact solution to the LP in Figure 1. We denote its optimal value by $\mathrm{OPT}^{(\mathrm{LP})}$ and the optimal value of the correlation clustering by $\mathrm{OPT}^{(\mathrm{CC})}$.

▶ **Lemma 15** (Correctness of Cluster). *Assume that* $x = \{ x_{uv} \}_{uv}$ *is a feasible solution to the LP in Figure 1. Then* Cluster(G, x) *computes a correlation clustering of cost* $3 \sum_{uv} x_{uv}$. *In particular, if* x *is an* α*-approximate solution to the LP (for some* $\alpha \geq 1$*), then* Cluster(G, x) *returns a* 3α*-approximate correlation clustering.*

Proof. Let $G^{(i)} = (V^{(i)}, E^{(i)})$ denote the graph G after the i-th iteration of the loop, i.e., $G^{(0)}$ is the initial graph G and $G^{(t)}$ is the empty graph for t the total number of iterations. Let u_i denote the pivot node selected in the i-th iteration of the algorithm. It is easy to check that the total number of violated preferences in the clustering C is equal to

$$\sum_{i=0}^{t-1} \sum_{\substack{vw: \\ u_i vw \in T(G^{(i)})}} 1$$

Indeed, in the i-th iteration we violate exactly the negative preferences of pairs vw which are both neighbors of u, and the positive preferences of pairs vw for which exactly one is a neighbor of u. In any such case and only in these cases, uvw is a bad triplet. Using that x is a feasible LP solution, we obtain the following bound:

$$\begin{aligned}\sum_{u \in V^{(i)}} \sum_{\substack{vw: \\ uvw \in T(G^{(i)})}} 1 &= 3 \cdot \sum_{uvw \in T(G^{(i)})} 1 \\ &\leq 3 \cdot \sum_{uvw \in T(G^{(i)})} x_{uv} + x_{vw} + x_{wu} \\ &= 3 \cdot \sum_{u \in V^{(i)}} \sum_{\substack{vw: \\ uvw \in T(G^{(i)})}} x_{vw}\end{aligned}$$

By the way we picked u_i in Line 3, we have that $\sum_{vw: uvw \in T(G^{(i)})} 1 \leq 3 \cdot \sum_{vw: uvw \in T(G^{(i)})} x_{vw}$. It follows that the total cost of the clustering is bounded by

$$\begin{aligned}\sum_{i=0}^{t-1} \sum_{\substack{vw: \\ u_i vw \in T(G^{(i)})}} 1 &\leq 3 \cdot \sum_{i=0}^{t-1} \sum_{\substack{vw: \\ u_i vw \in T(G^{(i)})}} x_{vw} \\ &\leq 3 \cdot \sum_{vw \in \binom{V}{2}} x_{vw}\end{aligned}$$

Here, we used that every pair of vertices is counted for in exactly one graph $G^{(i)}$. This finishes the first part of the lemma.

For the second part, assume that x is an α-approximate optimal solution, i.e., $\sum_{uv} x_{uv} \leq \alpha \operatorname{OPT}^{(\mathrm{LP})}$. We claim that $\operatorname{OPT}^{(\mathrm{LP})} \leq \operatorname{OPT}^{(\mathrm{CC})}$. Indeed, we can plug in any correlation clustering into the primal LP as follows: For every pair uv whose preference is violated set $x_{uv} = 1$ and for all other pairs set $x_{uv} = 0$. The important observation is that in any correlation clustering solution, we charge at least one edge in every bad triplet. Hence, the constraints of the LP are satisfied, and we obtain a feasible solution of value $\operatorname{OPT}^{(\mathrm{CC})}$. It follows that the correlation clustering constructed by $\textsc{Cluster}(G, x)$ has cost at most

$$3 \cdot \sum_{uv \in \binom{V}{2}} x_{uv} \leq 3\alpha \cdot \operatorname{OPT}^{(\mathrm{LP})} \leq 3\alpha \cdot \operatorname{OPT}^{(\mathrm{CC})}$$ ◀

▶ **Lemma 16** (Running Time of Cluster, [37]). *$\textsc{Cluster}(G, x)$ runs in time $O(n^3)$.*

Proof. We can efficiently implement $\textsc{Cluster}(G, x)$ by first precomputing $\sum_{vw: uvw \in T(G)} 1$ and $\sum_{vw: uvw \in T(G)} x_{vw}$ for every node u in time $O(n^3)$. Then in the remaining algorithm we can efficiently select the pivot (for instance, by exhaustively checking all nodes u) and remove its cluster from the graph. For every vertex u which is removed from the graph in that way, we can enumerate all bad triplets involving u and update the precomputed quantities appropriately. Since every node is removed exactly once, the total running time is bounded by $O(n^3)$. ◀

C Analysis of Constrained Correlation Clustering

Running Time. We first prove the running time of our algorithm.

