



PhD Thesis

Graphs, Algorithms, and Discrete Probabilities

Peter Michael Reichstein Rasmussen

Supervisor

Mikkel Thorup

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

Submission date: September 3, 2021.

Abstract

In this thesis, we prove new results within algorithms, graph theory, and cryptography. The thesis is divided into three topics covering two research papers each.

Hashing Algorithms: We say that a hash function is *strongly concentrated* if it provides Chernoff-style concentration bounds on hash based sums, similar to a fully random hash function. A string of research in tabulation based hashing, Pătraşcu and Thorup [JACM'12, SODA'13] and Thorup [FOCS'13], has led to several strongly concentrated hash functions. However, the hash functions are either slow in practise or only strongly concentrated when the expectation of the hash sum is sufficiently low. We introduce the first hashing schemes overcoming both these obstacles, *tabulation-permutation* and *tabulation-1permutation* hashing. Both hashing schemes are easy to implement, very fast in practice, and strongly concentrated without restrictions. Applying tabulation-1permutation hashing to streaming algorithms, in particular to the problems of counting distinct elements and estimating set similarity, we show that it theoretically offers significant speedups. For the problem of counting distinct elements, these speedups are verified experimentally.

Spectral Graph Theory: First, we prove that for any graph G on n vertices of maximum degree Δ , the multiplicity of the second eigenvalue of the normalised adjacency matrix of G is $O(n\Delta^{7/5}/\log^{1/5} n)$. This improves upon a result by Jiang et al. To obtain this bound on the eigenvalue multiplicity, we establish a result of independent interest. We show that the expected support of a random closed walk on G of length $2k < n$ is $\Omega(k^{1/5})$. Before this work, no such bound was known. Second, we show that *split-state non-malleable codes*, a cryptographic primitive, can be derived from a spectral expander graph in a very natural way. Previously, the only known constructions of split-state non-malleable codes relied on two-source randomness extractors, an arguably more esoteric object.

k-Edge Connectivity: For $k = 1, 2$, we provide an optimal randomised algorithm for decremental k -edge connectivity for graphs on n vertices and $m = \Omega(n \text{ polylog } n)$ edges. This improves upon the algorithm of Thorup [JACM'99], which is only optimal for dense graphs with $m = \Omega(n^2)$ edges. Furthermore, we improve the update time whenever $k = \log^{o(1)} n$. As a bonus, the algorithm for $k = 2$ immediately yields an optimal Las Vegas algorithm for the unique perfect matching problem for graphs on $m = \Omega(n \text{ polylog } n)$ edges. In a related paper, we prove a graph theoretic result bounding the number of k -edge connected components in a graph of high minimum degree. Roughly, we show that for an integer $k > 1$ and $\varepsilon > 0$, the maximum number of k -edge connected components of a graph on n vertices with minimum degree $(2 + \varepsilon)(k - 1)$ is $\Theta\left(\frac{n}{\varepsilon k}\right)$.

Resumé

I denne afhandling viser vi nye resultater indenfor algoritmer, grafteori og kryptografi. Afhandlingen er inddelt i tre emner, som hver dækker to forskningsartikler.

Hashing Algoritmer: Vi kalder en hash funktion *stærkt koncentreret* hvis den opfylder Chernoff-lignende koncentrationsuligheder for hash baserede summer, lig en fuldt tilfældig hash funktion. Tidligere forskning i tabuleringsbaseret hashing, Pătraşcu og Thorup [JACM'12, SODA'13] og Thorup [FOCS'13], har ledt til flere stærkt koncentrerede hash funktioner. Disse hash funktioner lider dog enten under kun at være stærkt koncentrerede, når man restringerer forventningsværdien af hash summen, eller under at være langsomme i praksis. Vi introducerer de første hash funktioner, som klarer begge disse udfordringer, *tabulation-permutation* og *tabulation-1permutation*. Begge hash funktioner er simple at implementere, meget hurtige i praksis og stærkt koncentrerede uden forbehold. Vi anvender tabulation-1permutation hash funktionen i streaming algoritmer, i særdeleshed til at tælle antal forskellige elementer i en datastrøm og til estimering af mængdelighed. Vi viser at dette giver væsentligt hurtigere køretider. Dette verificeres også eksperimentelt ved implementation af algoritmer til at tælle antal forskellige elementer i en datastrøm.

Spektral Grafteori: Først viser vi, at enhver graf G på n knuder med maksimal valens Δ opfylder at multipliciteten af den anden egenværdi af dens normaliserede nabomatrix er $\tilde{O}(n\Delta^{7/5}/\log^{1/5} n)$. Dette forbedrer et resultat af Jiang et al. For at vise denne ulighed om egenværdimultiplicitet, viser vi et resultat af uafhængig interesse. Vi beviser at den forventede størrelse af støtten af en tilfældig vandring på G af længde $2k$ er $\Omega(k^{1/5})$. Før dette fandtes ingen lignende nedre grænse. Dernæst viser vi at en delt-tilstands ikke-formbar kode på naturlig vis kan fremstilles fra en spektral udvidelsesgraf. Tidligere var alle kendte konstruktioner af delt-tilstands ikke-formbar koder baserede på to-kildes tilfældighedsudvindere, et noget mere esoterisk objekt.

k -kantforbundethed: For $k = 1, 2$ præsenterer vi en optimal randomiseret algoritme til dekrementel k -kantforbundethed for grafer på n knuder og $m = \Omega(n \text{ polylog } n)$ kanter. Dette forbedrer et resultat af Thorup [JACM'99], hvis algoritme kun er optimal for tætte grafer med $m = \Omega(n^2)$ kanter. Videre forbedrer vi opdateringstiden for $k = \log^{o(1)} n$. Som en bonus giver algoritmen for $k = 2$ en optimal Las Vegas algoritme til unik perfekt matching problemet for grafer med $m = \Omega(n \text{ polylog } n)$ kanter. I en relateret artikel viser vi et grafteoretisk resultat, som estimerer antallet af k -kantforbundne komponenter i en graf med høj minimumsvalens. Groft sagt viser vi at for et heltal $k > 1$ og $\varepsilon > 0$ er det maksimale antal k -kantforbundne komponenter af en graf på n knuder med minimumsvalens $(2 + \varepsilon)(k - 1)$ givet ved $\Theta\left(\frac{n}{\varepsilon k}\right)$.

Preface

The PhD programme is a roller coaster, both of emotion and science. In the world of research, predicting where a project will go or which subfield will provide the next promising opportunity is an almost impossible task. Acknowledging this, the initial plan for this PhD project included neither specific problems nor specific subfields to be explored. Instead, I have focused on methods and themes playing to my strengths, finding problems to work on in areas as far apart as mathematical graph theory, graph algorithms, cryptography, computational geometry, hashing, and streaming algorithms. Main threads for the research completed is that the results obtained rely on discrete methods, focused either in discrete probabilities, graph theory, or combinatorics. Furthermore, the research has often either been directly concerned with algorithms or applied an algorithmic mindset.

Four categories more or less cover the PhD project. Here, they are presented in something resembling chronological order. The papers mentioned are listed at the end of the preface in the order they appear.

Hashing Algorithms. My first major work as a PhD student at BARC was on the subject of fast hashing schemes satisfying Chernoff-style concentration bounds. I worked closely with Anders Aamand, Jakob B. T. Knudsen, and my supervisor Professor Mikkel Thorup on this. In the paper *Fast hashing with Strong Concentration Bounds* [AKK⁺20], we formalise the notion of *strong concentration* of a hash function and introduce two constant-time hashing scheme, *tabulation-permutation hashing* and *tabulation-1permutation hashing*. The hashing schemes are both strongly concentrated and fast enough to compete with some of the fastest hash functions in use, the only known hash functions in that speed range with such guarantees.

Having obtained fast, strongly concentrated hashing schemes, the natural work to follow it up explores the consequences of applying them in the implementation of popular streaming algorithms. In the paper *No Repetition: Fast Streaming with Highly Concentrated Hashing* [ADK⁺21], we show theoretically and experimentally that implementing algorithms for the problems of distinct elements and set similarity with our new tabulation-1permutation hashing scheme result in significant speedups.

Spectral Graph Theory. A passion of mine during the PhD programme has been the theory of graph spectre. Spectral graph theory is the fascinating observation that many properties of a graph are reflected in the spectrum of its adjacency matrix.

An especially noteworthy concept of spectral graph theory is the notion of expander graphs, graphs that are well-connected and hard to “disconnect”. I was allowed the opportunity to visit and work with my former advisor at UCLA, Professor Amit Sahai, on a project relating cryptography and expander graphs. In the paper *Expander Graphs are Non-Malleable Codes* [RS20], we prove a close connection between expander graphs and non-malleable codes, an information-theoretical cryptographic primitive. We find a natural way to produce a coding scheme from a

graph and show that if the graph is a sufficiently good spectral expander, the resulting coding scheme is non-malleable.

In spectral graph theory, the second eigenvalue of the normalised adjacency matrix of a graph is of particular interest since it carries a close connection to the expansion of the graph. On another stay abroad at University of California Berkeley, I visited Professor Nikhil Srivastava and worked with him and Theo McKenzie on bounding the multiplicity of the second eigenvalue of graphs. In the paper *Support of Closed Walks and Second Eigenvalue Multiplicity of Graphs* [MRS21], we prove a bound stronger than what was previously known. To do so, we tackle a problem of independent interest. We bound from below the support of closed random walks on graphs using methods from the theory of electrical flows. Before this paper, no such bound was known.

***k*-Edge Connectivity.** In connection with writing my master’s thesis, my advisor Mikkel Thorup suggested that I work with Anders Aamand on graph sampling. With collaborators, Mikkel had some ideas towards an algorithm for the problem of decremental connectivity. These ideas never quite panned out, but nonetheless, we arrived at an optimal algorithm for decremental connectivity on graph with n vertices and $m = \tilde{\Omega}(n^{3/2})$ edges. This, along with other observations made during the period, became my master’s thesis. Afterwards, we found a way to use the ideas from the first result recursively. In the paper *Optimal Decremental Connectivity in Non-Sparse Graphs* [AKL⁺21], we present an optimal algorithm for decremental k -edge connectivity for graphs on n vertices and $m = \Omega(n \text{ polylog } n)$ edges for $k = 1, 2$. For $k = \log^{o(1)} n$, we get constant update time and $n^{o(1)}$ query time whenever $m = n^{3/2+o(1)}$. Furthermore, results in pure graph theory from the master’s thesis became a manuscript of its own. In the paper *k-Edge Connected Components and Minimum Degree* [ART21], we bound from above the number of k -edge connected components a graph can contain as a function of its minimum degree.

Computational Geometry. During my time at BARC, I have had many interesting and fun side project in computational geometry courtesy of my wonderfully imaginative colleague Mikkel Abrahamsen. In most cases, these projects have been collaborations with quite a few participants. Two manuscripts resulted from this effort. First, in the paper *Classifying Convex Bodies by Their Contact and Intersection Graphs* [AAKR21], we consider intersection, unit distance, and contact graphs of convex bodies in the Euclidian plane. We shall only discuss intersection graphs here. Given a convex body, \mathcal{A} , the intersection graph of a number of translations of \mathcal{A} in the plane is the graph consisting of a vertex for each copy of \mathcal{A} and edges connecting every pair of vertices satisfying that their corresponding copies of \mathcal{A} intersect. We find a necessary and sufficient condition for convex bodies \mathcal{A} and \mathcal{B} to produce the same intersection (as well as unit distance and contact) graphs. Second, in the paper *Tiling with Squares and Packing Dominos in Polynomial Time* [AAAR20], we find an algorithm for tiling polyominoes with 2×2 squares and an algorithm for packing polyominoes with 2×1 dominos, both algorithms running in polynomial time in the size of the representation of the polyomino as a set of corners.

Papers

In accordance with the guidelines of the PhD School of the University of Copenhagen, the thesis is presented as a synopsis of a subset of the papers listed below. Together they present most of the research output produced over the course of the PhD programme. In the body of the thesis, a description of each of the selected papers is presented, and in the appendix, the papers are included in full. Note that in some cases, the manuscripts in the appendix are not identical with

the versions published.

Fast Hashing with Strong Concentration Bounds [AKK⁺20].

Anders Aamand, Jakob B. T. Knudsen, Mathias B. T. Knudsen, Peter M. R. Rasmussen, and Mikkel Thorup.
STOC 2020.

No Repetition: Fast Streaming with Highly Concentrated Hashing [ADK⁺21].

Anders Aamand, Debarati Das, Evangelos Kipouridis, Jakob B. T. Knudsen, Peter M. R. Rasmussen, and Mikkel Thorup.
Manuscript, 2020.

Expander Graphs are Non-Malleable Codes [RS20].

Peter M. R. Rasmussen and Amit Sahai.
ITC 2020.

Support of Closed Walks and Second Eigenvalue Multiplicity of Graphs [MRS21].

Theo McKenzie, Peter M. R. Rasmussen, and Nikhil Srivastava.
STOC 2021.

Optimal Decremental Connectivity in Non-Sparse Graphs [AKL⁺21].

Anders Aamand, Adam Karczmarzand, Jakub Łącki, Nikos Parotsidis, Peter M. R. Rasmussen, and Mikkel Thorup.
Manuscript, 2021.

k -Edge Connected Components and Minimum Degree [ART21].

Anders Aamand, Peter M. R. Rasmussen, and Mikkel Thorup.
Manuscript, 2021.

Classifying Convex Bodies by Their Contact and Intersection Graphs [AAKR21].

Anders Aamand, Mikkel Abrahamsen, Jakob B. T. Knudsen, and Peter M. R. Rasmussen.
SoCG 2021.

Tiling with Squares and Packing Dominos in Polynomial Time [AAAR20].

Anders Aamand, Mikkel Abrahamsen, Thomas D. Ahle, and Peter M. R. Rasmussen.
Manuscript, 2020.

Acknowledgements

I have been blessed with excellent company on this journey.

I wish to thank my PhD supervisor Mikkel Thorup for his inspiring enthusiasm for science and life, for his appreciation in good times, and for his support and understanding in bad times. I would also like to thank my former supervisor Amit Sahai of UCLA for his kindness and for believing in me.

To my colleagues at BARC, thank you all for creating a wonderfully quirky and fun work environment. I am very grateful for the many discussions, excursions, puzzles, and conversations we have shared.

I would like to thank my co-authors, Anders Aamand, Mikkel Abrahamsen, Thomas Dybdahl Ahle, Debarati Das, Adam Karczmarzand, Evangelos Kipouridis, Jakub Łącki, Jakob Bæk Tejs Knudsen, Mathias Bæk Tejs Knudsen, Theo McKenzie, Nikos Parotsidis, Amit Sahai, Nikhil Srivastava, and Mikkel Thorup. It has been a pleasure working with you, and I am proud of what we have accomplished.

To my friends and family, thank you for your smiles, care, and hugs throughout the PhD programme. I truly could not have done it without you.

And finally, I am grateful to Pernille for her love, support, and willingness to laugh at lame jokes, particularly in these last few months of the PhD programme.

Contents

1	Introduction	1
2	Hashing with Strong Concentration	2
2.1	Strong Concentration with Tabulation-Permutation	4
2.1.1	Previous Work	4
2.1.2	Fast, Strongly Concentrated Hashing	4
2.1.3	Experiments	7
2.2	Streaming with Strongly Concentrated Hashing	8
2.2.1	Estimators and Repetitions	8
2.2.2	Universal Hashing versus Strong Concentration	8
2.2.3	Experiments	10
2.2.4	Conclusion	12
3	Graph Spectra and Expanders	13
3.1	Random Walks and the Multiplicity of the Second Graph Eigenvalue	15
3.1.1	Our Results	15
3.1.2	Related Work and Open Problems	17
3.2	Non-Malleable Codes from Expander Graphs	18
3.2.1	Our Results	18
3.2.2	Perspectives	19
4	Edge Connectivity in Graphs	20
4.1	Maintaining k -edge Connected Components	22
4.1.1	Overview of Previous Results	22
4.1.2	Better k -Certificate and Optimal Decremental Connectivity	24
4.1.3	Discussion	25
4.2	Counting k -edge Connected Components	26
A	Fast Hashing with Strong Concentration Bounds	32
B	No Repetition: Fast Streaming with Highly Concentrated Hashing	90
C	Expander Graphs are Non-Malleable Codes	104
D	Support of Closed Walks and Second Eigenvalue Multiplicity of Graphs	115
E	Optimal Decremental Connectivity in Non-Sparse Graphs	140
F	k-Edge Connected Components and Minimum Degree	173

G	Classifying Convex Bodies by Their Contact and Intersection Graphs	182
H	Tiling with Squares and Packing Dominoes in Polynomial Time	199

Chapter 1

Introduction

In accordance with the guidelines of the PhD School of the University of Copenhagen, this thesis is presented as a synopsis of a subset of the papers produced over the course of the PhD programme. Beyond the overarching category of algorithms, two main themes categorise the selected works: connectivity properties of graphs and discrete probabilities. Most of the papers presented fall in the intersection of these themes.

Organisation. The thesis has been divided into three chapters with two papers presented in each chapter. The chapters correspond to the first three categories laid out in the preface. First, a chapter on hashing algorithms, which relies heavily on theory from discrete probability to introduce and apply new hash functions. Second, a chapter on spectral graph theory discussing random sampling of edges from expander graphs, new results on random walks on graphs, and multiplicity of graph eigenvalues. Third, a chapter on k -edge connectivity that considers the problem of decremental k -edge connectivity and explores the impact of high minimum degree on the number of distinct k -edge connected components of a graph.

Each chapter starts with an introduction to the overall topic followed by presentations of the two papers associated with the topic. The presentations discuss the results of the papers, relevant literature, and possible further research topics. It should be noted that the goal of this synopsis is to give an overview of the research and not to present a complete picture. For details and proofs, the interested reader is referred to the full manuscripts in the appendix.

Notation. Most notation will be introduced as needed. For $n \in \mathbb{N}$, we denote by $[n]$ the set $[n] = \{0, 1, \dots, n - 1\}$. To avoid confusion, it should be noted that some papers included in the appendix of this thesis do not adhere to this and instead let $[n] = \{1, 2, \dots, n\}$.

Chapter 2

Hashing with Strong Concentration

The following text is a slightly edited excerpt from the introduction of [AKK⁺20].

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

$$\Pr[X \geq (1 + \varepsilon)\mu] \leq \exp(-\mu \mathcal{C}(\varepsilon)) \quad \left[\leq \exp(-\varepsilon^2 \mu / 3) \text{ for } \varepsilon \leq 1 \right], \quad (2.1)$$

$$\Pr[X \leq (1 - \varepsilon)\mu] \leq \exp(-\mu \mathcal{C}(-\varepsilon)) \quad \left[\leq \exp(-\varepsilon^2 \mu / 2) \text{ for } \varepsilon \leq 1 \right]. \quad (2.2)$$

Here $\mathcal{C} : (-1, \infty) \rightarrow [0, \infty)$ is given by $\mathcal{C}(x) = (x + 1) \ln(x + 1) - x$, so $\exp(-\mathcal{C}(x)) = \frac{e^x}{(1+x)^{(1+x)}}$. Textbook proofs of (2.1) and (2.2) can be found in [MR95, §4]¹. Writing $\sigma^2 = \text{Var}[X]$, a more general bound is

$$\Pr[|X - \mu| \geq t] \leq 2 \exp(-\sigma^2 \mathcal{C}(t/\sigma^2)) \quad \left[\leq 2 \exp(-(t/\sigma)^2 / 3) \text{ for } t \leq \sigma^2 \right]. \quad (2.3)$$

Since $\sigma^2 \leq \mu$ and $\mathcal{C}(-\varepsilon) \leq 1.5 \mathcal{C}(\varepsilon)$ for $\varepsilon \leq 1$, (2.3) is at least as good as (2.1) and (2.2), up to constant factors, and often better. In this work, we state our results in relation to (2.3), known as Bennett's inequality [Ben62].