▶ **Lemma 22.** *The running time of Algorithm 2 is $O(n(n+m)+T(n,\binom{n}{2}))$, where $T(n',m')$ is an upper bound on the running time of the PIVOT algorithm we use on a graph with n' nodes and m' edges.*

Proof. Computing the connected components of (V,F) takes $O(n+|F|)$ time. Adding all the edges between supernodes takes $O(n^2)$ time. Then we can contract the supernodes (allowing parallel edges) in $O(n^2)$ time. Removing the edges between hostile supernodes takes $O(m+|H|)$ time.

For the steps in the loop of Line 6, notice that there are at most m edges connecting distinct supernodes, as the only edges we added were internal in supernodes. We can iterate over all these edges uv, and over all supernodes W. If W is connected with $s(u)$ and hostile with $s(v)$, then we remove uv and an arbitrary edge connecting W with $s(u)$, and similarly if W is connected with $s(v)$ and hostile with $s(u)$. This takes $O(n\cdot m)$ time. Each pair of edges removed trivially satisfies the requirements of Line 6. As we do not add edges in this step, it is impossible that when finishing there is still a pair of edges e_1,e_2 that needed to be removed; when processing e_1, we would remove e_1 along with some other edge (possibly different from e_2).

Rounding the connections between pairs of supernodes is done in $O(n^2)$ time.

The final graph may have at most $\binom{n}{2}$ edges (even if m was much smaller, e.g. in the case where all nodes belong in the same supernode), therefore the time spent by the PIVOT algorithm is at most $T(n,\binom{n}{2})$.

The claimed bound follows by both $|F|$ and $|H|$ being $O(n^2)$. ◀

Correctness. We finally prove correctness—that is, we prove that either the final algorithm satisfies all hard constraints, or no clustering can satisfy the hard constraints and the algorithm outputs "Impossible".

We start with showing that our algorithm correctly detects all cases where the hard constraints are impossible to satisfy.

▶ **Lemma 23.** *Given an instance (V,E,F,H) of Constrained Correlation Clustering, the graph $G' \leftarrow \textsc{Transform}(V,E,F,H)$ is equal to $(\emptyset,\emptyset)$ if and only if the Constrained Correlation Clustering instance is impossible to satisfy.*

Proof. We show that if two hostile nodes are in the same supernode, then the instance is not satisfiable and the algorithm correctly determines it; on the other hand, if no such hostile nodes exist, then there exists at least one valid clustering.

It holds that $G' = (\emptyset,\emptyset)$ if there exist nodes u_1,u_2 such that u_1,u_2 are in the same connected component of (V,F) and $u_1u_2 \in H$. Then u_1,u_2 must be in the same cluster (Lemma 9) and not be in the same cluster (because $u_1u_2 \in H$). Therefore the instance is impossible to satisfy.

Otherwise, no u_1,u_2 in the same supernode are hostile. Creating a cluster for each supernode is a valid clustering. To see this, notice that no hostility constraint is violated, by hypothesis. All friendliness constraints are satisfied because any two nodes that must be linked belong in the same supernode, and thus in the same cluster. Therefore such an instance is satisfiable. ◀

In the following we can thus assume that we have a satisfiable instance with no supernode being hostile to itself. The next lemma shows that friendly nodes have the same neighborhood and are connected.

▶ **Lemma 24.** *Given a satisfiable instance (V, E, F, H) of Constrained Correlation Clustering, let $G' \leftarrow \textsc{Transform}(V, E, F, H)$. For any $uv \in F$, it holds that u, v are connected in G' and their neighborhoods are the same.*

Proof. The idea is that all nodes in the same supernode are explicitly connected by the algorithm, in Line 3. Then all nodes of the same supernode connect to the exact same nodes due to the rounding step in Line 9.

More formally, as $uv \in F$, they are trivially both in the same connected component of (V, F). Thus they are in the same supernode.

As $s(u) = s(v)$, u and v get connected in Line 3. All subsequent steps only modify edges $u'v'$ where $s(u') \neq s(v')$, therefore u, v remain connected in G'. Similarly both u and v are connected with all other nodes in $s(u)$.

For nodes $w \notin s(u)$, when $\{ s(u), s(w) \}$ is processed in the loop of Line 9, either both u and v get connected to w or both get disconnected by w. ◀

Similarly, hostile nodes are disconnected and do not share any common neighbor.

▶ **Lemma 25.** *Given a satisfiable instance (V, E, F, H) of Constrained Correlation Clustering, let $G' \leftarrow \textsc{Transform}(V, E, F, H)$. For any $uv \in H$ it holds that u, v are not connected in G' and they have no common neighbor.*

Proof. The idea is that all nodes in hostile supernodes are explicitly disconnected by the algorithm, in Line 4. Then if two hostile nodes share a common neighbor, we drop both edges in Line 6.

More formally, as the instance is satisfiable, we have that $s(u) \neq s(v)$ by Lemma 23. Therefore no node in $s(u)$ is connected with a node in $s(v)$ after Line 4. In Line 6 we only remove edges, meaning that when we process $\{ s(u), s(v) \}$ in the loop of Line 9, the two supernodes are not connected, and they stay like that. Thus, u, v (and even $s(u), s(v)$) are not connected in G'.

After Line 6, for any supernode W we have that at least one from $s(u), s(v)$ are not connected with W, or else the loop would not terminate. Assume without loss of generality that $s(u)$ is not connected with W. Therefore, $s(u)$ is also not connected with W after the loop of Line 9, meaning that even if v is connected with a node $w \in W$, u is not as $s(u)$ is not connected with $s(w) = W$. This guarantees that they have no common neighbor. ◀

With these lemmas, we can conclude that a PIVOT algorithm on G' gives a clustering that satisfies the hard constraints. This was already observed in [37]; we include a short proof for intuition, as we also use this lemma in Appendix D.

▶ **Lemma 26.** *Let (V, E, F, H) be a satisfiable instance of Constrained Correlation Clustering and $G' = (V, E')$ be a graph such that any two friendly nodes are connected and have the same neighborhood in G', while hostile nodes are not connected and have no common neighbor in G'. Then applying a PIVOT algorithm on G' gives a clustering that satisfies the hard constraints. In particular, this holds for $G' = \textsc{Transform}(V, E, F, H)$.*

Proof. The idea is that due to the assumptions, the choice of the first pivot does not violate any hard constraint. As PIVOT algorithms progress, they work with induced subgraphs of

the original graph, which also satisfy the assumptions, and therefore no hard constraint is ever violated.

For the sake of contradiction, assume that two hostile nodes u, v are placed in the same cluster by a PIVOT algorithm. By definition of a PIVOT algorithm, this happens when we work with some $V' \subseteq V$ on the induced subgraph $G'[V']$, and we pivot on a node w that is connected with both u, v. As w is connected with both u, v in $G'[V']$, it is also connected with u, v in G'. But this contradicts the assumption on hostile nodes.

Similarly, for the sake of contradiction assume that two friendly nodes u, v are put in separate clusters by a PIVOT algorithm. Without loss of generality assume that u is the first to be placed in a cluster that does not contain v. Again, this happens when we work with some $V' \subseteq V$ on the induced subgraph $G'[V']$, and we pivot on a node w that is connected with u but not with v. As $G'[V']$ is an induced subgraph, w is connected with u but not with v in G'. But this contradicts the assumption on friendly nodes.

Therefore, by Lemmas 24 and 25 the claim holds for $G' = \textsc{Transform}(V, E, F, H)$. ◀

D Node-Weighted Correlation Clustering

Deterministic Algorithm. We first give the deterministic PIVOT algorithms for Node-Weighted Correlation Clustering (Theorem 4). We summarize the pseudocode in Algorithm 5. The analysis of the deterministic algorithm is similar to the PIVOT algorithm in Section 4.