Hashing is another fundamental tool of randomized algorithms dating back to the 1950s [Dum56]. A random hash function, $h : U \rightarrow R$, assigns a hash value, $h(x) \in R$, to every key $x \in U$. Here both U and R are typically bounded integer ranges. The original application was hash tables with chaining where x is placed in bin $h(x)$, but today, hash functions are ubiquitous in randomized algorithms. For instance, they play a fundamental role in streaming and distributed settings where a system uses a hash function to coordinate the random choices for a given key. In most applications, we require concentration bounds for one of the following cases of increasing generality.

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

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

2. We may instead be interested in the total weight of the balls with hash values in the interval $[y_1, y_2)$ for some $y_1, y_2 \in [m]$, that is, $X = \sum_{x \in S} w_x \cdot [y_1 \leq h(x) < y_2]$.
3. More generally, we may consider a fixed *value function* $v: U \times R \rightarrow [0, 1]$. For each key $x \in U$, we define the random variable $X_x = v(x, h(x))$, where the randomness of X_x stems from that of $h(x)$. We write $X = \sum_{x \in U} v(x, h(x))$ for the sum of these values.

To exemplify applications, the first case is common when trying to allocate resources; the second case arises in streaming algorithms; and the third case handles the computation of a complicated statistic, X , on incoming data. In each case, we wish the variable X to be concentrated around its mean, $\mu = \mathbb{E}[X]$, according to the Chernoff-style bound of (2.3). If we had fully random hashing, this would indeed be the case. However, storing a fully random hash function is infeasible. In this chapter, we instead study hash functions satisfying the following definition when X is a random variable as in one of the three cases above.

Definition 2.1. [Strong Concentration] Let $h: [u] \rightarrow [m]$ be a hash function, $S \subseteq [u]$ be a set of hash keys of size $n = |S|$, and $X = X(h, S)$ be a random variable, which is completely determined by h and S . Denote by $\mu = \mathbb{E}[X]$ and $\sigma^2 = \text{Var}[X]$ the expectation and variance of X . We say that X is *strongly concentrated with added error probability* $f(u, n, m)$ if for every $t > 0$,

$$\Pr[|X - \mu| \geq t] \leq O(\exp(-\Omega(\sigma^2 \mathcal{C}(t/\sigma^2))) + f(u, n, m)). \quad (2.4)$$

Note that the expression of (2.4) differs asymptotically from (2.3) only in the term $f(u, n, m)$. Hence, satisfying the definition, h may be considered to perform as well as a fully random hash function on the three cases listed above, except with probability $f(u, n, m)$. Ideally, we would like f to be inversely polynomial in the universe size u .

The remainder of this chapter presents the papers “Fast Hashing with Strong Concentration Bounds” [AKK⁺20] and “No repetition: Fast Streaming with Highly Concentrated Hashing” [ADK⁺21]. In [AKK⁺20], we construct the first fast and practical hashing schemes which provide strong concentration for the cases listed above, *tabulation-permutation* and *tabulation-1permutation* hashing. In [ADK⁺21], we apply our new hashing schemes towards streaming algorithms.

2.1 Strong Concentration with Tabulation-Permutation

In this section, we present the paper “Fast Hashing with Strong Concentration Bounds” [AKK⁺20]. First, we review previous work on constructing hashing with strong concentration as per Definition 2.1. Then, we present our new hashing schemes, our results, and some experiments. The section is composed of edited excerpts from the introduction of [AKK⁺20].

2.1.1 Previous Work

One way to achieve Chernoff-style bounds in the three use cases listed in the chapter introduction is through the classic k -independent hashing framework of Wegman and Carter [WC81]. The random hash function $h : U \rightarrow R$ is k -independent if for any k distinct keys $x_1, \dots, x_k \in U$, $(h(x_1), \dots, h(x_k))$ is uniformly distributed in R^k . Schmidt, Siegel, and Srinivasan [SSS95] have shown that with k -independence, the above Chernoff bounds hold with an added error probability decreasing exponentially in k . Unfortunately, a lower bound by Siegel [Sie04] implies that evaluating a k -independent hash function takes $\Omega(k)$ time unless we use an infeasible amount of space.

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

Thorup’s double tabulation [Tho13] is another constant-time tabulation based hashing scheme, which achieves highly independent hashing except for a small error probability. It is the main constant-time competitor of our new tabulation-permutation hashing. However, due to the large size of the constants involved, double tabulation is not competitive on speed in practice. In fact, double tabulation is 30 times slower than tabulation-permutation in our experiments.

In [AKK⁺20], we construct and analyse a new family of fast constant-time hash functions *tabulation-permutation* hashing that satisfy Chernoff-style concentration bounds like (2.3) without any restrictions on μ . Our bounds hold for all of the cases described above and all possible inputs. Furthermore, tabulation-permutation hashing is an order of magnitude faster than any other known hash function with similar concentration bounds, and almost as fast as simple and twisted tabulation hashing. We demonstrate this both theoretically and experimentally.

2.1.2 Fast, Strongly Concentrated Hashing

In [AKK⁺20], we prove new results regarding three hashing schemes, simple tabulation, tabulation-permutation, and tabulation-1permutation. The latter two are based on simple tabulation and introduced for the first time in [AKK⁺20]. Here, we shall describe the hash functions and our results.

Simple Tabulation *Simple tabulation* hashing dates back to Zobrist [Zob90]. In simple tabulation hashing, we consider the key domain U to be of the form $U = \Sigma^c$ for some character alphabet Σ and $c = O(1)$, such that each key consists of c characters of Σ . Let $m = 2^\ell$

be given and identify $[m] = \{0, 1, \dots, m - 1\}$ with $[2]^\ell$. A simple tabulation hash function, $h: \Sigma^c \rightarrow [m]$, is then defined as follows. For each $j \in \{1, \dots, c\}$ store a fully random character table $h_j: \Sigma \rightarrow [m]$ mapping characters of the alphabet Σ to ℓ -bit hash values. To evaluate h on a key $x = (x_1, \dots, x_c) \in \Sigma^c$, we compute $h(x) = h_1(x_1) \oplus \dots \oplus h_c(x_c)$, where \oplus denotes bitwise XOR – an extremely fast operation. With character tables in cache, this scheme is the fastest known 3-independent hashing scheme [PT12]. We will denote by $u = |U|$ the size of the key domain, identify $U = \Sigma^c$ with $[u]$, and always assume the size of the alphabet, $|\Sigma|$, to be a power of two. For instance, we could consider 32-bit keys consisting of four 8-bit characters.

Let $S \subseteq U$ and consider hashing $n = |S|$ weighted balls or keys into $m = 2^\ell$ bins using a simple tabulation function, $h: [u] \rightarrow [m]$, in line with the first case mentioned above. In [AKK⁺20, Section 4], we prove the following.

Theorem 2.2. *Let $h: [u] \rightarrow [m]$ be a simple tabulation hash function with $[u] = \Sigma^c$, $c = O(1)$. Let $S \subseteq [u]$ be given of size $n = |S|$ and assign to each key/ball $x \in S$ a weight $w_x \in [0, 1]$. Let $y \in [m]$, and define $X = \sum_{x \in S} w_x \cdot [h(x) = y]$ to be the total weight of the balls hashing to bin y . Then for any constant $\gamma > 0$, X is strongly concentrated with added error probability n/m^γ , where the constants of the asymptotics are determined solely by c and γ .*

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

However, for small m the error probability n/m^γ is prohibitive. For instance, unbiased coin tossing, corresponding to the case $m = 2$, has an added error probability of $n/2^\gamma$ which is useless. In [AKK⁺20, Section 8], we show that it is impossible to get good concentration bounds using simple tabulation hashing when the number of bins m is small. To handle all instances, including those with few bins, and to support much more general Chernoff bounds, we introduce a new hash function: tabulation-permutation hashing.

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

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

Theorem 2.3. *Let $h: [u] \rightarrow [r]$ be a tabulation-permutation hash function with $[u] = \Sigma^c$ and $[r] = \Sigma^d$, $c, d = O(1)$. Let $v: [u] \times [r] \rightarrow [0, 1]$ be a fixed value function that to each key $x \in [u]$ assigns a value $X_x = v(x, h(x)) \in [0, 1]$ depending on the hash value $h(x)$ and define $X = \sum_{x \in [u]} X_x$. For any constant $\gamma > 0$, X is strongly concentrated with added error probability $1/u^\gamma$, where the constants of the asymptotics are determined solely by c , d , and γ .*

Tabulation-permutation hashing allows us to hash into m bins for any $m \in \mathbb{N}$ (not necessarily a power of two) preserving the strong concentration from Theorem 2.3. To do so, simply define

the hash function $h^m: [u] \rightarrow [m]$ by $h^m(x) = h(x) \bmod m$. Relating back to Theorem 2.2, consider a set $S \subseteq U$ of n balls where each ball $x \in S$ has a weight $w_x \in [0, 1]$ and balls x outside S are defined to have weight $w_x = 0$. To measure the total weight of the balls landing in a given bin $y \in [m]$, we define the value function $v(x, z) = w_x \cdot [z \bmod m = y]$. Then

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

is exactly the desired quantity and we get the concentration bound from Theorem 2.3. Then the big advantage of tabulation-permutation hashing over simple tabulation hashing is that it reduces the added error probability from n/m^γ of Theorem 2.2 to the $1/u^\gamma$ of Theorem 2.3, where u is the size of the key universe. Thus, with tabulation-permutation hashing, we actually get Chernoff bounds with high probability regardless of the number of bins.

Pătraşcu and Thorup [PT13] introduced twisted tabulation that like our tabulation-permutation achieved Chernoff-style concentration bounds with a general value function v . Their bounds are equivalent to those of Theorem 2.3, but only under the restriction $\mu \leq |\Sigma|^{1-\Omega(1)}$. To understand how serious this restriction is, consider again tossing an unbiased coin for each key x in a set $S \subseteq [u]$, corresponding to the case $m = 2$ and $\mu = |S|/2$. With the restriction from [PT13], we can only handle $|S| \leq 2|\Sigma|^{1-\Omega(1)}$, but recall that Σ is chosen small enough for character tables to fit in fast cache, so this rules out any moderately large data set. In [AKK⁺20, Section 8] we show that for certain sets S , twisted tabulation has the same problems as simple tabulation when hashing to few bins. This implies that the restrictions from [PT13] cannot be lifted with a better analysis.

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

More precisely, a tabulation-1permutation hash function $h: \Sigma^c \rightarrow \Sigma^d$ is a composition, $h = \tau \circ g$, of a simple tabulation function, $g: \Sigma^c \rightarrow \Sigma^d$, and a random permutation, $\tau: \Sigma^d \rightarrow \Sigma^d$, of the most significant character, $\tau(z_1, \dots, z_d) = (\tau_1(z_1), z_2, \dots, z_d)$ for a random character permutation $\tau_1: \Sigma \rightarrow \Sigma$. This scheme, needing only $c + 1$ character lookups, is powerful enough for concentration within an arbitrary interval.

Theorem 2.4. *Let $h: [u] \rightarrow [r]$ be a tabulation-1permutation hash function with $[u] = \Sigma^c$ and $[r] = \Sigma^d$, $c, d = O(1)$. Consider a key/ball set $S \subseteq [u]$ of size $n = |S|$ where each ball $x \in S$ is assigned a weight $w_x \in [0, 1]$. Choose arbitrary hash values $y_1, y_2 \in [r]$ with $y_1 \leq y_2$. Define $X = \sum_{x \in S} w_x \cdot [y_1 \leq h(x) < y_2]$ to be the total weight of balls hashing to the interval $[y_1, y_2)$. Then for any constant $\gamma > 0$, X is strongly concentrated with added error probability $1/u^\gamma$, where the constants of the asymptotics are determined solely by c , d , and γ .*

One application of Theorem 2.4 is in the following sampling scenario: We set $y_1 = 0$, and sample all keys with $h(x) < y_2$. Each key is then sampled with probability y_2/r , and Theorem 2.4 gives concentration on the number of samples. In [ADK⁺21], to be discussed in the next section, this is used for more efficient implementations of streaming algorithms.

2.1.3 Experiments

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

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

The findings of the experiments are presented in Table 2.1. For further details, see the full manuscript in the appendix.

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

Table 2.1: The time for different hash functions to hash 10^7 keys of length 32 bits and 64 bits, respectively, to ranges of size 32 bits and 64 bits. The experiment was carried out on two computers. The hash functions written in italics are those without general Chernoff-style bounds. Hash functions written in bold are the contributions of this paper. The hash functions in regular font are known to provide Chernoff-style bounds. Note that we were unable to implement double tabulation from 64 bits to 64 bits since the hash tables were too large to fit in memory.

2.2 Streaming with Strongly Concentrated Hashing

In this section, we present the paper “No Repetition: Fast Streaming with Highly Concentrated Hashing” [ADK⁺21]. The section is composed of edited excerpts from [ADK⁺21]. More specifically, Section 2.2.1 is an edited excerpt of Section 1 of [ADK⁺21]; Section 2.2.2 consists of edited excerpts of Sections 2 and 3 of [ADK⁺21]; Section 2.2.3 consists of edited excerpts of Section 5 of [ADK⁺21]; and Section 2.2.4 is an excerpt of Section 6 of [ADK⁺21].

2.2.1 Estimators and Repetitions

Recent years have brought with them a huge demand for algorithms that can process and compute statistics on large streams of data. Common statistics of interest include the number of distinct elements of a stream and the set similarity between two large sets. The sheer volume of the data makes storing a complete copy of the stream and performing exact computations an impossible task, and so, if the data is not processed in time, the information is lost. To solve this problem, we therefore have to resort to estimation algorithms. For instance, instead of precisely counting the number of unique visitors to a website, we may settle for a good estimate. In simple terms, the goal of these estimation algorithms are as follows: For a data stream S and some statistic $\mathcal{F} = \mathcal{F}(S)$, we want an estimator $\hat{\mathcal{F}}$ such that $\hat{\mathcal{F}} \in (1 \pm \epsilon)\mathcal{F}$ with some high probability at least $1 - \delta$.

To get such an estimator, a common strategy is to design one that works with constant probability, and then boost the probability using independent repetitions. In [ADK⁺21], our running example of such an algorithm is the classical algorithm of Bar-Yossef et al. [BYJK⁺02] to estimate the number of distinct elements in a stream. Using strongly universal hashing to process each element, we obtain an estimator such that the probability of getting too large an error is at most a constant, e.g., $1/4$. By performing r independent repetitions and taking the median of the estimators, the error probability falls exponentially in r . However, running r independent experiments increases the processing time by a factor r .

In [ADK⁺21], we make the point that with a hash function with strong concentration bounds at hand, such as the tabulation-permutation or tabulation-1permutation hashing schemes of the previous section, we can get the same high probability bounds without any need for repetitions. Instead of r independent sketches, we have a single sketch that is $\Theta(r)$ times bigger, so the total space is essentially the same. However, we only apply a single hash function, processing each element in constant time regardless of r , and the overall algorithms just get simpler. The idea is generic and can be applied to other algorithms.

2.2.2 Universal Hashing versus Strong Concentration

In this presentation, we shall not focus too closely on implementing algorithms for counting distinct elements or set similarity, which is done in [ADK⁺21]. For such details, see the full manuscript in the appendix. Instead, we shall describe how hashing with strong concentration can be applied in general.

In [ADK⁺21], we consider a very common situation, where the crucial component of an algorithm is a random hash function $h: U \rightarrow [0, 1)$ mapping some key universe U , e.g. 64-bit keys, uniformly into $R = [0, 1)$. The application is the following. Let $S \subset U$ be a set of keys and $p \in [0, 1)$ a threshold value. We wish to compute the number X of keys of S that hash below p , i.e.

$$X = |\{s \in S \mid h(s) < p\}|.$$

Here p could be an unknown function of S , but p should be independent of the random hash function h . Since the mean μ of x is $\mathbb{E}[X] = |S|p$, we may estimate the size of S by X/p with precision increasing in the concentration of X around its mean. In particular, we are interested in the probability δ that X deviates from μ by more than a factor $\varepsilon > 0$, i.e., the probability $\delta = \Pr[|X - \mu| \geq \varepsilon\mu]$.

If the hash function h is fully random, we get the classic Chernoff concentration bounds on X (see, e.g, [MR95]):

$$\Pr[X \geq (1 + \varepsilon)\mu] \leq \exp(-\varepsilon^2\mu/3) \text{ for } 0 \leq \varepsilon \leq 1, \quad (2.5)$$

$$\Pr[X \leq (1 - \varepsilon)\mu] \leq \exp(-\varepsilon^2\mu/2) \text{ for } 0 \leq \varepsilon \leq 1. \quad (2.6)$$

Unfortunately, we cannot implement fully random hash functions as it requires space as big as the universe.

Strongly Universal Hashing. To get something implementable in practice, Wegman and Carter [WC81] proposed strongly universal hashing.

Definition 2.5. [Strongly Universal Hashing] A hash function $h: U \rightarrow R$ is *strongly universal* if for every pair of distinct keys $x, y \in U$, the distribution of $(h(x), h(y))$ is uniform on R^2 .

Many common hash functions are strongly universal. The textbook example of a strongly universal hash function hashing into $[0, 1)$ is the Multiply-Mod-Prime hash function [CW79]. Also worth mentioning is the Multiply-Shift hash function [DM92].

Assuming we have a strongly universal hash function $h: U \rightarrow [0, 1)$, we again let X be the number of elements from S that hash below p . Then $\mu = \mathbb{E}[X] = |S|p$ and because the hash values are 2-independent, we have $\text{Var}[X] \leq \mathbb{E}[X] = \mu$. Therefore, by Chebyshev's inequality,

$$\Pr[|X - \mu| \geq \varepsilon\mu] \leq 1/(\varepsilon^2\mu). \quad (2.7)$$

As $\varepsilon^2\mu$ gets large, we see that the error probability of strongly universal hashing is much higher than the Chernoff bounds with fully random hashing. However, using the well-known median trick, it is still possible to guarantee high concentration by aiming for a constant error probability like $\delta = 1/4$ and then using the median over independent repetitions of the estimation to reduce the error probability to something akin to (2.5) and (2.6).