Figure 3 The LP relaxation for Node-Weighted Correlation Clustering.

$$\begin{aligned} \min \quad & \sum_{uv \in \binom{V}{2}} x_{uv} \\ \text{s.t.} \quad & \frac{x_{uv}}{\omega_u \omega_v} + \frac{x_{vw}}{\omega_v \omega_w} + \frac{x_{wu}}{\omega_w \omega_u} \geq 1 \quad \forall uvw \in T(G), \\ & x_{uv} \geq 0 \quad \forall uv \in \binom{V}{2} \end{aligned}$$

Algorithm 5 The adapted PIVOT algorithm to $(3+\epsilon)$-approximate Node-Weighted Correlation Clustering.

Compute a $(1+\frac{\epsilon}{3})$-approximate solution $x = \{ x_{uv} \}_{uv \in \binom{V}{2}}$ of the LP in Figure 3
$C \leftarrow \emptyset$
while $V \neq \emptyset$ **do**
  Pick a pivot node $u \in V$ minimizing

$$\frac{\sum_{vw : uvw \in T(G)} \omega_v \omega_w}{\sum_{vw : uvw \in T(G)} x_{vw}}$$

  Add a cluster containing u and all its neighbors to C
  Remove u, its neighbors and all their incident edges from G
return C

▶ **Lemma 27** (Correctness of Algorithm 5). *Algorithm 5 correctly approximates Node-Weighted Correlation Clustering with approximation factor* $3+\epsilon$.

Proof. We use the same notation as in Lemma 15. That is, let $G^{(i)} = (V^{(i)}, E^{(i)})$ denote the graph G after removing the i-th cluster and let u_i denote the i-th pivot node. By the same reasoning as in Lemma 15, the total cost of the node-weighted clustering constructed by the algorithm is exactly

$$\sum_{i=0}^{t-1} \sum_{\substack{vw: \\ u_i vw \in T(G^{(i)})}} \omega_v \omega_w.$$

In order to bound this cost, we again bound the cost of selecting the average node u as a pivot—this time however, we weight the nodes proportional to their weights ω_u:

$$\begin{aligned}
\sum_{u \in V^{(i)}} \omega_u \cdot \sum_{\substack{vw: \\ uvw \in T(G^{(i)})}} \omega_v \omega_w &= 3 \cdot \sum_{uvw \in T(G^{(i)})} \omega_u \omega_v \omega_w \\
&\leq 3 \cdot \sum_{uvw \in T(G^{(i)})} \omega_u x_{vw} + \omega_v x_{wu} + \omega_w x_{uv} \\
&= 3 \cdot \sum_{u \in V^{(i)}} \omega_u \cdot \sum_{\substack{vw: \\ uvw \in T(G^{(i)})}} x_{vw}.
\end{aligned}$$

In the second step, we applied the LP constraint. Using this inequality, we conclude that for any pivot node the ratio in Line 4 is bounded by 3. Assuming that x is a $(1+\frac{\epsilon}{3})$-approximate solution to the LP in Figure 3, we obtain the following upper bound on the cost of the node-weighted correlation clustering:

$$\begin{aligned}
\sum_{i=0}^{t-1} \sum_{\substack{vw \\ u_i vw \in T(G^{(i)})}} \omega_v \omega_w &\leq 3 \cdot \sum_{i=0}^{t-1} \sum_{\substack{vw \\ u_i vw \in T(G^{(i)})}} x_{vw} \\
&\leq 3 \cdot \sum_{vw \in V} x_{vw} \\
&\leq (3+\epsilon) \cdot \mathrm{OPT}^{(\mathrm{LP})} \\
&\leq (3+\epsilon) \cdot \mathrm{OPT}^{(\mathrm{NWCC})}.
\end{aligned}$$

Here, in order to bound $\mathrm{OPT}^{(\mathrm{LP})} \leq \mathrm{OPT}^{(\mathrm{NWCC})}$ (the optimal cost of the node-weighted correlation clustering), we argue that any node-weighted correlation clustering can be turned into a feasible solution of the LP in Figure 3. Indeed, for any pair uv whose preference is violated assign $x_{uv} = \omega_u \omega_v$ and for any pair uv whose preference is respected assign $x_{uv} = 0$. Recalling that every bad triplet involves at least one pair whose preference was violated we conclude that all constraints of the LP are satisfied. Thus x is a feasible solution and we have that $\mathrm{OPT}^{(\mathrm{LP})} \leq \mathrm{OPT}^{(\mathrm{NWCC})}$. ◀

Proof of Theorem 4. We use Algorithm 5. The correctness follows from the previous Lemma 27. To appropriately bound the running time, we use the same insight as in Section 4: Since the LP in Figure 3 is a covering LP, we can use the combinatorial algorithm in Theorem 13 to approximate the LP in Line 1 in time $\widetilde{O}(n^3 \epsilon^{-3})$. This proves the first part of the theorem.

For the second part we instead use an all-purpose LP solver to find an exact solution to the LP in Line 1 in time $O((n^3)^{2.373}) = O(n^{7.119})$ [21, 36, 5]. ◀

Algorithm 6 The randomized PIVOT algorithm computing an expected 3-approximation of Node-Weighted Correlation Clustering.

```
Initialize the weighted sampling data structure on V with weights {ω_u}_{u∈V}
C ← ∅
while V ≠ ∅ do
    u ← SAMPLE()
    Add a cluster containing u and all its neighbors to C
    Remove u, its neighbors and all their incident edges from G
    Run REMOVE(u) and REMOVE(v) for all neighbors v of u
return C
```

Randomized Algorithm. In this section we describe our optimal randomized PIVOT algorithm for Node-Weighted Correlation Clustering (Theorem 5). As the decisive ingredient, we provide a data structure to perform weighted sampling on a decremental set:

▶ **Lemma 28** (Weighted Sampling). *Let A be a set of initially n objects with associated weights $\{\omega_a\}_{a\in A}$. There is a data structure supporting the following operations on A:*

- SAMPLE(): *Samples and removes an element $a \in A$, where $a \in A$ is selected with probability $\omega_a / \sum_{a'\in A} \omega_{a'}$.*
- REMOVE(a): *Removes a from A.*

The total time to initialize the data structure and to run the previous operations until A is empty is bounded by $O(n)$, with high probability $1 - \frac{1}{n^c}$, for any constant $c > 0$.

We postpone the proof of Lemma 28 for now and first analyze the randomized algorithm in Algorithm 6. It can be seen as a natural generalization of the sampling algorithm from [3].

▶ **Lemma 29** (Correctness of Algorithm 6). *Algorithm 6 correctly approximates Node-Weighted Correlation Clustering with expected approximation factor 3.*

Proof. We borrow the notation from Lemma 27, writing $G^{(i)} = (V^{(i)}, E^{(i)})$ for the graph after removing the i-th cluster and u_i for the i-th pivot node. Assuming the correctness of the sampling data structure (Lemma 28), u_i is sampled from V_i with probability $\omega_u / \sum_{v\in V} \omega_v$. Again, the total cost of the clustering computed by Algorithm 6 is exactly

$$\sum_{i=0}^{t-1} \sum_{\substack{vw: \\ u_i vw \in T(G^{(i)})}} \omega_v \omega_w.$$

Let $x = \{x_{uv}\}_{uv}$ denote an optimal solution to the LP in Figure 3. (In contrast to the deterministic algorithm, here we do not compute the solution.) To bound the inner sum in

expectation, we use the same computation as in Lemma 27, relying on the LP constraint:

$$\begin{aligned}
\mathop{\mathbf{E}}_{u\in V^{(i)}}\left(\sum_{\substack{vw:\\ uvw\in T(G^{(i)})}} \omega_v\omega_w\right) &= \frac{1}{\sum_{v\in V^{(i)}}\omega_v}\cdot\sum_{u\in V^{(i)}}\omega_u\cdot\sum_{\substack{vw:\\ uvw\in T(G^{(i)})}}\omega_v\omega_w \\
&= \frac{3}{\sum_{v\in V^{(i)}}\omega_v}\cdot\sum_{uvw\in T(G^{(i)})}\omega_u\omega_v\omega_w \\
&\le \frac{3}{\sum_{v\in V^{(i)}}\omega_v}\cdot\sum_{uvw\in T(G^{(i)})}\omega_u x_{vw}+\omega_v x_{wu}+\omega_w x_{uv} \\
&= 3\cdot\mathop{\mathbf{E}}_{u\in V^{(i)}}\left(\sum_{\substack{vw:\\ uvw\in T(G^{(i)})}} x_{vw}\right).
\end{aligned}$$