Strong Concentration with Tabulation-1Permutation. Using a hashing scheme with strong concentration as per Definition 2.1 instead, allows us to retain the beautiful concentration bounds of the fully random hash function except with a small additive error probability. Previously known hash functions such as k -wise PolyHash or double tabulation hashing are much too slow for practical applications, and hence, this solution has not been properly considered. Our introduction in [AKK⁺20] of the tabulation-1permutation hashing scheme changes this picture. As described in Section 2.1.3, tabulation-1permutation hashing exhibits speeds comparable to some of the fastest universal hash functions. At the same time, it follows by Theorem 2.4 that for every $\gamma > 1$ and $0 < \varepsilon < 1$, the estimator X implemented with tabulation-1permutation satisfies the concentration inequality

$$\begin{aligned} \Pr[|X - \mu| \geq \varepsilon\mu] &\leq 2 \exp(-\Omega(\sigma^2\mathcal{C}(\varepsilon\mu/\sigma^2))) + 1/|U|^\gamma \\ &\leq 2 \exp(-\Omega(\mu\mathcal{C}(\varepsilon))) + 1/|U|^\gamma \\ &\leq 2 \exp(-\Omega(\varepsilon^2\mu/3)) + 1/|U|^\gamma, \end{aligned}$$

where the second inequality follows by Lemma 3.4 of [AKK⁺20] and the fact that $\mu \geq \sigma^2$, and the third inequality is equivalent to (2.2). Comparing with (2.5) and (2.6), we see that up to a constant factor in the exponent and the negligible term $1/|U|^\gamma$, we get the same concentration as with fully random hashing.

2.2.3 Experiments

In [ADK⁺21], we demonstrate that our approach performs well in practice by performing experiments showing the strength of Tabulation-1Permutation for counting distinct elements in a stream. We compare with the fastest known strongly universal hash functions, Multiply-Mod-Prime [CW79] and Multiply-Shift [DHKP97], and with other commonly used hash functions such as MurmurHash3 [App16] and BLAKE3 [JOWO20]. Without the use of independent repetitions, we demonstrate that Tabulation-1Permutation provides more reliable estimates than the fast strongly universal hash functions. Moreover, the implementation with Tabulation-1Permutation is faster than when using MurmurHash3 and BLAKE3. In fact, BLAKE3 was approximately 150 times slower than Tabulation-1Permutation, so we disregard it in our experiments. On the other hand, the implementation with Tabulation-1Permutation is both faster and provides better estimates than when implementing the algorithm with the strongly universal hash functions and independent repetitions.

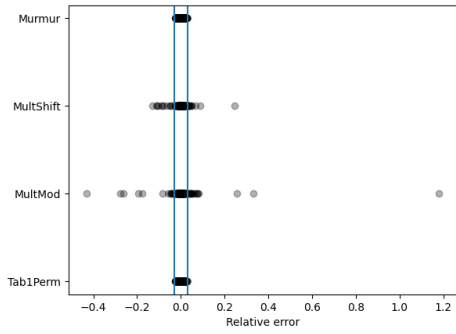
Remark. It is important to note that no amount of experiments can prove that a hashing scheme performs well on all possible data, and finding problematic data sets for a given hash family is often a non-trivial task². The results from [AKK⁺20] show that Tabulation-1Permutation performs well on *any* possible data set with high probability. In other words, if implementing the above streaming algorithms with Tabulation-1Permutation, we no longer have to cross our fingers that the data sets encountered does not have hidden structure which interacts badly with the hashing scheme. Furthermore, Tabulation-1Permutation is very fast, so this new guarantee comes with no compromise on the speed of the algorithms.

In this presentation, we consider a subset of the experiments of [ADK⁺21] that compare implementations of the bottom- k algorithm for counting distinct elements using tabulation-1permutation, Multiply-Mod-Prime, Multiply-Shift, and MurmurHash3. The comparisons are on the basis of speed and concentration. For a more comprehensive picture and other experimental setups, see the full manuscript in the appendix.

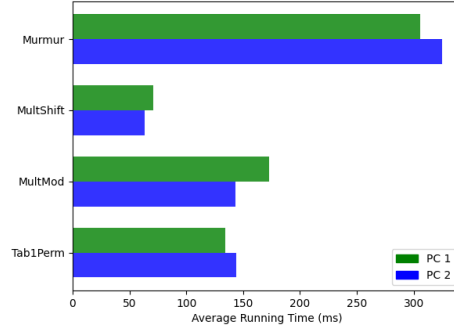
First, we consider the case where none of the algorithms use repetitions to improve their concentrations. The results are presented in Fig. 2.1. Here, we see that tabulation-1permutation is slightly slower than the Multiply-Shift hashing scheme, but by less than a factor of three. Furthermore, we see that the relative error of tabulation-1permutation stays nicely inside the accepted range of error, while the universal hash functions Multiply-Mod-Prime and Multiply-Shift have some large deviations in their estimates.

Second, we consider the case where the hash functions without strong concentration bounds apply the median trick. The results are presented in Fig. 2.2. Inspecting the results, we observe that with repetitions, the Multiply-Mod-Prime and Multiply-Shift hashing schemes achieve concentration on par with tabulation-1permutation. However, the repetitions cause them both to be significantly slower.

²For Multiply-Shift and Multiply-Mod-Prime, we have concrete examples of data sets on for which they fail. Moreover, in [Boß12] the authors provide concrete bad data sets for MurmurHash3.

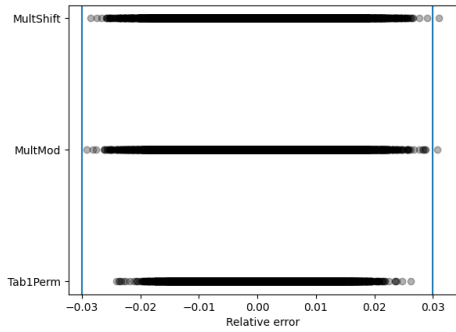


(a) Relative error of independent repetitions of bottom- k algorithm implemented with various hashing schemes applied to a structured dataset. Parameters are calibrated to aim for an error of $\varepsilon = 3\%$, marked by the blue lines. Dataset cardinality: 5×10^5
Experiments per hash-function: 5×10^4

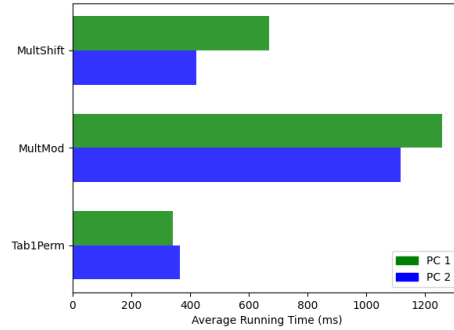


(b) Time taken by bottom- k algorithm implemented with various hashing schemes. Dataset cardinality: 5×10^7
 $k: 3500$

Figure 2.1: Relative error and timing when estimating the number of distinct elements using various hash functions on synthetic data. The experiments did not use independent repetitions.



(a) Relative error of independent repetitions of bottom- k algorithms applied to a structured dataset. Parameters are calibrated to aim for an error of $\varepsilon = 3\%$, marked by the blue lines. For Multiply-Mod-Prime and Multiply-Shift we have $(r, k) = (5, 4900)$. For Tabulation-1Permutation $(r, k) = (1, 24500)$. Dataset cardinality: 5×10^5
Experiments per hash-function: 3×10^4



(b) Time taken by bottom- k algorithms. With Multiply-Mod-Prime and Multiply-Shift, $(r, k) = (5, 700)$. With Tabulation-1Permutation, $(r, k) = (1, 3500)$
Dataset cardinality: 5×10^7

Figure 2.2: Relative error and timing when estimating the number of distinct elements using various hash functions on synthetic data. The experiments applied the median trick for Multiply-Shift and Multiply-Mod-Prime. The parameter r is the number of sketches maintained in parallel.

2.2.4 Conclusion

In [ADK⁺21], we show that the use of hash functions with strong concentration bounds, like Tabulation-1Permutation, can speed up streaming algorithms by avoiding time consuming independent repetitions, and still provide accurate statistical estimates with high probability. Specifically, we study algorithms for estimating the number of distinct elements in a stream and the similarity between two large sets.

Our results are backed up by experiments. They show that widely used hash functions like Multiply-Mod-Prime and Multiply-Shift yield inaccurate estimates when only maintaining a single sketch. When boosting the success probability of Multiply-Shift and Multiply-Mod-Prime using independent repetitions, the implementation with Tabulation-1Permutation both becomes faster and still provides better estimates. Finally, the running time of Tabulation-1Permutation is better than that of other commonly used hash functions like MurmurHash3, which provided reliable estimates in our experiments but which have no similar general theoretical guarantees.

Chapter 3

Graph Spectra and Expanders

Let $G = (V, E)$ be a multigraph on $n = |V|$ vertices. The *adjacency matrix* of G is the $n \times n$ matrix A_G with rows and columns indexed by the set V and where for any $u, v \in V$, $(A_G)_{uv}$ is the number of edges between u and v . The *spectrum* of G is the spectrum of A_G , i.e., the set of eigenvalues of A_G . The study of graph spectra, spectral graph theory, first emerged as a purely mathematical subfield of graph theory, but has since found applications in several other fields. As such it is a popular field of study within theoretical computer science. Among other things, clues to the connectivity of a graph and the behaviour of random walks on a graph can be determined from its spectrum. Both of these connections are something we shall touch upon in this chapter.

A particularly interesting topic central to graph spectra is that of expander graphs and the spectral gap. Suppose that G is connected and d -regular, meaning that each vertex of G has degree exactly $d \in \mathbb{N}$. The n eigenvalues of A_G counted with multiplicity may be ordered as $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$ with $\lambda_1 = d$ and $\lambda_1 > \lambda_2$. Two quantities of interest are the *spectral gap* of G given by $d - \lambda_2$ and the *spectral expansion* of G given by $\lambda(G) = \max_{2 \leq i \leq n} \{|\lambda_i|\} = \max\{\lambda_2, |\lambda_n|\}$. These quantities reveal properties regarding the connectivity of G in the following senses.

First, the spectral gap bounds the *Cheeger constant* $h(G)$, the largest real number satisfying that for every set of vertices $S \subset V$, at least $h(G)|S|$ edges connect S to the remaining graph. Formally,

$$h(G) = \min_{\substack{S \subset V \\ 0 < |S| \leq n/2}} \frac{|\partial S|}{|S|},$$

where ∂S denotes the edges in G from S to $V \setminus S$. The Cheeger constant of a graph is also known as its *edge expansion*. One of the iconic results of spectral graph theory relates the spectral gap to the Cheeger constant. The larger the spectral gap, the larger the Cheeger constant and the edge expansion.

Theorem 3.1 (Cheeger Inequality [Alo86], [SJ89]). *For every d -regular graph G ,*

$$\frac{d - \lambda_2(G)}{2} \leq h(G) \leq \sqrt{2d(d - \lambda_2(G))}.$$

Second, the spectral expansion $\lambda(G)$ has many useful properties. The smaller $\lambda(G)$ is compared with d , the smaller the *mixing time* of random walks on G and the faster the running time of certain randomised algorithms. Especially beautiful is the following theorem, the expander

mixing lemma, which bounds the number of edges between two subsets of vertices of G compared with the expected number of such edges in a random d -regular graph. Here, for vertex subsets $A, B \subset V$, $E(A, B)$ denotes the set of edges between A and B in G .

Theorem 3.2 (Expander Mixing Lemma [AC88]). *For every d -regular graph $G = (V, E)$ and every pair of subsets $A, B \subset V$,*

$$\left| |E(A, B)| - \frac{d|A||B|}{n} \right| \leq \lambda(G) \sqrt{|A||B|}.$$

Given vertex subsets $A, B \subset V$, the expected number of edges between A and B in a random d -regular graph on V is exactly $\frac{d|A||B|}{n}$. Thus, the expander mixing lemma can be interpreted as saying that the smaller the spectral expansion $\lambda(G)$, the closer the behaviour of G is to a random d -regular graph. This is a key feature of good expander graphs. Random graphs have many beautiful properties, but are not the static objects needed for algorithmic applications. Expander graphs exhibit many of the same properties but can have succinct, static descriptions, making them ideal for algorithmic applications.

In the sections to follow, the two papers “Support of Closed Walks and Second Eigenvalue Multiplicity of Graphs” [MRS21] and “Expander Graphs are Non-Malleable Codes” [RS20] are presented. The first explores the multiplicity of the second eigenvalue λ_2 as well as the connection between spectral graph theory and random walks on graphs. The second applies spectral expander graphs and the expander mixing lemma to construct a cryptographic primitive called a non-malleable code.

3.1 Random Walks and the Multiplicity of the Second Graph Eigenvalue

In this section, we present the paper “Support of Closed Walks and Second Eigenvalue Multiplicity of Graphs” [MRS21]. The section is composed of edited excerpts from sections 1 and 6 of [MRS21].

As noted above, the eigenvalues of matrices associated with graphs play an important role in many areas of mathematics and computer science. In their recent beautiful work on the equiangular lines problem, Jiang, Tidor, Yao, Zhang, and Zhao [JTY⁺19] proved the following novel result constraining the distribution of the adjacency eigenvalues of *all* connected graphs of sufficiently low degree.

Theorem 3.3. *If G is a connected graph of maximum degree Δ on n vertices, then the multiplicity of the second largest eigenvalue of its adjacency matrix A_G is bounded by $O(n \log \Delta / \log \log(n))$.*

For their application to equiangular lines, [JTY⁺19] only needed to show that the multiplicity of the second eigenvalue is $o(n)$, but they asked whether the $O(n / \log \log(n))$ dependence in Theorem 3.3 could be improved, noting a huge gap between this and the best known lower bound of $\Omega(n^{1/3})$ achieved by certain Cayley graphs of $\text{PSL}(2, p)$ (see [JTY⁺19, Section 4]).

Meanwhile, in the theoretical computer science community, the largest eigenvalues of the *normalized* adjacency matrix $\tilde{A}_G := D_G^{-1/2} A_G D_G^{-1/2}$ (for D_G the diagonal matrix of degrees) have received much attention over the past decade due to their relation with graph partitioning problems and the unique games conjecture (see e.g. [Kol11, BRS11, LRTV12, GT15, LOGT14, ABS15, BGH⁺15, LOG18]); in particular, many algorithmic tasks become easier on graphs with few large normalized adjacency eigenvalues. Thus, it is of interest to know how many of these eigenvalues there can be in the worst case.

3.1.1 Our Results

In [MRS21], we prove significantly stronger upper bounds than Theorem 3.3 on the second eigenvalue multiplicity for the normalized adjacency matrix. Order the eigenvalues of \tilde{A}_G as $\lambda_1(\tilde{A}_G) \geq \lambda_2(\tilde{A}_G) \geq \dots \geq \lambda_n(\tilde{A}_G)$, and let $m_G(I)$ denote the number of eigenvalues of \tilde{A}_G in an interval I .

Theorem 3.4. *If G is a connected graph of maximum degree Δ on n vertices with $\lambda_2(\tilde{A}_G) = \lambda_2$, then¹*

$$m_G \left(\left[\left(1 - \frac{\log \log \Delta n}{\log \Delta n}\right) \lambda_2, \lambda_2 \right] \right) = \tilde{O} \left(n \cdot \frac{\Delta^{7/5}}{\log^{1/5} n} \right). \quad (3.1)$$

Because of the relationship $\tilde{A}_G = \frac{1}{d} A_G$ when G is regular, (3.1) gives a substantial improvement on Theorem 3.3 in the regular case (in the non-regular case, the results are incomparable as they concern different matrices). In addition to the stronger $O(n / \text{polylog}(n))$ bound, a notable difference between our result and Theorem 3.3 is that we control the number of eigenvalues in a small interval containing λ_2 . Though we do not know whether the exponents in (3.1) are sharp, we show in [MRS21] that constant degree bipartite Ramanujan graphs have at least $\Omega(n / \log^{3/2} n)$ eigenvalues in the interval appearing in (3.1), indicating that $O(n / \text{polylog}(n))$ is the correct regime for the maximum number of eigenvalues in such an interval when Δ is constant.

¹All asymptotics are as $n \rightarrow \infty$ and the notation $\tilde{O}(\cdot)$ suppresses polyloglog(n) terms.

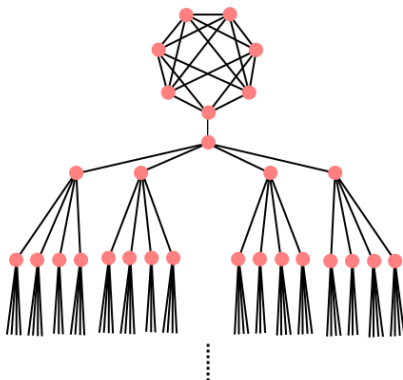


Figure 3.1: For a regular graph composed of a near-clique attached to an infinite tree, a closed walk of length $2k$ starting from within the near-clique does not typically go deeper than $O(\log k)$ down the tree. However, the support of such a closed walk is typically $k^{\Theta(1)}$.

Theorem 3.4 is nontrivial for all $\Delta = \tilde{o}(\log^{1/7} n)$; as remarked in [JTY⁺19], Paley graphs have degree $\Omega(n)$ and second eigenvalue multiplicity $\Omega(n)$, so some bound on the degree is required to obtain sublinear multiplicity.

The main new ingredient in the proof of Theorem 3.4 is a polynomial lower bound on the support of (i.e., number of distinct vertices traversed by) a simple random walk of fixed length conditioned to return to its starting point. The bound holds for any connected graph and any starting vertex and may be of independent interest.

Theorem 3.5. *Suppose G is connected and of maximum degree Δ on n vertices and x is any vertex in G . Let $\gamma_x^{2k} = (x = X_0, X_1, \dots, X_{2k})$ denote a random walk of length $2k < n$ sampled according to the simple random walk on G starting at x . Then*

$$\mathbb{P}(\text{support}(\gamma_x^{2k}) \leq s | X_{2k} = X_0) \leq \exp\left(-\frac{k}{65\Delta^7 s^4}\right) \quad \text{for } s \leq \frac{1}{4} \left(\frac{k}{\Delta^7 \log \Delta}\right)^{1/5}. \quad (3.2)$$

In particular, this means that for constant Δ , the typical support of a closed random walk of length $2k$ is least $\Omega(k^{1/5})$. It may be tempting to compare Theorem 3.5 with the familiar fact that a random closed walk of length $2k$ on \mathbb{Z} (or in continuous time, a standard Brownian bridge run for time $2k$) attains a maximum distance of $\Omega(\sqrt{k})$ from its origin. However, as seen in Fig. 3.1, there are regular graphs for which a closed walk of length $2k$ from a particular vertex x travels a maximum distance of only $\text{polylog}(k)$ with high probability. Theorem 3.5 reveals that nonetheless the number of *distinct* vertices traversed is always typically $\text{poly}(k)$. We do not know if the specific exponent of $k^{1/5}$ supplied by Theorem 3.5 is sharp, but considering a cycle graph shows that it is not possible to do better than $k^{1/2}$.

Given Theorem 3.5, our proof of Theorem 3.4 follows the strategy of [JTY⁺19]: since most closed walks in G have large support, the number of such walks may be drastically reduced by deleting a small number of vertices from G . By a moment calculation relating the spectrum to self return probabilities and a Cauchy interlacing argument, this implies an upper bound on the multiplicity of $\lambda_2(\tilde{A}_G)$. The crucial difference is that we are able to delete only $n/\text{polylog}(n)$ vertices whereas they delete $n/\text{poly log log}(n)$.

3.1.2 Related Work and Open Problems

We conclude the presentation of [MRS21] with some related work and potential further research.

Eigenvalue Multiplicity. Despite the straightforward nature of the question, relatively little is known about eigenvalue multiplicity of general graphs. As discussed in [JTY⁺19], if one assumes that G is a bounded degree expander graph, then the bound of Theorem 3.3 can be improved to $O(n/\log n)$.

There is a large gap between our upper bound of $O(n/\log^{1/5} n)$ on the multiplicity of the second eigenvalue and the lower bound of $n^{1/3}$ mentioned after Theorem 3.3. It is very natural to ask, whether the bound of this paper may be improved. To improve the bound beyond $O(n/\text{polylog}(n))$, however, it appears that a very different approach is needed.

Open Problem 1 (Similar to Question 6.3 of [JTY⁺19]). *Let $d > 1$ be fixed integer. Does there exist an $\varepsilon > 0$ such that for every connected d -regular graph G on n vertices, the multiplicity of the second largest eigenvalue of A_G is $O(n^{1-\varepsilon})$?*

In the present paper, we rely on the trace method to bound eigenvalue multiplicity through closed walks. There are three drawbacks to this approach that stops a bound on the second eigenvalue multiplicity below $n/\text{polylog}(n)$. First, considering walks of length $\omega(\log(n))$ makes the top eigenvalue dominate the trace, leaving no information behind. Second, considering the trace $\text{Tr } \tilde{A}_G^k$ for $k = O(\log(n))$ it is impossible to distinguish eigenvalues that differ by $O(1/\log(n))$. Third, in [MRS21], we give an example of graphs such that there are $\Omega(n/\text{polylog}(n))$ eigenvalues in a range of that size around the second eigenvalue. Thus, the trace method reaches a natural barrier at $n/\text{polylog}(n)$.

Support of Walks. There are as far as we are aware no known lower bounds for the support of a random closed walk of fixed length in a general graph (or even Cayley graph). It is relatively easy to derive such bounds for bounded degree graphs if the length of the walk is sufficiently larger than the mixing time of the simple random walk on the graph; the key feature of Theorem 3.5, which is needed for our application, is that the length of the walk can be taken to be much smaller.

The support of open walks (namely removing the condition that the walk ends at the starting point) is better understood. There are Chernoff-type bounds on the size of the support of a random walk based on the spectral gap [Gil98, Kah97]. Such bounds and their variants are an important tool in derandomization. It is very natural to ask whether a bound on the support of closed random walks could also have applications in theoretical computer science. We leave it as an open problem to apply Theorem 3.5 in such a setting.

We have no reason to believe that the exponent of $1/5$ appearing in Theorem 3.5 is sharp. In fact, we know of no example where where the answer is $o(k^{1/2})$. An improvement over Theorem 3.5 would immediately yield an improvement of Theorem 3.4.

Open Problem 2. *Let $d > 1$ be a fixed integer. Does there exist an $\alpha > 1/5$ such that for every connected d -regular graph G on n vertices and every vertex x of G , a random closed walk of length $2k < n$ rooted at x has support $\Omega(k^\alpha)$ in expectation? Is it even true for $\alpha = 1/2$? Does such a bound hold for simple random walks in general?*

3.2 Non-Malleable Codes from Expander Graphs

In this section, we present the paper “Expander Graphs are Non-Malleable Codes” [RS20]. We start by introducing non-malleable codes and then describe our results. The section consists of edited excerpts from the introduction of [RS20].

A key goal in theoretical computer science is the identification of structures that exhibit resilience to adversarial tampering. The classical notion in this space is that of an error-detection or error-correction code, where we seek to ensure that tampering caused by an adversary that can modify a *bounded* number of symbols in a codeword can be detected or corrected.

But what if the number of errors that an adversary can introduce is unbounded? The objective of error detection or correction is clearly impossible to achieve in this setting – the adversary can simply replace the transmitted codeword with an encoding of some other fixed value. Thus, the main question of study in this context concerns the notion of *malleability*: informally speaking, our core goal must be to prevent the adversary from replacing an encoding of a value x with an encoding of some other related value $\tilde{x} \neq x$.

The central information-theoretic object in this setting is called a split-state non-malleable code [DPW18]. Since their introduction in 2010 [DPW18], split-state non-malleable codes have been the subject of intense study within theoretical computer science [DPW18, DKO13, ADL14, CZ14, CGL16, Li17]. A split-state non-malleable code [DPW18] consists of randomized encoding and decoding algorithms (enc, dec). A message $m \in \mathcal{M}$ is encoded as a pair of strings $(L, R) \in \{0, 1\}^k \times \{0, 1\}^k$, such that $\text{dec}(L, R) = m$. An adversary then specifies an arbitrary pair of functions $g, h : \{0, 1\}^k \rightarrow \{0, 1\}^k$. The code is said to be non-malleable if, intuitively, the message obtained as $\text{dec}(g(L), h(R))$ is “unrelated” to the original message m . This is formalised as follows.

Definition 3.6. [Split-State Non-Malleable Code] Let a coding scheme (enc, dec) be given by functions $\text{enc} : \mathcal{M} \rightarrow \mathcal{L} \times \mathcal{R}$ and $\text{dec} : \mathcal{L} \times \mathcal{R} \rightarrow \mathcal{M} \cup \{\perp\}$. We say that (enc, dec) is ϵ -*non-malleable* if for every pair of functions $f : \mathcal{L} \rightarrow \mathcal{L}$ and $g : \mathcal{R} \rightarrow \mathcal{R}$ there exists a distribution $D_{f,g}$ taking values in $\mathcal{M} \cup \{\perp, *\}$ such that for every $m \in \mathcal{M}$, the statistical distance between the two experiments

$$A := \left\{ \begin{array}{l} (L, R) := \text{enc}(m) \\ \text{Output } \text{dec}(f(L), g(R)) \end{array} \right\}$$

$$B := \left\{ \begin{array}{l} \text{Sample } \tilde{m} \text{ from } D_{f,g} \\ \text{If } \tilde{m} = * \text{ output } m \text{ else output } \tilde{m} \end{array} \right\}$$

is at most ϵ .

3.2.1 Our Results

All known constructions and proofs of security for explicit split-state non-malleable codes have required complex mathematical proofs, and all known such proofs either directly or indirectly used the mathematics behind constructions of two-source extractors [DKO13, ADL14, CZ14, CGL16, Li17]. In fact, after constructing the first non-malleable code in the split-state model Dziembowski, Kazana, and Obremski wrote: “This brings a natural question if we could show some relationship between the extractors and the non-malleable codes in the split-state model. Unfortunately, there is no obvious way of formalizing the conjecture that non-malleable codes need to be based on extractors” [DKO13].

In [RS20], we seek to establish new, simpler, foundations for the construction of split-state non-malleable codes. We do so by answering in the negative the implicit conjecture of [DKO13];

we show that it is not necessary to base constructions of non-malleable codes on the theory of extractors. Instead, we construct a non-malleable code for single bit messages from expander graphs, an object arguably simpler and more fundamental than two-source extractors.

Taking a regular graph G , we introduce in [RS20] a natural way to encode a single bit message.

Definition 3.7. [Graph Code] Let $G = (V, E)$ be a graph. The *graph code* associated with G , denoted $(\text{enc}_G, \text{dec}_G)$, encodes a single bit $b \in \{0, 1\}$ as a pair of vertices of G as follows.

$$\text{enc}_G(b) = \begin{cases} (u, v) \leftarrow_u E, & b = 1 \\ (u, v) \leftarrow_u (V \times V) \setminus E, & b = 0, \end{cases}$$

$$\text{dec}(u, v) = \begin{cases} 1, & (u, v) \in E \\ 0, & (u, v) \notin E. \end{cases}$$

Here, $X \leftarrow_u A$ signifies that X is sampled uniformly at random from the set A .

The main theorem of [RS20] states that if G is a sufficiently good spectral expander, the graph code associated with G is non-malleable.

Theorem 3.8. *Let G be a d -regular graph with spectral expansion λ satisfying $n = \Omega(d^3 \log d / \lambda)$. Then the single bit split-state non-malleable code $(\text{enc}_G, \text{dec}_G)$ is $O(\lambda^{3/2}/d)$ -non-malleable.*

3.2.2 Perspectives

Our main contribution is that the construction of non-malleable codes from expander graphs opens up a new line of attack in the study of split-state non-malleable codes. It is important to keep in mind that current constructions of non-malleable codes supporting messages of arbitrary length use many ideas pioneered in the construction of [DKO13], in particular the use of extractors. While we do not yet know how to generalise our results beyond single-bit messages, we speculate that further investigation building upon our work will reveal a deeper connection and more powerful simple constructions based on expanders.

It should be noted that two-source extractors are well-known to exhibit expansion properties; however, in all previous proofs, much more than mere expansion was used to argue non-malleability. Indeed previous proofs apply extractors repeatedly. We also note that it is not surprising that 1-bit non-malleable codes will exhibit some sort of expansion properties. Our contribution is the converse: that good expansion is *sufficient* for the construction of non-malleable codes.

Chapter 4

Edge Connectivity in Graphs

Let $G = (V, E)$ be a graph. A fundamental question in graph theory and algorithms is: Given two vertices $u, v \in V$, is there a path on G that starts at u and ends at v ? If this is the case, we say that u and v are *connected* in G . This notion generalises in a natural way.

Definition 4.1. [*k*-Edge Connectivity] Let $G = (V, E)$ be a graph. Two vertices $u, v \in V$ are *k-edge connected* if there exists k distinct paths $\gamma_1, \dots, \gamma_k$ on G each of them starting at u and ending at v and no two of them sharing an edge.

Equivalently, u and v are *k-edge connected* in G if the removal of any $k - 1$ edges of G leave u and v still connected. Being *k-edge connected* is an equivalence relation on the vertices of G , and we call the corresponding equivalence classes the *k-edge connected classes* of G .

Taking a broader perspective, we say that the graph G itself is *connected* if every pair of distinct vertices of G is connected. Generalising this notion as well, we say that G is *k-edge connected* if every pair of vertices of G are *k-edge connected*. Equivalently, G is *k-edge connected* if for every set $S \subset E$ of $|S| = k - 1$ edges, $G \setminus S$ is connected. Note that with this definition, the trivial graph on a single vertex is *k-edge connected*.

Definition 4.2. [*k*-Edge Connected Component] Let $G = (V, E)$ be a graph. An induced subgraph $H = G[U], U \subset V$, is a *k-edge connected component* of G if H is *k-edge connected* and for every $U \subsetneq U' \subset V$ the induced subgraph $G[U']$ is not *k-edge connected*.

In other words, a *k-edge connected component* of G is a maximal *k-edge connected* induced subgraph of G . Since the trivial graph is *k-edge connected*, every vertex of G is contained in a *k-edge connected component*. Furthermore, every pair of distinct *k-edge connected components* of a graph are disjoint. Thus, the *k-edge connected components* of G partition the vertices of G .

It is important to keep in mind the non-obvious distinction between *k-edge connected classes* and *components*. For $k = 1, 2$ the *k-edge connected classes* and *components* coincide, but for $k \geq 3$ this is not necessarily the case. For an example with $k = 3$, see Fig. 4.1.

In this chapter, we present the two papers “Optimal Decremental Connectivity in Non-Sparse Graphs” [AKL⁺21] and “*k*-Edge Connected Components and Minimum Degree” [ART21]. The first considers the algorithmic problem of keeping track of the *k-edge connected components* of a graph while edges are deleted one by one. The second studies how many *k-edge connected components* a graph with high minimum degree may contain.

Remark 4.3. The work presented in this chapter partially overlaps with the master thesis submitted by the author in September 2020 [Ras20]. The main results of [ART21] resemble

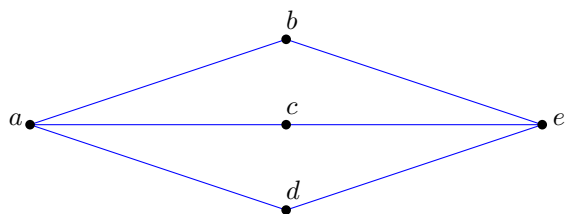


Figure 4.1: The 3-edge connected classes of the above graph are $\{a, e\}$, $\{b\}$, $\{c\}$, and $\{d\}$, since a and e are 3-edge connected by the paths through b , c , and d . The 3-edge connected components of the graph are simply the individual vertices of the graph, since the subgraph induced by a and e is not 3-edge connected (nor connected for that matter).

those presented in [Ras20]. However, the proofs have been rewritten, a significant error in the main theorem has been fixed, and further observations have been added along with some historical context. The research of [AKL⁺21] is a strong improvement over the work done in [Ras20]. Most concretely, the algorithm of [Ras20] is only feasible for graphs on n vertices and $m = \Omega(n^{3/2} \text{polylog } n)$ edges, whereas the algorithm presented in [AKL⁺21] is feasible for all graphs with $m = \Omega(n \text{polylog } n)$ edges. Furthermore, while some of the central ideas from [Ras20] appear in [AKL⁺21], a significant amount of work and new ideas have gone into strengthening the results of [AKL⁺21].

4.1 Maintaining k -edge Connected Components

In this section, we present the paper ““Optimal Decremental Connectivity in Non-Sparse Graphs” [AKL⁺21]. We start with an introduction to dynamic graph algorithms and dynamic k -edge connectivity, proceed with an overview of previous results, and finally describe the results of the paper.

The topic of this section is dynamic graph algorithms. A traditional (static) graph algorithm takes as input a graph $G = (V, E)$ and a query q and outputs the answer to q with respect to G . For instance, a graph algorithm may be tasked with deciding if two vertices u and v are connected in G . In contrast, a dynamic graph algorithm \mathcal{A} receives as input a starting graph G followed by a sequence of updates, u_1, \dots, u_t , each of which either removes or adds an edge to G . We denote by $G_i, i \geq 0$ the graph G after i updates have been applied. The updates are interspersed with queries q_1, \dots, q_ℓ . The task of \mathcal{A} is to maintain a data structure such that when query q_j occurs immediately after update u_i , \mathcal{A} may respond with the answer to the query q_j on G_i . Note that the updates and queries are received one at a time, so \mathcal{A} is for instance oblivious to update u_{i+1} when processing update u_i . Adopting the example from the static graph algorithm, the queries may be to decide whether a pair of vertices u and v are connected in G_i . This is known as the *dynamic connectivity problem*. In that case, \mathcal{A} should maintain a data structure that makes such queries quick to answer.

The performance of dynamic graph algorithms is measured in two ways: The *update time* and the *query time*. If we denote by $T(u_i)$ and $T(q_j)$ the time taken to perform update u_i and answer query q_j , the worst case update and query times will be $\max T(u_i)$ and $\max T(q_j)$, respectively. In our case, we will mostly work with *amortised update time*, $\frac{1}{t} \sum T(u_i)$, i.e., the average time taken to perform an update. In fact, since we are working with randomised algorithms, we consider *expected amortised update time*.

Furthermore, dynamic graph algorithms come in three different flavours or settings. They are the *fully*, *decremental*, and *incremental* dynamic graph algorithms. Fully dynamic graph algorithms are as described above: an update is either the insertion or deletion of an edge. Meanwhile, for decremental or incremental dynamic graph algorithms, respectively, we consider the case where edges are only ever deleted or added. When measuring performance, we suppose for the fully and incremental dynamic graph algorithms that the starting graph G is the empty graph, and for decremental dynamic graph algorithms that all edges of G are eventually removed such that the final graph G_t is empty.

Dynamic connectivity is perhaps the most fundamental dynamic graph problem. In [AKL⁺21], we consider this problem and the more general problem of k -edge connectivity, i.e., asking whether vertices are k -edge connected rather than simply connected.

4.1.1 Overview of Previous Results

The world of dynamic connectivity is vast, so we will only briefly describe the relevant results from the literature.

Fully Dynamic Connectivity. Fully dynamic connectivity is extremely well-studied (see e.g., [CGL⁺20, EGIN97, Fre85, HK99, HdLT01, HHKP17, KKM13, KKPT16, NSW17, PD04, PT11, Tho00, Wan15, Wul13]) with near-optimal upper and lower bounds. We shall not dwell on the lower bounds of Pătraşcu and Demaine [PD04] and Pătraşcu and Thorup [PT11] except to say that any fully dynamic algorithm for connectivity with amortised update time $o(\log n)$ has query time at least $\Omega(n^{1-o(1)})$. In other words, with amortised update time $o(\log n)$ we can

k	Amortised update time	Worst case query time	Source
1	$O(\log(n) \log^2 \log(n))$	$O(\log(n)/\log \log(n))$	Huang et al. [HHKP17]
2	$O(\log^2(n) \log^2 \log(n))$	$O(\log(n)/\log \log(n))$	Holm, Rotenberg, and Thorup [HRT18]
$\log^{o(1)}(n)$	$n^{o(1)}$	$n^{o(1)}$	Jin and Sun [JS20]

Figure 4.2: The state of the art in randomised fully dynamic k -connectivity. It is worth noting that up to factors of polyloglog n , the algorithm of Huang et al. exhibits an optimal trade of in performance between update and query time. Furthermore, the $n^{o(1)}$ update time of Jin and Sun is in fact worst case.

do almost no better on queries than to simply run a static connectivity algorithm. When it comes to upper bounds, Fig. 4.2 displays the current state of the art for randomised algorithms. For the purpose of this presentation, the results may be sufficiently summarised as follows. For $k = 1, 2$, fully dynamic k -connectivity is solved with update and query time $O(\text{polylog } n)$, while for $k = \log^{o(1)}(n)$, it may be solved with update and query time $n^{o(1)}$.

Incremental Dynamic Connectivity. Consider a setting where we start from an empty graph and insert edges until we arrive at a graph G on n vertices and m edges, asking q queries along the way. Applying the union-find algorithm analysed by Tarjan [Tar75], we perform $O(m+q)$ find and $O(n)$ union operations. The total cost is then $O((m+q)\alpha(m+q, n))$, where α denotes the inverse-Ackermann function. Since $\alpha(a \log a, a) = O(1)$, the total running time is linear in $m+q$ (and hence optimal) for all graphs with $m = \Omega(n \log n)$ edges. This has been shown to be optimal for incremental dynamic connectivity [FS89].

Decremental Dynamic Connectivity. In [Tho99], Thorup presents an algorithm for the problem of decremental dynamic connectivity. The algorithm answers queries in constant time and the time spent performing the updates required to delete every edge of a graph G on n vertices and m edges is $O(n \text{ polylog } n + m \log(n^2/m))$. In other words, the amortised update time is $O(1)$ for dense graphs with $m = \Omega(n^2)$ and $O(\log n)$ otherwise. This also holds for 2-edge connectivity. For k -edge connectivity with $k \geq 3$, a slightly different result holds, which we will get back to.

The algorithm by Thorup [Tho99] is in fact a reduction from the decremental dynamic k -edge connectivity problem to the *fully dynamic k -edge-cut problem* and the fully dynamic k -edge connectivity problem. The fully dynamic k -edge-cut problem is the problem of maintaining in a graph H undergoing edge deletions and insertions an edge of H that is contained in a cut of size $< k$ if such an edge exists. We denote by $T_k(n)$ the amortised update time needed to solve the fully dynamic k -edge-cut problem. We have $T_1(n) = O(1)$ since for $k = 1$ there is nothing to maintain, and, for $k = 2$, the algorithm of [HRT18] featured in Fig. 4.2 maintains the bridges of the graph, so $T_2(n) = O(\log^2(n) \log^2 \log(n))$.

The reduction operates by maintaining a *k -certificate* of G .

Definition 4.4. [k -certificate] Let G be a graph. A subgraph $H \subset G$ is a *k -certificate* for G if the k -edge connected components of H and G are the same and H contains every edge of G that connects distinct k -edge connected components.