It follows that the expected total cost is bounded by

$$\begin{aligned}
\mathop{\mathbf{E}}_{u_0,\dots,u_{t-1}}\left(\sum_{i=0}^{t-1}\sum_{\substack{vw:\\ u_ivw\in T(G^{(i)})}}\omega_v\omega_w\right) &\le 3\cdot\mathop{\mathbf{E}}_{u_0,\dots,u_{t-1}}\left(\sum_{i=0}^{t-1}\sum_{\substack{vw:\\ u_ivw\in T(G^{(i)})}} x_{vw}\right) \\
&\le 3\cdot\sum_{vw\in\binom{V}{2}} x_{vw} \\
&\le 3\cdot\mathrm{OPT}^{(\mathrm{LP})} \\
&\le 3\cdot\mathrm{OPT}^{(\mathrm{NWCC})},
\end{aligned}$$

where we used the same arguments as in Lemma 27. In particular, we used that for *any* choice of pivot nodes every pair vw appears in the sum at most once and we can therefore drop the expectation. ◀

Proof of Theorem 5. By Lemma 29, Algorithm 6 correctly approximates Node-Weighted Correlation Clustering within a factor 3, in expectation.

To bound the running time of the algorithm, first recall that by Lemma 28 the total time to initialize and sample from the weighted sampling data structure is $O(n)$ with high probability $1-1/\operatorname{poly}(n)$. The time to construct the clustering is $O(n+m)$ as any edge in the graph is touched exactly once. ◀

We remark that this randomized algorithm can in fact be seen as a reduction from Node-Weighted Correlation Clustering to Constrained Correlation Clustering: For any node u, construct a supernode containing ω_u many new nodes (all of which are joined by friendliness constraints). Then there is a one-to-one correspondence between node-weighted and constrained correlation clusters of the same cost. The constructed instance's size is proportional to the sum of weights and can thus be much larger than the original instance's size; however we can use our weighted sampling data structure to efficiently sample from it anyways.

Weighted Sampling. We finally provide a proof of Lemma 28. It heavily relies on Walker's Alias Method [41, 40], which we summarize in the following theorem:

▶ **Theorem 30** (Alias Method [41, 40]). *Let A be a set of n objects with weights $\{\omega_a\}_{a\in A}$. In time $O(n)$ we can preprocess A, and then sample $a \in A$ with probability $\omega_a / \sum_{a'\in A}\omega_{a'}$ in constant time.*

We start with the description of our data structure. Throughout, we partition the objects in A into buckets $B_1, \dots, B_\ell$, such that the i-th bucket B_i contains objects with weights in $[2^{i-1}, 2^i)$. Assuming that each weight is bounded by $\text{poly}(n)$, the number of buckets is bounded by $\ell = O(\log n)$. For every bucket $i = 1, \dots, \ell$, we maintain an *initial weight* p_i and an *actual weight* q_i. Initially, we set $p_i, q_i \leftarrow \sum_{s\in B_i} w(s_i)$. Moreover, using the Alias Method we preprocess in $O(\log n)$ time the set of buckets $\{B_1, \dots, B_\ell\}$ with their associated initial weights p_i. Additionally, using the Alias Method we preprocess each individual bucket B_i with associated weights $\{\omega_a\}_{a\in B_i}$.

Next, we describe how to implement the supported operations:

- Remove(a): Let a be contained in the i-th bucket B_i. We remove a from B_i, and update the actual weight $q_i \leftarrow q_i - \omega_a$ (but not the initial weight p_i). If $q_i < p_i/2$, then we recompute $p_i \leftarrow \sum_{a\in B_i}\omega_a$ and we recompute the Alias Method both on B_i as well as on the set of buckets (based on their initial weights p_i).
- Sample(): We sample a bucket $i = 1, \dots, \ell$ with probability proportional to its initial weight p_i using the Alias Method. With probability q_i/p_i we *accept* the bucket, otherwise we *reject* and sample a new bucket.

  Next, using the preprocessed Alias Method we sample an element a from B_i. If the element is no longer contained in B_i (as it was removed in the meantime), we repeat and sample a new element. As soon as an element a is found we report a and call Remove(a).