A k -certificate H of G contains all information regarding k -edge connectivity of G . The graph H has the same k -edge connected components and classes. Let G be a graph on n vertices and m edges. The algorithm of Thorup [Tho99] maintains a k -certificate of G in total time $O(m \log(n^2/m) + knT_k(n) \text{ polylog}(n))$ using an algorithm for the fully dynamic k -edge-cut

problem. Over the course of all deletions, a total of $O(kn)$ edges are inserted in or deleted from the certificate. To achieve an algorithm for decremental k -edge connectivity, we apply an algorithm for fully dynamic k -edge connectivity to the certificate. For $k \in \{1, 2\}$, since the certificate is only updated $O(kn)$ times and the fully dynamic k -edge connectivity of [HHKP17, HRT18] have polylogarithmic amortised update times, the total time to maintain decremental k -edge connectivity is $O(m \log(n^2/m) + n \text{polylog } n)$. For $k = \log^{o(1)} n$, we may apply the fully dynamic min-cut algorithm of Thorup [Tho07] which establishes $T_k(n) = n^{1/2+o(1)}$ along with the $n^{o(1)}$ update time of the fully dynamic k -edge connectivity algorithm of [JS20] to obtain decremental k -edge connectivity in total time $O(m \log(n^2/m) + n^{3/2+o(1)})$ and query time $O(n^{o(1)})$.

4.1.2 Better k -Certificate and Optimal Decremental Connectivity

In [AKL⁺21], we improve on the reduction of Thorup [Tho99] and get the following result.

Theorem 4.5. *Let G be a graph on n vertices and m edges. There exists a randomised algorithm for decrementally maintaining a k -certificate of G in total time $O(m + knT_k(n) \text{polylog}(n))$ using $O(kn \text{polylog}(n))$ updates to the certificate.*

This replaces the term $O(m \log(n^2/m))$ in the running time of the algorithm of Thorup [Tho99] with simply $O(m)$. Applying the analysis above, this improvement immediately implies the following.

Theorem 4.6. *Let G be a graph on n vertices and m edges. There exist algorithms for the decremental dynamic 1- and 2-edge connectivity problems on G with constant query time and total update time $O(n \text{polylog}(n) + m)$.*

Thus, for every graph G on n vertices and $m = \Omega(n \text{polylog}(n))$ edges, we present an optimal algorithms for for decremental dynamic 1- and 2-edge connectivity. In contrast, the algorithm of Thorup [Tho99] is only optimal whenever $m = \Omega(n^2)$.

Furthermore, for k -edge connectivity, we obtain the following.

Theorem 4.7. *Let G be a graph on n vertices and m edges. For $k = \log^{o(1)} n$, there exist algorithms for the decremental dynamic k -edge connectivity problems on G with $n^{o(1)}$ query time and total update time $O(n^{3/2+o(1)} + m)$.*

A nice application of the results above is to the unique perfect matching problem. Gabow et al. [GKT01] use decremental dynamic 2-edge connectivity to yield an algorithm for deciding whether a graph has a unique perfect matching. Plugging in Theorem 4.6 immediately yields the following.

Theorem 4.8. *Let G be a graph on n vertices and m edges. There exists a Las Vegas algorithm that decides in time $O(n \text{polylog}(n) + m)$ whether G has a unique perfect matching.*

Again, whenever $G = \Omega(n \text{polylog}(n))$, this is $O(m)$ and thus, optimal.

As evident above, our overall approach to the problem of decremental k -edge connectivity is similar to Thorup's. Our main innovation is a new sparse randomised k -certificate which may be maintained with $O(1)$ amortised time spent per edge deletion in contrast to the $O(\log(n^2/m))$ required by the certificate of Thorup [Tho99]. Unfortunately, a discussion of the details of the certificates is beyond this synopsis. We refer the reader to the full manuscript of [AKL⁺21] in the appendix.

4.1.3 Discussion

Given that we have near-optimal randomised algorithms for fully dynamic connectivity and an optimal algorithm for incremental dynamic connectivity, it is very interesting that our work in [AKL⁺21] also comes very close to establishing an optimal algorithm for decremental dynamic connectivity. Except for sparse graphs on $m = O(n \text{ polylog } n)$ edges, our algorithm solves the question of randomised decremental dynamic connectivity. This raises the question of what happens for sparser graphs.

Open Problem 3. *What expected amortised update time can be achieved for graphs on n vertices and $m = O(n \text{ polylog}(n))$ edges for the problem of decremental dynamic connectivity?*

4.2 Counting k -edge Connected Components

In this section, we present the paper “ k -Edge Connected Components and Minimum Degree” [ART21]. The following text consists of edited excerpts from Sections 1, 3, and 4 of [ART21].

All graphs in this section are simple. A key observation applied in [AKL⁺21] is that graphs with high minimum degree δ are well-connected. In particular, they have few k -edge connected components whenever $\delta \gg k$. In [ART21], we further explore and quantify this phenomenon. It is well-known that whenever a graph with a fixed number of vertices contains sufficiently many edges, non-trivial k -edge connected components begin to emerge. This is the content of a classical result by Mader.

Theorem 4.9 (Mader [Mad71]). *Let k and $n > k$ be positive integers. Every graph on n vertices and strictly more than $(k-1)(n-k/2)$ edges contains a non-trivial k -edge connected component. Furthermore, there exists a graph on n vertices and $(k-1)(n-k/2)$ edges containing only trivial k -edge connected components.*

We consider a related question, establishing a connection between the minimum degree and the number of k -edge connected components of a graph. The main result of [ART21] is an upper bound on the number of k -edge connected components.

Theorem 4.10. *Let $k > 1$ be an integer and $\varepsilon > 0$. Every graph on n vertices with minimum degree $(2 + \varepsilon)(k - 1)$ has at most $\frac{10n}{\varepsilon k}$ distinct k -edge connected components.*

We further show that this bound is tight in two ways. First, the upper bound is tight asymptotically in the sense that for every integer $k > 1$ and $\varepsilon > 0$, the maximal possible number of k -edge connected components of a graph on some n vertices with minimum degree $(2 + \varepsilon)(k - 1)$ is $\Theta\left(\frac{n}{\varepsilon k}\right)$. The lower bound establishing this is the content of another theorem of [ART21].

Theorem 4.11. *For every integer $k > 1$ and every $\varepsilon \geq 1/(k - 1)$ there exists $n \in \mathbb{N}$ and a graph G with n vertices, minimum degree $\geq (2 + \varepsilon)(k - 1)$, and at least $\frac{n}{3\varepsilon k}$ distinct k -edge connected components.*

Second, the limit at minimum degree $2(k - 1)$ proposed by the theorem is in fact tight in the following sense. For fixed $k > 1$, there is a constant $\delta < 1$ such that every graph on some n vertices with minimum degree $2k - 1$ has at most δn distinct k -edge connected components. However, there is no such constant for graphs of minimum degree $2(k - 1)$. This is laid out in the following two propositions.

Proposition 4.12. *Let G be a graph on n vertices. If G has minimum degree $2k - 1$, the number of distinct k -edge connected components of G is at most $\frac{2k-1}{3k-1}n$.*

Proposition 4.13. *For every integer $k > 1$ and real $\gamma \in (0, 1)$ there exists $n \in \mathbb{N}$ and a graph G on n vertices satisfying that G has minimum degree $2(k - 1)$ and the number of k -edge connected components of G is at least γn .*

Finally, we note that since this work was focused on establishing the asymptotic behaviour of the number of k -edge connected components, the constants appearing in the theorems can almost certainly be improved.

Open Problem 4. *What are the right constants for the results of Theorem 4.10 and Theorem 4.11?*

Bibliography

- [AAAR20] Anders Aamand, Mikkel Abrahamsen, Thomas D. Ahle, and Peter Michael Reichstein Rasmussen. Tiling with squares and packing dominos in polynomial time. *CoRR*, 2020. <https://arxiv.org/abs/2011.10983>. In submission.
- [AAKR21] Anders Aamand, Mikkel Abrahamsen, Jakob Bæk Tejs Knudsen, and Peter Michael Reichstein Rasmussen. Classifying Convex Bodies by Their Contact and Intersection Graphs. In *Proceedings of the 37th ACM Symposium on Computational Geometry (SoCG)*, volume 189, pages 3:1–3:16, 2021.
- [ABS15] Sanjeev Arora, Boaz Barak, and David Steurer. Subexponential algorithms for unique games and related problems. *Journal of the ACM*, 62(5):1–25, 2015.
- [AC88] N. Alon and F.R.K. Chung. Explicit construction of linear sized tolerant networks. In *Graph Theory and Applications*, volume 38, pages 15–19. 1988.
- [ADK⁺21] Anders Aamand, Debarati Das, Evangelos Kipouridis, Jakob B. T. Knudsen, Peter Michael Reichstein Rasmussen, and Mikkel Thorup. No repetition: Fast streaming with highly concentrated hashing. Manuscript, 2021.
- [ADL14] Divesh Aggarwal, Yevgeniy Dodis, and Shachar Lovett. Non-malleable codes from additive combinatorics. In *Proceedings of the 46th ACM Symposium on Theory of Computing (STOC)*, pages 774–783, 2014.
- [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 Symposium on Theory of Computing (STOC)*, pages 1265–1278, 2020.
- [AKL⁺21] Anders Aamand, Adam Karczmarzand, Jakub Łącki, Nikos Parotsidis, Peter Michael Reichstein Rasmussen, and Mikkel Thorup. Optimal decremental connectivity in non-sparse graphs. Manuscript, 2021.
- [Alo86] Noga Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, June 1986.
- [App16] Austin Appleby. Murmurhash3. <https://github.com/aappleby/smhasher/wiki/MurmurHash3>, 2016.
- [ART21] Anders Aamand, Peter Michael Reichstein Rasmussen, and Mikkel Thorup. k -edge connected components and minimum degree. Manuscript, 2021.
- [Ben62] George Bennett. Probability inequalities for the sum of independent random variables. *Journal of the American Statistical Association*, 57(297):33–45, 1962.

- [Ber24] Sergei Natanovich Bernstein. On a modification of Chebyshev’s inequality and of the error formula of Laplace. *Ann. Sci. Inst. Sav. Ukraine, Sect. Math.*, (1):38–49, 1924.
- [BGH⁺15] Boaz Barak, Parikshit Gopalan, Johan Håstad, Raghu Meka, Prasad Raghavendra, and David Steurer. Making the long code shorter. *SIAM Journal on Computing*, 44(5):1287–1324, 2015.
- [Boß12] Martin Boklet. Breaking murmur: Hash-flooding dos reloaded. <https://embooss.github.io/blog/2012/12/14/breaking-murmur-hash-flooding-dos-reloaded/>, 2012.
- [BRS11] Boaz Barak, Prasad Raghavendra, and David Steurer. Rounding semidefinite programming hierarchies via global correlation. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 472–481, 2011.
- [BYJK⁺02] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 1–10, 2002.
- [CGL16] Eshan Chattopadhyay, Vipul Goyal, and Xin Li. Non-malleable extractors and codes, with their many tampered extensions. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*, pages 285–298, 2016.
- [CGL⁺20] Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. In *Proceedings of the 61st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 1158–1167, 11 2020.
- [Che52] Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23(4):493–507, 1952.
- [CW79] Larry Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979. Announced at STOC’77.
- [CZ14] Eshan Chattopadhyay and David Zuckerman. Non-malleable codes against constant split-state tampering. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 306–315, 2014.
- [DHKP97] Martin Dietzfelbinger, Torben Hagerup, Jyrki Katajainen, and Martti Penttonen. A reliable randomized algorithm for the closest-pair problem. *Journal of Algorithms*, 25(1):19–51, 1997.
- [DKO13] Stefan Dziembowski, Tomasz Kazana, and Maciej Obremski. Non-malleable codes from two-source extractors. In *Proceedings of the 33rd Annual Cryptology Conference (CRYPTO)*, volume 8043, pages 239–257, 2013.
- [DM92] Martin Dietzfelbinger and Friedhelm Meyer auf der Heide. Dynamic hashing in real time. In *Informatik, Festschrift zum 60. Geburtstag von Günter Hotz*, pages 95–119. 1992.

- [DPW18] Stefan Dziembowski, Krzysztof Pietrzak, and Daniel Wichs. Non-malleable codes. *Journal of the ACM*, 65(4), 2018. Announced at ICS'10.
- [Dum56] Arnold I. Dumey. Indexing for rapid random access memory systems. *Computers and Automation*, 5(12):6–9, 1956.
- [EGIN97] David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *Journal of the ACM*, 44(5):669–696, 1997.
- [Fre85] Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM Journal on Computing*, 14(4):781–798, 1985.
- [FS89] M. Fredman and M. Saks. The cell probe complexity of dynamic data structures. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC)*, pages 345–354, 1989.
- [Gil98] David Gillman. A chernoff bound for random walks on expander graphs. *SIAM Journal on Computing*, 27(4):1203–1220, 1998.
- [GKT01] Harold N. Gabow, Haim Kaplan, and Robert Endre Tarjan. Unique maximum matching algorithms. *Journal of Algorithms*, 40(2):159–183, 2001.
- [GT15] S. Gharan and L. Trevisan. A new regularity lemma and faster approximation algorithms for low threshold rank graphs. *Theory of Computing*, 11:241–256, 2015.
- [HdLT01] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM*, 48(4):723–760, 2001.
- [HHKP17] Shang-En Huang, Dawei Huang, Tsvi Kopelowitz, and Seth Pettie. Fully dynamic connectivity in $O(\log n(\log \log n)^2)$ amortized expected time. In *Proceedings of the 28th Annual ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 510–520, 2017.
- [HK99] Monika Rauch Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *Journal of the ACM*, 46(4):502–516, 1999.
- [HRT18] Jacob Holm, Eva Rotenberg, and Mikkel Thorup. Dynamic bridge-finding in $\tilde{O}(\log^2 n)$ amortized time. In *Proceedings of the 29th Annual ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 35–52, 2018.
- [JOWO20] Samuel Neves Jack O’Connor, Jean-Philippe Aumasson and Zooko Wilcox-O’Hearn. Blake3. <https://github.com/BLAKE3-team/BLAKE3>, 2020.
- [JS20] Wenyu Jin and Xiaorui Sun. Fully dynamic c-edge connectivity in subpolynomial time. *CoRR*, 2020. <https://arxiv.org/abs/2004.07650>.
- [JTY+19] Zilin Jiang, Jonathan Tidor, Yuan Yao, Shengtong Zhang, and Yufei Zhao. Equian-gular lines with a fixed angle. *CoRR*, 2019. <https://arxiv.org/abs/1907.12466>.
- [Kah97] Nabil Kahale. Large deviation bounds for markov chains. *Combinatorics Probability and Computing*, 6(4):465–474, 1997.

- [KKM13] Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the 24th Annual ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 1131–1142, 2013.
- [KKPT16] Casper Kejlberg-Rasmussen, Tsvi Kopelowitz, Seth Pettie, and Mikkel Thorup. Faster worst case deterministic dynamic connectivity. In *Proceedings of the 24th Annual European Symposium on Algorithms (ESA)*, pages 53:1–53:15, 2016.
- [Kol11] Alexandra Kolla. Spectral algorithms for unique games. *computational complexity*, 20(2):177–206, 2011.
- [Li17] Xin Li. Improved non-malleable extractors, non-malleable codes and independent source extractors. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1144–1156, 2017.
- [LOG18] Russell Lyons and Shayan Oveis Gharan. Sharp bounds on random walk eigenvalues via spectral embedding. *International Mathematics Research Notices*, 2018(24):7555–7605, 2018.
- [LOGT14] James R Lee, Shayan Oveis Gharan, and Luca Trevisan. Multiway spectral partitioning and higher-order cheeger inequalities. *Journal of the ACM*, 61(6):1–30, 2014.
- [LRTV12] Anand Louis, Prasad Raghavendra, Prasad Tetali, and Santosh Vempala. Many sparse cuts via higher eigenvalues. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1131–1140, 2012.
- [Mad71] W. Mader. Minimalen-fach kantenzusammenhängende graphen. *Mathematische Annalen*, 191:21–28, 1971.
- [MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
- [MRS21] Theo McKenzie, Peter Michael Reichstein Rasmussen, and Nikhil Srivastava. Support of closed walks and second eigenvalue multiplicity of graphs. In *Proceedings of the 53rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 396–407, 2021.
- [NSW17] Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *Proceedings of the 58th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 950–961, 2017.
- [PD04] Mihai Pătraşcu and Erik D. Demaine. Lower bounds for dynamic connectivity. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 546–553, 2004.
- [PT11] Mihai Pătraşcu and Mikkel Thorup. Don’t rush into a union: take time to find your roots. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 559–568, 2011.
- [PT12] Mihai Pătraşcu and Mikkel Thorup. The power of simple tabulation-based hashing. *Journal of the ACM*, 59(3):14:1–14:50, 2012. Announced at STOC’11.

- [PT13] Mihai Pătraşcu and Mikkel Thorup. Twisted tabulation hashing. In *Proceedings of the 24th Annual ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 209–228, 2013.
- [Ras20] Peter Michael Reichstein Rasmussen. Incremental graph connectivity via uniform edge sampling, 2020. Master’s Thesis.
- [RS20] Peter Michael Reichstein Rasmussen and Amit Sahai. Expander graphs are non-malleable codes. In *Proceedings of the 1st Conference on Information-Theoretic Cryptography (ITC)*, pages 6:1–6:10, 2020.
- [Sie04] Alan Siegel. On universal classes of extremely random constant-time hash functions. *SIAM Journal on Computing*, 33(3):505–543, 2004. Announced at FOCS’89.
- [SJ89] Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Information and Computation*, 82(1):93–133, 1989.
- [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. Announced at SODA’93.
- [Tar75] Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975.
- [Tho99] Mikkel Thorup. Incremental dynamic connectivity. *Journal of Algorithms*, 33(2):229–243, 1999.
- [Tho00] Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 343–350, 2000.
- [Tho07] Mikkel Thorup. Fully-dynamic min-cut. *Combinatorica*, 27(1):91–127, 2007.
- [Tho13] Mikkel Thorup. Simple tabulation, fast expanders, double tabulation, and high independence. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 90–99, 2013.
- [Wan15] Zhengyu Wang. An improved randomized data structure for dynamic graph connectivity. *CoRR*, 2015. <http://arxiv.org/abs/1510.04590>.
- [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. Announced at FOCS’79.
- [Wul13] Christian Wulff-Nilsen. Faster deterministic fully-dynamic graph connectivity. In *Proceedings of the 24th Annual ACM/SIAM Symposium on Discrete Algorithms (SODA)*, pages 1757–1769, 2013.
- [Zob90] A. Zobrist. A new hashing method with application for game playing. *ICGA Journal*, 13:69–73, 1990. Also appeared as Technical Report number 88, University of Wisconsin-Madison, 1970.

Appendix A

Fast Hashing with Strong Concentration Bounds

Fast Hashing with Strong Concentration Bounds

ANDERS AAMAND, BARC, University of Copenhagen, Denmark

JAKOB BÆK TEJS KNUDSEN, BARC, University of Copenhagen, Denmark

MATHIAS BÆK TEJS KNUDSEN, SupWiz, Denmark

PETER MICHAEL REICHSTEIN RASMUSSEN, BARC, University of Copenhagen, Denmark

MIKKEL THORUP, BARC, University of Copenhagen, Denmark

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

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

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

CCS Concepts: • **Theory of computation** → **Pseudorandomness and derandomization**; *Data structures design and analysis*; *Sketching and sampling*.

Additional Key Words and Phrases: hashing, Chernoff bounds, concentration bounds, streaming algorithms, sampling

CONTENTS

Abstract	1
Contents	2
1 Introduction	1
1.1 Simple Tabulation Hashing	3
1.2 Tabulation-Permutation Hashing	4
1.3 Tabulation-1Permutation	5
1.4 Subpolynomial Error Probabilities	6
1.5 Generic Remarks on Universe Reduction and Amount of Randomness	6
1.6 Techniques	7
1.7 Related Work – Theoretical and Experimental Comparisons	9
2 Technical Theorems and how they Combine	16
2.1 Improved Analysis of Simple Tabulation	17
2.2 Permuting the Hash Range	18
2.3 Squaring the Hash Range	19
2.4 Concentration in Arbitrary Intervals.	19
3 Preliminaries	20
3.1 Notation	20
3.2 Probability Theory and Martingales	21
3.3 Martingale Concentration Inequalities	22
4 Analysis of Simple Tabulation	23
4.1 Simple Tabulation Basics	23
4.2 Bounding the Sum of Squared Deviations	24
4.3 Establishing the Concentration Bound	34
5 General Value Functions – Arbitrary Bins	38
6 Extending the Hash Range	44
7 Query invariance	48
8 Tightness of Concentration: Simple Tabulation into Few Bins	49
References	51
A Experiments	52

1 INTRODUCTION

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

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

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

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

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

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

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

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

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

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

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

$f(u, n, m)$ if for every $t > 0$,

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

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

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

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

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

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

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

In this paper, we construct and analyse a new family of fast hash functions *tabulation-permutation* hashing that has Chernoff-style concentration bounds like (3) without any restrictions on μ . This generality is important if building a general online system with no knowledge of future input. Later, we shall give concrete examples from streaming where μ is in fact large. Our bounds hold for all of the cases described above and all possible inputs. Furthermore, tabulation-permutation hashing is an order of magnitude faster than any other known hash function with similar concentration bounds, and almost as fast as simple and twisted tabulation. We demonstrate this both theoretically and

experimentally. Stepping back, our main theoretical contribution lies in the field of analysis of algorithms, and is in the spirit of Knuth’s analysis of linear probing [29], which shows strong theoretical guarantees for a very practical algorithm. We show that tabulation-permutation hashing has strong theoretical Chernoff-style concentration bounds. Moreover, on the practical side, we perform experiments, summarized in Table 1 of Section 1.7, demonstrating that it is comparable in speed to some of the fastest hash functions in use, none of which provide similar concentration bounds.

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

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

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

1.1 Simple Tabulation Hashing

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

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

THEOREM 1.1. *Let $h: [u] \rightarrow [m]$ be a simple tabulation hash function with $[u] = \Sigma^c$, $c = O(1)$. Let $S \subseteq [u]$ be given of size $n = |S|$ and assign to each key/ball $x \in S$ a weight $w_x \in [0, 1]$. Let $y \in [m]$, and define $X = \sum_{x \in S} w_x \cdot [h(x) = y]$ to*

be the total weight of the balls hashing to bin y . Then for any constant $\gamma > 0$, X is strongly concentrated with added error probability n/m^γ , where the constants of the asymptotics are determined solely by c and γ . Furthermore, this concentration is query invariant.

In Theorem 1.1, we note that the expectation, $\mu = \mathbb{E}[X]$, and the variance, $\sigma^2 = \text{Var}[X]$, are the same as if h were a fully random hash function since h is 3-independent. This is true even when conditioning on the hash value of a query key having a specific value. The bound provided by Theorem 1.1 is therefore the same as the variance based Chernoff bound (3) except for a constant delay in the exponential decrease and an added error probability of n/m^γ . Since $\sigma^2 \leq \mu$, Theorem 1.1 also implies the classic one-sided Chernoff bounds (1) and (2), again with the constant delay and the added error probability as above, and a leading factor of 2.

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

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

1.2 Tabulation-Permutation Hashing

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

If we precompute tables $T_j: \Sigma \rightarrow \Sigma^d$, where

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

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

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

THEOREM 1.2. *Let $h: [u] \rightarrow [r]$ be a tabulation-permutation hash function with $[u] = \Sigma^c$ and $[r] = \Sigma^d$, $c, d = O(1)$. Let $v: [u] \times [r] \rightarrow [0, 1]$ be a fixed value function that to each key $x \in [u]$ assigns a value $X_x = v(x, h(x)) \in [0, 1]$ depending on the hash value $h(x)$ and define $X = \sum_{x \in [u]} X_x$. For any constant $\gamma > 0$, X is strongly concentrated with*

added error probability $1/u^\gamma$, where the constants of the asymptotics are determined solely by c , d , and γ . Furthermore, this concentration is query invariant.

Tabulation-permutation hashing inherits the 3-independence of simple tabulation, so as in Theorem 1.1, $\mu = \mathbb{E}[X]$ and $\sigma^2 = \text{Var}[X]$ have exactly the same values as if h were a fully-random hash function. Again, this is true even when conditioning on the hash value of a query key having a specific value.

Tabulation-permutation hashing allows us to hash into m bins for any $m \in \mathbb{N}$ (not necessarily a power of two) preserving the strong concentration from Theorem 1.2. To do so, simply define the hash function $h^m: [u] \rightarrow [m]$ by $h^m(x) = h(x) \bmod m$. Relating back to Theorem 1.1, consider a set $S \subseteq U$ of n balls where each ball $x \in S$ has a weight $w_x \in [0, 1]$ and balls x outside S are defined to have weight $w_x = 0$. To measure the total weight of the balls landing in a given bin $y \in [m]$, we define the value function $v(x, z) = w_x \cdot [z \bmod m = y]$. Then

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

is exactly the desired quantity and we get the concentration bound from Theorem 1.2. Then the big advantage of tabulation-permutation hashing over simple tabulation hashing is that it reduces the added error probability from n/m^γ of Theorem 1.1 to the $1/u^\gamma$ of Theorem 1.2, where u is the size of the key universe. Thus, with tabulation-permutation hashing, we actually get Chernoff bounds with high probability regardless of the number of bins.

Pătraşcu and Thorup [41] introduced twisted tabulation that like our tabulation-permutation achieved Chernoff-style concentration bounds with a general value function v . Their bounds are equivalent to those of Theorem 1.2, but only under the restriction $\mu \leq |\Sigma|^{1-\Omega(1)}$. To understand how serious this restriction is, consider again tossing an unbiased coin for each key x in a set $S \subseteq [u]$, corresponding to the case $m = 2$ and $\mu = |S|/2$. With the restriction from [41], we can only handle $|S| \leq 2|\Sigma|^{1-\Omega(1)}$, but recall that Σ is chosen small enough for character tables to fit in fast cache, so this rules out any moderately large data set. We are going to show that for certain sets S , twisted tabulation has the same problems as simple tabulation when hashing to few bins. This implies that the restrictions from [41] cannot be lifted with a better analysis.

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

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

1.3 Tabulation-1Permutation

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

of the d output characters of a simple tabulation function $g: \Sigma^c \rightarrow \Sigma^d$. Instead, tabulation-1permutation only permutes the most significant character.

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

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

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

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

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

THEOREM 1.3. *Let $h: [u] \rightarrow [r]$ be a tabulation-1permutation hash function with $[u] = \Sigma^c$ and $[r] = \Sigma^d$, $c, d = O(1)$. Consider a key/ball set $S \subseteq [u]$ of size $n = |S|$ where each ball $x \in S$ is assigned a weight $w_x \in [0, 1]$. Choose arbitrary hash values $y_1, y_2 \in [r]$ with $y_1 \leq y_2$. Define $X = \sum_{x \in S} w_x \cdot \mathbb{1}_{[y_1 \leq h(x) < y_2]}$ to be the total weight of balls hashing to the interval $[y_1, y_2)$. Then for any constant $\gamma > 0$, X is strongly concentrated with added error probability $1/u^\gamma$, where the constants of the asymptotics are determined solely by c, d , and γ . Furthermore, this concentration is query invariant.*

One application of Theorem 1.3 is in the following sampling scenario: We set $y_1 = 0$, and sample all keys with $h(x) < y_2$. Each key is then sampled with probability y_2/r , and Theorem 1.3 gives concentration on the number of samples. In [1] this is used for more efficient implementations of streaming algorithms.

Another application is efficiently hashing into an arbitrary number $m \leq r$ of bins. We previously discussed using hash values modulo m , but a general mod-operation is often quite slow. Instead we can think of hash values as fractions $h(x)/r \in [0, 1)$. Multiplying by m , we get a value in $[0, m)$, and the bin index is then obtained by rounding down to the nearest integer. This implementation is very efficient because r is a power of two, $r = 2^b$, so the rounding is obtained by a right-shift by b bits. To hash a key x to $[m]$, we simply compute $h^m(x) = (h(x) * m) \gg b$. Then x hashes to bin $d \in [m]$ if and only if $d \in [y_1, y_2) \subseteq [r]$ where $y_1 = \lfloor rd/m \rfloor$ and $y_2 = \lfloor r(d+1)/m \rfloor$, so the number of keys hashing to a bin is concentrated as in Theorem 1.3. Moreover, h^m uses only $c + 1$ character lookups and a single multiplication in addition to some very fast shifts and bit-wise Boolean operations.

1.4 Subpolynomial Error Probabilities

In Theorem 1.2 and 1.3, we have $\Pr[|X - \mu| \geq t] = O(\exp(-\Omega(\sigma^2 C(t/\sigma^2)))) + 1/u^\gamma$ which holds for any fixed γ . The value of γ affects the constant hidden in the Ω -notation delaying the exponential decrease. In Section 8, we will show that the same bound does not hold if γ is replaced by any slow-growing but unbounded function. Nevertheless, it follows from our analysis that for every $\alpha(u) = \omega(1)$ there exists $\beta(u) = \omega(1)$ such that whenever $\exp(-\sigma^2 C(t/\sigma^2)) < 1/u^{\alpha(u)}$, $\Pr[|X - \mu| \geq t] \leq 1/u^{\beta(u)}$.

1.5 Generic Remarks on Universe Reduction and Amount of Randomness

The following observations are fairly standard in the literature. Suppose we wish to hash a set of keys S belonging to some universe \mathcal{U} . The universe may be so large compared to S that it is not efficient to directly implement a theoretically

powerful hashing scheme like tabulation-permutation hashing. A standard first step is to perform a *universe reduction*, mapping \mathcal{U} randomly to “signatures” in $[u] = \{0, 1, \dots, u-1\}$, where $u = n^{O(1)}$, e.g. $u = n^3$, so that no two keys from S are expected to get the same signature [9]. As the only theoretical property required for the universe reduction is a low collision probability, this step can be implemented using very simple hash functions as described in [46]. In this paper, we generally assume that this universe reduction has already been done, if needed, hence that we only need to deal with keys from a universe $[u]$ of size polynomial in n . For any small constant $\varepsilon > 0$ we may thus pick $c = O(1/\varepsilon)$ such that the space used for our hash tables, $\Theta(|\Sigma|)$, is $O(n^\varepsilon)$. Practically speaking, this justifies focusing on the hashing of 32- and 64-bit keys.

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

1.6 Techniques

The paper relies on three main technical insights to establish the concentration inequality for tabulation-permutation hashing of Theorem 1.2. We shall here describe each of these ideas and argue that each is in fact necessary towards an efficient hash function with strong concentration bounds.

1.6.1 Improved Analysis of Simple Tabulation. The first step towards proving Theorem 1.2 is to better understand the distribution of simple tabulation hashing. We describe below how an extensive combinatorial analysis makes it possible to prove a generalised version of Theorem 1.1.

To describe the main idea of this technical contribution, we must first introduce some ideas from previous work in the area. This will also serve to highlight the inherent limitations of previous approaches. A simplified account is the following. Let $h: \Sigma^c \rightarrow [m]$ be a simple tabulation hash function, let $y \in [m]$ be given, and for some subset of keys $S \subseteq \Sigma^c$, let $X = \sum_{x \in S} [h(x) = y]$ be the random variable denoting the number of elements $x \in S$ that have hash value $h(x) = y$. Our goal is to bound the deviation of X from its mean $\mu = |S|/m$. We first note that picking a random simple tabulation hash function $h: \Sigma^c \rightarrow [m]$ amounts to filling the c character tables, each of size Σ , with uniformly random hash values. Thus, picking a simple tabulation hash function $h: \Sigma^c \rightarrow [m]$ corresponds to picking a uniformly random hash function $h: [c] \times \Sigma \rightarrow [m]$. We call $[c] \times \Sigma$ the set of *position characters*. Viewing a key $x = (x_1, \dots, x_c) \in \Sigma^c$ as a set of position characters, $x = \{(1, x_1), \dots, (c, x_c)\}$, and slightly abusing notation, it then holds that $h(x) = \bigoplus_{\alpha \in x} h(\alpha)$. Now let $\alpha_1, \dots, \alpha_r$ be a (for the sake of the proof) well-chosen ordering of the position characters. For each $k \in [r+1]$, we define the random variable $X_k = \mathbb{E}[X \mid h(\alpha_1), \dots, h(\alpha_k)]$, where $h(\alpha_i)$ is the value of the entry of the lookup table of h corresponding to α_i . The process $(X_k)_{k=0}^r$ is then a martingale. We can view this as revealing the lookup table of h one entry at a time and adjusting our expectation of the outcome of X accordingly. Defining the martingale difference $Y_k = X_k - X_{k-1}$, we can express X as a sum $X = \mu + \sum_{k=1}^{c \cdot |\Sigma|} Y_k$. Previous work has then bounded the sum using a Chernoff inequality for martingales as follows. Due to the nature of the ordering of $\{\alpha_i\}_{i=1}^r$, we can find $M > 0$ such that with high probability, $|Y_k| \leq M$ for every k . Then conditioned on each of the Y_k s being bounded, X satisfies the

Chernoff bounds of (1) and (2) except the exponent is divided by M . As long as the expectation, μ , satisfies $\mu = O(|\Sigma|)$, it is possible² that $M = O(1)$, yielding Chernoff bounds with a constant in the delay of the exponential decrease. However, since there are only $c \cdot |\Sigma|$ variables, Y_k , it is clear that $M \geq \mu/(c \cdot |\Sigma|)$. Thus, whenever $\mu = \omega(|\Sigma|)$, the delay of the exponential decrease is super-constant, meaning that we do not get asymptotically tight Chernoff-style bounds. This obstacle has been an inherent issue with the previous techniques in analysing both simple tabulation [38] as well as twisted tabulation [41]. Being unable to bound anything beyond the absolute deviation of each variable Y_k , it is impossible to get good concentration bounds for large expectations, μ .

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

An extensive combinatorial analysis of simple tabulation hashing yields high-probability bounds on the sum of variances that is tight up to constant factors. This is key in establishing an induction that allows us to prove Theorem 1.1. Complementing our improved bounds, we will show that simple tabulation hashing inherently does not support Chernoff-style concentration bounds for small m .

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

Let us look at the individual steps of h^m . First, we use simple tabulation mapping into the “character bins”, Σ . The number of balls in any given character bin is nicely concentrated, but only because $|\Sigma|$ is large. Next, perform a permutation followed by the mod m operation. The last two steps correspond to the way we would deal a deck of $|\Sigma|$ cards into m hands. The cards are shuffled by a random permutation, then dealt to the m players one card at a time in cyclic order. The end result is that each of the final m bins is assigned exactly $|\Sigma|/m$ random character bins. An important point is now that because the partitioning is *exact*, the error in the number of balls in a final bin stems solely from the errors in the $|\Sigma|/m$ character bins, and because the partitioning is *random*, we expect the positive and negative errors to cancel out nicely. The analysis, which is far from trivial, requires much more than these properties. For example, we also need the bound described in Section 1.6.1 on the sum of variances. This bound ensures that not only is the number of balls in the individual character bins nicely concentrated around the mean, but moreover, there is only a small number of character bins for which the error is large. That these things combine to yield strong concentration, not only in the specific example above, but for general value functions as in Theorem 1.2, is quite magical.

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

First, one may ask why we need the permutation at all. After all, the mod m operation also partitions the $|\Sigma|$ character bins into groups of the same size, $|\Sigma|/m$. The issue is that while a simple tabulation hash function, $g: \Sigma^c \rightarrow \Sigma$, has good concentration in each of the individual character bins, the $|\Sigma|/m$ character bins being picked out by the mod m

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

operation constitute a very structured subset of Σ , and the errors from this set of bins could be highly correlated. We indeed show that the structure of simple tabulation causes this to happen for certain sets of keys, both theoretically (Section 8) and experimentally (Appendix A).

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

1.6.3 Squaring the Hash Range. The last piece of the puzzle is a trick to extend the range of a hash function satisfying Chernoff-style bounds. We wish to construct a hash function $h: \Sigma^c \rightarrow [m]$ satisfying Chernoff-style bounds for m arbitrarily large as in Theorem 1.2. At first sight, the trick of the previous subsection would appear to suffice for the purpose. However, if we let $g = \tau \circ h$ be the composition of a simple tabulation hash function $h: \Sigma^c \rightarrow [m]$ and τ a random permutation of $[m]$, we run into trouble if for instance $[m] = \Sigma^c$. In this case, a random permutation of $[m]$ would require space equal to that of a fully random function $f: \Sigma^c \rightarrow [m]$, but the whole point of hashing is to use less space. Hence, we instead prove the following. Let $a: C \rightarrow D$ and $b: C \rightarrow D$ be two independent hash functions satisfying Chernoff-style bounds for general value functions. Then this property is preserved up to constant factors under “concatenation”, i.e., if we let $c: C \rightarrow D^2$ be given by $c(x) = (a(x), b(x))$, then c is also a hash function satisfying Chernoff-style bounds for general value functions, albeit with a slightly worse constant delay in the exponential decrease than a and b . Thus, this technique allows us to “square” the range of a hash function.

With this at hand, let $h_1, h_2: \Sigma^c \rightarrow \Sigma$ be defined as $h_1 = \tau_1 \circ g_1$ and $h_2 = \tau_2 \circ g_2$, where $g_1, g_2: \Sigma^c \rightarrow \Sigma$ are simple tabulation hash functions and $\tau_1, \tau_2: \Sigma \rightarrow \Sigma$ are random permutations. Then the concatenation $h: \Sigma^c \rightarrow \Sigma^2$ of h_1 and h_2 can be considered a composition of a simple tabulation function $g: \Sigma^c \rightarrow \Sigma^2$ given by $g(x) = (g_1(x), g_2(x))$ and a coordinate-wise permutation $\tau = (\tau_1, \tau_2): \Sigma^2 \rightarrow \Sigma^2$, where the latter is given by $\tau(x_1, x_2) = (\tau_1(x_1), \tau_2(x_2))$, $x_1, x_2 \in \Sigma$. Applying our composition result, gives that g also satisfies Chernoff-style bounds. Repeating this procedure $\lceil \log(d) \rceil = O(1)$ times, yields the desired concentration bound for tabulation-permutation hashing $h: \Sigma^c \rightarrow \Sigma^d$ described in Theorem 1.2.