First, in Lemma 31 we argue for the correctness of the data structure. Then in Lemmas 32–34 we analyze the total running time.

▶ **Lemma 31.** Sample() *correctly returns an element a with probability $\omega_a / \sum_{a'\in A}\omega_{a'}$.*

Proof. The statement is proven in two steps. First, we show that every bucket B_i is selected with probability $\sum_{a\in B_i}\omega_a / \sum_{a'\in A}\omega_{a'}$. Indeed, it is easy to check that we invariantly have $q_i = \sum_{a\in B_i}\omega_a$. Moreover, every bucket B_i is sampled with probability $p_i / \sum_{a'\in A}\omega_{a'}$ by the Alias Method. Since we accept every bucket with probability q_i/p_i (and resample otherwise), the probability of accepting B_i is indeed

$$\frac{p_i}{\sum_{a'\in A}\omega_{a'}} \cdot \frac{q_i}{p_i} = \frac{q_i}{\sum_{a'\in A}\omega_{a'}} = \frac{\sum_{a\in B_i}\omega_a}{\sum_{a'\in A}\omega_{a'}}.$$

Second, assume that we accepted B_i and continue sampling $a \in B_i$. Since we resample whenever a non-existing element is returned, each existing element $a \in B_i$ is sampled with probability exactly ω_a/q_i. By combining both steps, we obtain the claim. ◀

▶ **Lemma 32.** *The preprocessing time is bounded by $O(n)$.*

Proof. The computation of p_i and q_i takes time $O(n)$. Moreover, to initialize the Alias Method on the buckets takes time $O(\log n)$, and to initialize the Alias Method on an individual bucket B_i takes time $O(|B_i|)$. Thus, the total preprocessing time is bounded by $O(n + \sum_i |B_i|) = O(n)$. ◀

▶ **Lemma 33.** *The total time of all* Remove($\cdot$) *operations is bounded by $O(n)$.*

Proof. For a given element a, we can find its bucket B_i and update the actual weight q_i of that bucket in constant time. Since there are at most n operations, this amounts to time $O(n)$. It remains to bound the time to reconstruct the Alias Method data structures.

We are left to argue about the total time of rebuilding. We only rebuild the structure of a single bucket if its actual weight dropped in half since the last rebuilding. As the objects in a bucket have weights that are within a factor 2, we have removed at least a fraction of $\frac{1}{3}$ objects in the bucket since the last rebuilding. Thus, the total number of reconstructions of a bucket B_i is $O(\log |B_i|) = O(\log n)$ and the total time for these reconstructions is bounded by $\sum_{k=0}^{\infty}(\frac{2}{3})^k|B_i| = O(|B_i|)$. Summing over all buckets, the total reconstruction time is bounded by $O(n)$.

We also rebuild the Alias Method structure on the set of buckets each time a bucket is rebuilt. Since there are only $O(\log n)$ buckets, each of which is rebuilt at most $O(\log n)$ times with running time $O(\log n)$, the total time for this part is bounded by $O(\log^3 n)$. ◀

▶ **Lemma 34.** *The total time of all* Sample() *operations is bounded by* $O(n)$ *with probability* $1 - \frac{1}{n^c}$*, for any constant* $c > 0$*.*

Proof. Consider a single execution of Sample(). Since at any point during the lifetime of the data structure we have that $q_i > \frac{p_i}{2}$, we accept the sampled bucket with probability at least $\frac{1}{2}$. Moreover, by the same argument we find an existing element in that bucket with probability $\frac{1}{2}$ as well. The number of repetitions of both of these random experiments can be modeled by a geometric random variable with constant expectation. Hence, using a standard concentration bound for the sum of independent geometric random variables [31], the total number repetitions across the at most n executions of Sample() is bounded by $O(n)$ with probability at most $1 - \frac{1}{n^c}$, for any constant $c > 0$. ◀

In combination, Lemmas 31–34 prove the correctness of Lemma 28.

Non-Integer Weights. Throughout we assumed that the weights are integers, but what if the weights are instead rationals (or reals in an appropriate model of computation)? Our deterministic algorithm uses the weights only in the solution of the LP and is therefore unaffected by the change. We remark that our randomized algorithm can also be adapted to 3-approximate Node-Weighted Correlation Clustering with the same running time even if rational weights were allowed by adapting the weighted sampling structure.