1.7 Related Work – Theoretical and Experimental Comparisons

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

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

as in Definition 1 and Theorems 1.2 and 1.3, or possibly with σ^2 replaced by $\mu \geq \sigma^2$. Theoretically, we will only compare with other hashing schemes that are relevant to this goal. In doing so, we distinguish between the hash functions that

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

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

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

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

achieve Chernoff-style bounds with restrictions on the expected value and those that, like our new hash functions, do so without such restrictions, which is what we want for all possible input. An overview of the theoretical guarantees of the hash functions relevant to our discussion is presented in Table 2. Empirically, we shall compare the practical evaluation time of tabulation-permutation and permutation-1permutation to the fastest commonly used hash functions and to hash functions with similar theoretical guarantees. A major goal of algorithmic analysis is to understand the theoretical behavior of simple algorithms that work well in practice, providing them with good theoretical guarantees such as worst-case behavior. For instance, one may recall Knuth’s analysis of linear probing [29], showing that this very practical algorithm has strong theoretical guarantees. In a similar vein, we not only show that the hashing schemes of tabulation-permutation and tabulation-1permutation have strong theoretical guarantees, we also perform experiments, summarized in Table 1, demonstrating that in practice they are comparable in speed to some of the most efficient hash functions in use, none of which have similar concentration guarantees. Thus, with our new hash functions, hashing with strong theoretical concentration guarantees is suddenly feasible for time-critical applications.

1.7.1 High Independence and Tabulation. Before this paper, the only known way to obtain unrestricted Chernoff-style concentration bounds with hash functions that can be evaluated in constant time was through k -independent hashing.

Recall that a hash function $h : U \rightarrow R$ is k -independent if the distribution of $(h(x_1), \dots, h(x_k))$ is uniform in R^k for every choice of distinct keys $x_1, \dots, x_k \in U$. Schmidt, Siegel, and Srinivasan [43] have shown that with k -independent hashing, we have Chernoff-style concentration bounds in all three cases mentioned at the beginning of the introduction up to an added error probability decreasing exponentially in k . With $k = \Theta(\gamma \log u)$, this means Chernoff-style concentration with an added error probability of $1/u^\gamma$ like in Theorem 1.2 and 1.3. However, evaluating any k -independent hash function takes time $\Omega(k)$ unless we use a lot of space. Indeed, a cell probe lower bound by Siegel [44] states that evaluating a k -independent hash function over a key domain $[u]$ using $t < k$ probes, requires us to use at least $u^{1/t}$ cells to represent the hash function. Thus, aiming for Chernoff concentration through k -independence with $k = \Omega(\log u)$ and with constant evaluation time, we would have to use $u^{\Omega(1)}$ space like our tabulation-permutation. Here it should be mentioned that k -independent PolyHash modulo a prime p can be evaluated at k points in total time $O(k \log^2 k)$ using multipoint evaluation methods. Then the average evaluation time is $O(\log^2 k)$, but it requires that the hashing can be done to batches of k keys at a time. We can no longer hash one key at a time, continuing with other code before we hash the next key. This could be a problem for some applications. A bigger practical issue is that it is no longer a black box implementation of a hash function. To understand the issue, think of Google's codebase where thousands of programs are making library calls to hash functions. A change to multipoint evaluation would require rewriting all of the calling programs, checking in each case that batch hashing suffices — a huge task that likely would create many errors. A final point is that multipoint evaluation is complicated to implement yet still not as fast as our tabulation-permutation hashing. Turning to upper bounds, Siegel designed a $u^{\Omega(1/c^2)}$ -independent hash function that can be represented in tables of size $u^{1/c}$ and evaluated in $c^{O(c)}$ time. With $c = O(1)$, this suffices for Chernoff-style concentration bounds by the argument above. However, as Siegel states, the hashing scheme is “far too slow for any practical application”.

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

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

With Thorup's double tabulation, for $(\log u)$ -independence, we need $\log u \leq |\Sigma|^{1/(5c)} = u^{1/(5c^2)}$. In choosing values for u and c that work in practice, this inequality is very restrictive. Indeed, even for $c = 2$, $\log u \leq u^{1/20}$, which roughly

implies that $\log u \geq 140$. Combined with the fact that the character tables use space $12c|\Sigma|$, and that $|\Sigma| \geq (\log u)^{5c}$, this is an intimidating amount of space. Another problem is the error probability over h_1 of $1 - o(\Sigma^{2-d/(2c)})$. If we want this to be $O(1/u)$, like in the error bounds from Theorem 1.2 and 1.3, we need $d \geq 2(c^2 + 2c)$. Thus, while things work well asymptotically, these constraints make it hard to implement highly independent double tabulation on any normal computer. However, based on a more careful analysis of the case with 32-bit keys, Thorup shows that using $c = 2$ characters of 16 bits, and $d = 20$ derived characters, gives a 100-independent hash function with probability $1 - 1.5 \times 10^{-42}$. According to [45] we cannot use significantly fewer resources even if we just want 4-independence. For hashing 32-bit keys, this means making 22 lookups for each query and using tables of total size $40 \cdot 2^{16}$. In contrast, if we hash 32-bit keys with tabulation-permutation hashing, we may use 8-bit characters with $d = c = 4$, thus making only 8 lookups in tables of total size $8 \cdot 2^8$. For this setting of parameters, our experiments (summarized in Table 1) show that double tabulation is approximately 30 times slower than tabulation-permutation hashing. For 64-bit keys, Thorup [45] suggests implementing double tabulation with $c = 3$ characters of 22 bits and $d = 24$. This would require 26 lookups in tables of total size $48 \cdot 2^{22}$. We were not able to implement this on a regular laptop due to the space requirement.

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

1.7.2 Space Bounded Independence and Chernoff Bounds. One of the earliest attempts of obtaining strong concentration bounds via hashing is a simple and elegant construction by Dietzfelbinger and Meyer auf der Heide [19]. For some parameters m, s, d , their hash family maps to $[m]$, can be represented with $O(s)$ space, and uses a $(d + 1)$ -independent hash function as a subroutine, where $d = O(1)$, e.g., a degree- d polynomial. In terms of our main goal of Chernoff-style bounds, their result can be cast as follows: Considering the number of balls from a fixed, but unknown, subset $S \subseteq U$, with $|S| = n$, that hashes to a specific bin, their result yields Chernoff bounds like ours with a constant delay in the exponential decrease and with an added error probability of $n \left(\frac{n}{ms}\right)^d$. The expected number of balls in a given bin is $\mu = n/m$, so the added error probability is $n(\mu/s)^d$. To compare with tabulation-permutation, suppose we insist on using space $O(|\Sigma|)$ and that we moreover want the added error probability to be $u^{-\gamma} = |\Sigma|^{-c\gamma}$ like in Theorems 1.2 and 1.3. With the hashing scheme from [19], we then need $\mu = O(|\Sigma|^{1-\gamma c/d})$. If we want to be able to handle expectations of order, e.g. $|\Sigma|^{1/2}$, we thus need $d \geq 2c\gamma$. For 64-bit key, $c = 8$, and $\gamma = 1$, say, this means that we need to evaluate a 16-independent hash function. In general, we see that the concentration bound above suffers from the same issues as those provided by Pătraşcu and Thorup for simple and twisted tabulation [38, 41], namely that we only have Chernoff-style concentration if the expected value is much smaller than the space used.

Going in a different direction, Dietzfelbinger and Rink [20] use universe splitting to create a hash function that is highly independent (building on previous works [21, 22, 25, 27]) but, contrasting double tabulation as described above, only within a fixed set S , not the entire universe. The construction requires an upper bound n on the size of S , and a polynomial error probability of $n^{-\gamma}$ is tolerated. Here $\gamma = O(1)$ is part of the construction and affects the evaluation time. Assuming no such error has occurred, which is not checked, the hash function is highly independent on S . As with Siegel's and Thorup's highly independent hashing discussed above, this implies Chernoff bounds without the constant delay in the exponential decrease, but this time only within the fixed set S . In the same setting, Pagh and Pagh [37] have presented a hash function that uses $(1 + o(1))n$ space and which is fully independent on any given set S of size at most n with high probability. This result is very useful, e.g., as part of solving a static problem of size n using linear space, since, with high probability, we may assume fully-random hashing as a subroutine. However, from a Chernoff bound perspective, the fixed polynomial error probability implies that we do not benefit from any independence above

$O(\log n)$, using the aforementioned results from [43]. More importantly, we do not want to impose any limitations to the size of the sets we wish to hash in this paper. Consider for example the classic problem of counting distinct elements in a huge data stream. The size of the data stream might be very large, but regardless, the hashing schemes of this paper will only use space $O(u^{1/c})$ with c chosen small enough for hash tables to fit in fast cache.

Finally, Dahlgaard et al. [16] have shown that on a given set S of size $|S| \leq |\Sigma|/2$ a double tabulation hash function, $h = h_2 \circ h_1$ as described above, is fully random with probability $1 - |\Sigma|^{1-\lfloor d/2 \rfloor}$ over the choice of h_1 . For an error probability of $1/u$, we set $d = (2c + 2)$ yielding a hash function that can be evaluated with $3c + 2$ character lookups and using $(4c + 4)|\Sigma|$ space. This can be used to simplify the above construction by Pagh and Pagh [37]. Dahlgaard et al. [16] also propose mixed tabulation hashing which they use for statistics over k -partitions. Their analysis is easily modified to yield Chernoff-style bounds for intervals similar to our bounds for tabulation-1permutation hashing presented in Theorem 1.3, but with the restriction that the expectation μ is at most $|\Sigma|/\log^{2c} |\Sigma|$. This restriction is better than the earlier mentioned restrictions $\mu \leq |\Sigma|^{1/2}$ for simple tabulation [38] and $\mu \leq |\Sigma|^{1-\Omega(1)}$ for twisted tabulation [41]. For mixed tabulation hashing, Dahlgaard et al. use $3c + 2$ lookups and $(5c + 4)|\Sigma|$ space. In comparison, tabulation-1permutation hashing, which has no restriction on μ , uses only $c + 1$ lookups and $(c + 1)|\Sigma|$ space.

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

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

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

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

Concerning weaker, but potentially faster, hashing schemes we have chosen two types of hash functions for the comparison. First, we have the fast 2-independent hash functions multiply-shift (with addition) and 2-independent PolyHash. They are among the fastest hash functions in use and are commonly used in streaming algorithms. It should

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

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

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

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

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

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

Theoretically, tabulation-based hashing methods are incomparable in speed to multiply-shift and 2-independent PolyHash, since the latter methods use constant space but multiplication which has circuit complexity $\Theta(\log w / \log \log w)$

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

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

Mixed Tabulation. We implemented mixed tabulation hashing with the same parameters ($c = d$) as in [17]. With these parameters the scheme uses $2c$ lookups from $2c$ character tables, where c of the lookups are to table entries that are double as long as the output, which may explain its worse performance with 64-bit domains. In our experiments, mixed tabulation performs slightly worse than tabulation-permutation hashing. Recall from above that mixed tabulation is faster than many popular hash functions without theoretical guarantees, hence so is our tabulation-permutation.

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

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

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

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

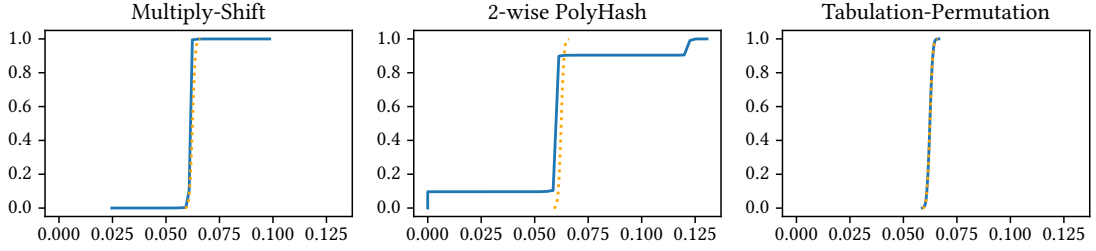


Fig. 1. Hashing the arithmetic progression $\{a \cdot i \mid i \in [50000]\}$ to 16 bins for a random integer a . The dotted line is a 100-independent PolyHash.

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

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

2 TECHNICAL THEOREMS AND HOW THEY COMBINE

We now formally state our main technical results, in their full generality, and show how they combine to yield Theorems 1.1, 1.2, and 1.3. A fair warning should be given to the reader. The theorems to follow are intricate and arguably somewhat inaccessible at first read. Rather than trying to understand everything at once, we suggest that the reader use this section as a roadmap for the main body of the paper. We will, however, do our best to explain the contents of the results as well as disentangling the various assumptions in the theorems.

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

It turns out that the proofs of our results become a little cleaner when we assume that value functions take values in $[-1, 1]$, so from here on we state our results in relation to such value functions. Theorems 1.1, 1.2, and 1.3 will still follow, as the value functions in these theorems can also be viewed as having range $[-1, 1]$.

2.1 Improved Analysis of Simple Tabulation

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

THEOREM 2.1. *Let $h: \Sigma^c \rightarrow [m]$ be a simple tabulation hash function and $S \subseteq \Sigma^c$ be a set of keys of size $n = |S|$. Let $v: \Sigma^c \times [m] \rightarrow [-1, 1]$ be a value function such that the set $Q = \{i \in [m] \mid \exists x \in \Sigma^c : v(x, i) \neq 0\}$ satisfies $|Q| \leq m^\epsilon$, where $\epsilon < \frac{1}{4}$ is a constant.*

- (1) *For any constant $\gamma \geq 1$, the random variable $V = \sum_{x \in S} v(x, h(x))$ is strongly concentrated with added error probability $O_{\gamma, \epsilon, c}(n/m^\gamma)$, where the constants of the asymptotics are determined by c and γ . Furthermore, this concentration is query invariant.*
- (2) *For $j \in [m]$ define the random variable $V_j = \sum_{x \in S} v(x, h(x) \oplus j)$ and let $\mu = \mathbb{E}[V_j]$, noting that this is independent of j . For any $\gamma \geq 1$,*

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

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

The technical assumption involving Q states that the value function has *bounded support* in the hash range: The value $v(x, h(x))$ can only possibly be non-zero if $h(x)$ lies in the relatively small set Q of size at most m^ϵ . In fact, when proving Theorem 1.1 it suffices to assume that $|Q| = 1$, as we shall see below, but for our analysis of tabulation-permutation hashing we need the more general result above. Another nice illustration of the power of Theorem 2.1 holding with value functions of *any* bounded support will appear when we prove Theorem 1.3 in Section 2.4.

To see that Theorem 1.1 is implied by Theorem 2.1, one may observe that the latter is a generalization of the former. Let $y \in [m]$ be the bin and $(w_x)_{x \in S}$ be the weights of the balls from S in the setting of Theorem 1.1. Then defining the value function $v: \Sigma^c \times [m] \rightarrow [0, 1]$,

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

we find that $X = \sum_{x \in S} w_x \cdot [h(x) = y] = \sum_{x \in S} v(x, h(x))$ is strongly concentrated by part 1 of Theorem 2.1 and the concentration is query invariant.

Finally, the bound (5) requires some explaining. For this, we consider the toy example of Theorem 1.1. Suppose we have a set $S \subseteq [u]$ of balls with weights $(w_x)_{x \in S}$ and we throw them into the bins of $[m]$ using a simple tabulation hash function. We focus on the total weight of balls landing in bin 0, defining the value function by $v(x, y) = w_x$ for $x \in S$ and $y = 0$, and $v(x, y) = 0$ otherwise. In this case, $\mu = \frac{1}{m} \sum_{x \in S} w_x$ denotes the expected total weight in any single bin and $V_j = \sum_{x \in S} w_x \cdot [h(x) = j]$ denotes the total weight in bin $j \in [m]$. Then (5) states that $\sum_{j \in [m]} (V_j - \mu)^2 = O(\|w\|_2^2)$ with high probability in m . This is exactly a bound on the *variance* of the weight of balls landing in one of the bins when each of the hash values of the keys of S are shifted by an XOR with a uniformly random element of $[m]$. Note that this example corresponds to the case where $|Q| = 1$. In its full generality, i.e., for general value functions of bounded support, (5) is similarly a bound on the variance of the value obtained from the keys of S when their hash values are

each shifted by a uniformly random XOR. This variance bound turns out to be an important ingredient in our proof of the strong concentration in the first part of Theorem 2.1. As described in Section 1.6.1 the proof proceeds by fixing the hash values of the position characters $[c] \times \Sigma$ in a carefully chosen order, $\alpha_1 < \dots < \alpha_r$. Defining G_i to be those keys that contain α_i as a position character but no α_j with $j > i$, the internal clustering of the keys of G_i is determined solely by $(h(\alpha_j))_{j < i}$ and fixing $h(\alpha_i)$ “shifts” each of these keys by an XOR with $h(\alpha_i)$. Now (5), applied with $S = G_i$, exactly yields a bound on the variance of the total value obtained from the keys from G_i when fixing the random XOR $h(\alpha_i)$. Thus, (5) conveniently bounds the variance of the martingale described in Section 1.6.1. As such, (5) is merely a technical tool, but we have a more important reason for including the bound in the theorem. As it turns out, for any hash function satisfying the conclusion of Theorem 2.1, composing with a uniformly random permutation yields a hash family having Chernoff-style concentration bounds for any value function as we describe next.

2.2 Permuting the Hash Range

Our next step in proving Theorem 1.2 is to show that, given a hash function with concentration bounds like in Theorem 2.1, composing with a uniformly random permutation of the entire range yields a hash function with Chernoff-style concentration for general value functions. The main theorem, proved in Section 5, is as follows.

THEOREM 2.2. *Let $\varepsilon \in (0, 1]$ and $m \geq 2$ be given. Let $g: [u] \rightarrow [m]$ be a 3-independent hash function satisfying the following. For every $\gamma > 0$, and for every value function $v: [u] \times [m] \rightarrow [-1, 1]$ such that the set $Q = \{i \in [m] \mid \exists x \in [u]: v(x, i) \neq 0\}$ is of size $|Q| \leq m^\varepsilon$, the two conclusions of Theorem 2.1 holds with respect to g .*

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

We believe the theorem to be of independent interest. From a hash function that only performs well for value functions supported on an asymptotically small subset of the bins we can construct a hash function performing well for any value function – simply by composing with a random permutation. Theorem 2.1 shows that simple tabulation satisfies the two conditions in the theorem above. It follows that if $m = |U|^{\Omega(1)}$, e.g., if $m = |\Sigma|$, then composing a simple tabulation hash function $g: \Sigma^c \rightarrow [m]$ with a uniformly random permutation $\tau: [m] \rightarrow [m]$ yields a hash function $h = \tau \circ g$ having Chernoff-style bounds for general value functions asymptotically matching those from the fully random setting up to an added error probability inversely polynomial in the size of the universe. In particular these bounds hold for tabulation-permutation hashing from Σ^c to Σ , that is, using just a single permutation, which yields the result of Theorem 1.2 in the case $d = 1$. If we desire a range of size $m \gg |\Sigma|$ the permutation τ becomes too expensive to store. Recall that in tabulation-permutation hashing from Σ^c to Σ^d we instead use d permutations $\tau_1, \dots, \tau_d: \Sigma \rightarrow \Sigma$, hashing

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

Towards proving that this is sensible, the last step in the proof of Theorem 1.2 is to show that concatenating the outputs of independent hash functions preserves the property of having Chernoff-style concentration for general value functions.

2.3 Squaring the Hash Range

The third and final step towards proving Theorem 1.2 is showing that concatenating the hash values of two independent hash functions each with Chernoff-style bounds for general value functions yields a new hash function with similar Chernoff-style bounds up to constant factors. In particular it will follow that tabulation-permutation hashing has Chernoff-style bounds for general value functions. However, as with Theorem 2.2, the result is of more general interest. Since it uses the input hash functions in a black box manner, it is a general tool towards constructing new hash functions with Chernoff-style bounds. The main theorem, proved in Section 6, is the following.

THEOREM 2.3. *Let $h_1: A \rightarrow B_1$ and $h_2: A \rightarrow B_2$ be 2-wise independent hash functions with a common domain such that for every pair of value functions, $v_1: A \times B_1 \rightarrow [-1, 1]$ and $v_2: A \times B_2 \rightarrow [-1, 1]$, the random variables $X_1 = \sum_{a \in A} v_1(a, h_1(a))$ and $X_2 = \sum_{a \in A} v_2(a, h_2(a))$ are strongly concentrated with added error probability f_1 and f_2 , respectively, and the concentration is query invariant. Suppose further that h_1 and h_2 are independent. Then the hash function $h = (h_1, h_2): A \rightarrow B_1 \times B_2$, which is the concatenation of h_1 and h_2 , satisfies that for every value function $v: A \times (B_1 \times B_2) \rightarrow [-1, 1]$, the random variable $X = \sum_{a \in A} v(a, h(a)) = \sum_{a \in A} v(a, h_1(a), h_2(a))$ is strongly concentrated with additive error $O(f_1 + f_2)$ and the concentration is query invariant.*

We argue that Theorem 2.3, combined with the previous results, leads to Theorem 1.2.

PROOF OF THEOREM 1.2. We proceed by induction on d . For $d = 1$ the result follows from Theorem 2.1 and 2.2 as described in the previous subsection. Now suppose $d > 1$ and that the result holds for smaller values of d . Let $\gamma = O(1)$ be given. Let $d_1 = \lfloor d/2 \rfloor$ and $d_2 = \lceil d/2 \rceil$. A tabulation-permutation hash function $h: \Sigma^c \rightarrow \Sigma^d$ is the concatenation of two independent tabulation-permutation hash functions $h_1: \Sigma^c \rightarrow \Sigma^{d_1}$ and $h_2: \Sigma^c \rightarrow \Sigma^{d_2}$. Letting $A = \Sigma^c$, $B_1 = \Sigma^{d_1}$, $B_2 = \Sigma^{d_2}$, the induction hypothesis gives that the conditions of Theorem 2.3 are satisfied and the conclusion follows. Note that since $d = O(1)$, the induction is only applied a constant number of times. Hence, the constants hidden in the asymptotics of Definition 1 are still constant. \square

2.4 Concentration in Arbitrary Intervals.

We will now show how we can use our main result, Theorem 1.2, together with our improved understanding of simple tabulation Theorem 2.1 to obtain Theorem 1.3 which shows that the extra efficient tabulation-1permutation hashing provides Chernoff-style concentration for the special case of weighted balls and intervals. This section also serves as an illustration of how our previous results play in tandem, and it illustrates the importance of Theorem 2.1 holding, not just for single bins, but for any value function of bounded support.

PROOF OF THEOREM 1.3. Let $S \subseteq [u]$ be a set of keys, with each key $x \in S$ having a weight $w_x \in [0, 1]$. Let $h = \tau \circ g: \Sigma^c \rightarrow \Sigma^d = [r]$ be a tabulation-1permutation hash function, with $g: \Sigma^c \rightarrow \Sigma^d$ a simple tabulation hash function and $\tau: \Sigma^d \rightarrow \Sigma^d$ a random permutation of the most significant character, $\tau(z_1, \dots, z_d) = (\tau_1(z_1), z_2, \dots, z_d)$ for a uniformly random permutation $\tau_1: \Sigma \rightarrow \Sigma$. Let $y_1, y_2 \in \Sigma^d$ and X be defined as in Theorem 1.3, $X = \sum_{x \in S} w_x \cdot [y_1 \leq h(x) < y_2]$. Set $\mu = \mathbb{E}[X]$, and $\sigma^2 = \text{Var}[X]$. For simplicity we assume that $|I| \geq r/2$. Otherwise, we just apply the argument below with I replaced by $[r] \setminus I = [0, y_1) \cup [y_2, r)$, which we view as an interval in the cyclic ordering of $[r]$. We will partition $I = [y_1, y_2)$ into a constant number of intervals in such a way that our previous results yield Chernoff style concentration bound on the total weight of keys landing within each of these intervals. The desired result will follow.

To be precise, let $t > 0$ and $\gamma = O(1)$ be given. Let $P_1 = \{x \in \Sigma \mid \forall y \in \Sigma^{d-1} : (x, y) \in I\}$ and $I_1 = \{(x_1, \dots, x_d) \in \Sigma^d \mid x_1 \in P_1\}$. Whether or not $h(x) \in I_1$ for a key $x \in \Sigma^c$ depends solely on the most significant character of $h(x)$. With $X_1 = \sum_{x \in S} w_x \cdot [h(x) \in I_1]$, $\mu_1 = \mathbb{E}[X_1]$, and $\sigma_1^2 = \text{Var}[X_1]$, we can therefore apply Theorem 1.2 to obtain that for any $t' > 0$ and $\gamma' = O(1)$,

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

for some constant C . Here we used that $\sigma_1^2 \leq \sigma^2$ as $|I_1| \leq |I| \leq |\Sigma|^d/2$. Next, let $d_1 = \lg |\Sigma|$ and $d_2, \dots, d_\ell \in \mathbb{N}$ be such that for $2 \leq i \leq \ell$, it holds that $2^{d_i} \leq (2^{d_1+d_2+\dots+d_i})^{1/4}$, and further $2^{d_1+d_2+\dots+d_\ell} = |\Sigma|^d$. We may assume that u and hence $|\Sigma|$ is larger than some constant as otherwise the bound in Theorem 1.3 is trivial. It is then easy to check that we may choose ℓ and the $(d_i)_{2 \leq i \leq \ell}$ such that $\ell = O(\log d) = O(1)$. We will from now on consider elements of Σ^d as numbers written in binary or, equivalently, bit strings of length $d' := d_1 + \dots + d_\ell$. For $i = 1, \dots, \ell$ we define a map $\rho_i : \Sigma^d \rightarrow [2]^{d_1+\dots+d_i}$ as follows. If $x = b_1 \dots b_{d'} \in [2]^{d'}$, then $\rho_i(x)$ is the length $d_1 + \dots + d_i$ bit string $b_1 \dots b_{d_1+\dots+d_i}$. Set $J_1 = I$. For $i = 2, \dots, \ell$ we define $J_i \subseteq I$ and $I_i \subseteq I$ recursively as follows. First, we let $J_i = J_{i-1} \setminus I_{i-1}$. Second, we define I_i to consist of those elements of $x \in J_i$ such that if $y \in \Sigma^c$ has $\rho_i(y) = \rho_i(x)$, then $y \in J_i$. In other words, I_i consists of those elements of J_i that remain in J_i when the least significant $d_{i+1} + \dots + d_\ell$ bits of x are changed in an arbitrary manner. It is readily checked that for $i = 1, \dots, \ell$, I_i is a disjoint union of two (potentially empty) intervals $I_i = I_i^{(1)} \cup I_i^{(2)}$ such that for each $j \in \{1, 2\}$ and $x, y \in I_i^{(j)}$, $\rho_i(x) = \rho_i(y)$. Moreover, the sets $(I_i)_{i=1}^\ell$ are pairwise disjoint and $I = \bigcup_{i=1}^\ell I_i$.

We already saw in (6) that we have Chernoff-style concentration for the total weight of balls landing in I_1 . We now show that the same is true for $I_i^{(j)}$ for each $i = 2, \dots, \ell$ and $j \in \{0, 1\}$. So let such an i and j be fixed. Note that whether or not $h(x) \in I_i^{(j)}$, for a key $x \in \Sigma^c$, depends solely on the most significant $d_1 + \dots + d_i$ bits of $h(x)$. Let $h' : \Sigma^c \rightarrow [2]^{d_1+\dots+d_i}$ be defined by $h'(x) = \rho_i(h(x))$. Then h' is itself a simple tabulation hash function and $h'(x)$ is obtained by removing the $d_{i+1} + \dots + d_\ell$ least significant bits of $h(x)$. Letting $I' = \rho_i(I_i^{(j)})$, it thus holds that $h(x) \in I_i^{(j)}$ if and only if $h'(x) \in I'$. Let now $X_i^{(j)} = \sum_{x \in S} w_x \cdot [h(x) \in I_i^{(j)}]$, $\mu_i^{(j)} = \mathbb{E}[X_i^{(j)}]$, and $\sigma_1^2 = \text{Var}[X_i^{(j)}] \leq \sigma^2$. As $|I'| \leq 2^{d_i} \leq (2^{d_1+\dots+d_i})^{1/4}$, we can apply Theorem 2.1 to conclude that for $t' > 0$ and $\gamma' = O(1)$,

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

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

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

as desired. \square

3 PRELIMINARIES

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

3.1 Notation

Throughout the paper, we use the following general notation.

- We let $[n]$ denote the set $\{0, 1, \dots, n-1\}$.

- For a statement or event Q we let $[Q]$ be the indicator variable on Q , i.e.,

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

- Whenever $Y_0, \dots, Y_{n-1} \in \mathbb{R}$ are variables and $i \in [n+1]$, we shall denote by $Y_{<i}$ the sum $\sum_{j<i} Y_j$. Likewise, whenever A_0, \dots, A_{n-1} are sets and $i \in [n+1]$, we shall denote by $A_{<i}$ the set $\bigcup_{j<i} A_j$.
- Suppose we have a hash function $h: A \rightarrow B$ with domain A and range B . We shall often associate weight and value functions with h as follows.

- A function $w: A \rightarrow \mathbb{R}$ is called a *weight function*, corresponding to the idea that every ball or key $x \in A$ has an associated weight, $w(x) \in \mathbb{R}$. Occasionally, we shall write w_x for $w(x)$.
- A function $v: A \times B \rightarrow \mathbb{R}$ is called a *value function*, with the interpretation that a key $x \in A$ yields a value $v(x, h(x))$ depending on the bin/hash value $h(x) \in B$.

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

3.2 Probability Theory and Martingales

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

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

- Let $X_1, \dots, X_n: \Omega \rightarrow \mathbb{R}$ be \mathcal{F} -measurable random variables. We denote by $\mathcal{G} = \sigma(X_1, \dots, X_n) \subset \mathcal{F}$ the smallest σ -algebra such that X_1, \dots, X_n are all \mathcal{G} -measurable. We say that \mathcal{G} is the *sigma algebra generated by* X_1, \dots, X_n . Intuitively, $\sigma(X_1, \dots, X_n)$ represents the collective information regarding the outcome of the joint distribution (X_1, \dots, X_n) .
- Let $X: \Omega \rightarrow \mathbb{R}$ be an \mathcal{F} -measurable random variable, and let \mathcal{G} be a σ -algebra with $\mathcal{G} \subset \mathcal{F}$. If $\mathbb{E}[|X|] < \infty$, we may define the random variable $\mathbb{E}[X | \mathcal{G}]$ to be the *conditional expectation* of X given \mathcal{G} . It is important to note that $\mathbb{E}[X | \mathcal{G}]$ is \mathcal{G} -measurable. In the context of the above notation, $\mathbb{E}[X | \sigma(X_1, \dots, X_n)] = \mathbb{E}[X | X_1, \dots, X_n]$ is the expectation of X as a function of the outcomes of X_1, \dots, X_n .

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

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

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

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

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

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

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

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

We shall apply this construction in the following situation. Suppose we have random variables Z_1, \dots, Z_r taking values in the measure spaces A_1, \dots, A_r and denote by Z the joint distribution (Z_1, \dots, Z_r) . For some function $f: A_1 \times \dots \times A_r \rightarrow \mathbb{R}$, we wish to assess the value of $f(Z)$. We may then define the filtration $\mathcal{F}_i = \sigma(Z_1, \dots, Z_i)$ for $i \in [r+1]$ and set $X_i = \mathbb{E}[f(Z) | \mathcal{F}_i]$. This yields a martingale $(X_i, \mathcal{F}_i)_{i=0}^r$ with $X_0 = \mathbb{E}[f(Z)]$ and $X_r = f(Z)$. This is known as a Doob martingale and the construction will be used in Section 5 to prove Theorem 2.2.

- Let $(Z_i, \mathcal{F}_i)_{i=0}^r$ be an adapted sequence and define $Y_0 = 0$ and $Y_i = Z_i - \mathbb{E}[Z_i | \mathcal{F}_{i-1}]$ for $i \in \{1, \dots, r\}$. Then $(Y_i, \mathcal{F}_i)_{i=1}^r$ is a martingale difference. This construction is applied in Section 4 to prove Theorem 2.1.

3.3 Martingale Concentration Inequalities

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

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

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

THEOREM 3.1 (FAN ET AL. [24]). Let $\sigma > 0$ be given. Let $(X_i, \mathcal{F}_i)_{i=0}^r$ be a martingale difference such that almost surely $|X_i| \leq 1$ for all $i \in \{1, \dots, r\}$ and $\sum_{i=1}^r \mathbb{E}[X_i^2 | \mathcal{F}_{i-1}] \leq \sigma^2$. Writing $X = \sum_{i=1}^r X_i$, it holds for any $t \geq 0$ that

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

Simple calculations yield the following corollary.

COROLLARY 3.2. *Suppose that $(X_i, \mathcal{F}_i)_{i=0}^r$ is a martingale difference and there exist $M, \sigma \geq 0$ such that $|X_i| \leq M$ for all $i \in \{1, \dots, r\}$ and $\sum_{i=1}^r \mathbb{E}[X_i^2 | \mathcal{F}_{i-1}] \leq \sigma^2$. Define $X = \sum_{i=1}^r X_i$. For any $t \geq 0$ it holds that*

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

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

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

LEMMA 3.3. *For any $x \geq 0$*

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

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

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

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

LEMMA 3.4. *For any $a \geq 0$. If $b \geq 1$ then*

$$bC(a) \leq C(ab) \leq b^2C(a).$$

If $0 \leq b \leq 1$ then

$$b^2C(a) \leq C(ab) \leq bC(a).$$

Note that as a corollary, if $b = \Theta(1)$ and $a \geq 0$, then $C(ba) = \Theta(C(a))$. The final lemma shows that the bound of Corollary 3.2 only gets worse when σ^2 or M is replaced by some larger number.

LEMMA 3.5. *Let $a \geq 0$ be given. On \mathbb{R}^+ , the following two functions are decreasing*

$$\begin{aligned} x &\mapsto xC\left(\frac{a}{x}\right), \\ x &\mapsto \frac{C(ax)}{x^2}. \end{aligned}$$

PROOF. Let $0 < x \leq y$ be given. We then observe that the first function is indeed decreasing since by the first bound of Lemma 3.4, $xC(a/x) = xC((a/y) \cdot (y/x)) \geq yC(a/y)$. That the second function is decreasing follows from a similar argument. \square

4 ANALYSIS OF SIMPLE TABULATION

In this section, we analyze the simple tabulation hashing scheme. The section is divided in three parts. First, there will be an introductory section regarding simple tabulation hashing and associated notation. Second, we shall prove the sum of squares result (Eq. (5)). The final section presents a proof of Theorem 2.1. In order to make the exposition slightly simpler and more accessible, we postpone the argument that our concentration bounds are query invariant to Section 7.

4.1 Simple Tabulation Basics

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

Definition 8 (Simple Tabulation Hashing). Let Σ be an alphabet, $c \geq 1$ an integer, and $m = 2^k$, $k > 0$, a power of two. A simple tabulation hash function, $h: \Sigma^c \rightarrow [m]$, is a random variable taking values in the set of functions from Σ^c to

$[m]$, chosen with respect to the following distribution. For each $j \in \{1, \dots, c\}$, let $h_j: \Sigma \rightarrow [m]$ be a fully random hash function, in other words, a uniformly random function from Σ to $[m]$. We evaluate h on the key $x = (x_1, \dots, x_c) \in \Sigma^c$ by computing $h(x) = \bigoplus_{j=1}^c h_j(x_j)$, where \oplus denotes bitwise XOR.

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

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

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

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

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

LEMMA 4.1 (THORUP AND ZHANG [47]). *Let $h: \Sigma^c \rightarrow [2^r]$ be a simple tabulation hash function. For each $i \in \{1, \dots, t\}$, let $s_i \subseteq \{1, \dots, c\} \times \Sigma$ be a set of position characters of Σ^c . Let $j \in \{1, \dots, t\}$. If every subset of indices $B \subseteq \{1, \dots, t\}$ containing j satisfies $\bigoplus_{i \in B} s_i \neq \emptyset$, then the distribution of $h(s_j)$ is independent of the joint distribution $(h(s_i))_{i \neq j}$.*

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

4.2 Bounding the Sum of Squared Deviations

In the following section we shall prove the bound (5) of Theorem 2.1 from Section 2.1, stated independently here as Theorem 4.5. It is a technical, albeit crucial, step on the way to proving Theorem 2.1 itself. The foundation of the proof of Theorem 4.5 is a series of combinatorial observations regarding simple tabulation hashing.

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

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

Put in other words, let $<$ denote the ordering on A , let x be a key of S , and let β_1, \dots, β_c be the position characters of x such that $\beta_1 < \beta_2 < \dots < \beta_c$, i.e., β_c is last in the ordering of A . Then $x \in G_i$ if and only if $\alpha_i = \beta_c$. In relation to simple tabulation, this has the following meaning. In the proof, we shall fix the values $h(\alpha_j)$ one at a time starting at $j = 1$ and ending at $j = r$. For every $x \in G_i$, the value of $h(x)$ is then undecided before $h(\alpha_i)$ is known, but is known once $h(\alpha_1), \dots, h(\alpha_i)$ are all fixed. In analyzing the contribution of each group to the final outcome of the process, we start by proving a generalization of a result from [38]. It says that if we assign each key a weight, it is always possible to choose the ordering of the position characters such that the total weight of each group is relatively small. The original lemma simply assigned weight 1 to every key.

LEMMA 4.2. *Let $S \subseteq \Sigma^c$ be given and let $A = \{\alpha \in x \mid x \in S\}$ be the position characters of the keys of S . Let $w: \Sigma^c \rightarrow \mathbb{R}_{\geq 0}$ be a weight function. Then there exists an ordering of the position characters, $\{\alpha_1, \dots, \alpha_r\} = A$ such that for every $i \in \{1, \dots, r\}$, the group $G_i = \{x \in S \mid \{\alpha_i\} \subseteq x \subseteq \{\alpha_1, \dots, \alpha_i\}\}$ satisfies*

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

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

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

satisfies

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

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

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

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

□

Suppose we have keys $x_1, \dots, x_t \in \Sigma^c$. It follows as a corollary of Lemma 4.1 that with a simple tabulation hash function $h: \Sigma^c \rightarrow [m]$, the values $h(x_1), \dots, h(x_t)$ are completely independent if and only if there does not exist a subset of indices $B \subseteq \{1, \dots, t\}$ with $\bigoplus_{i \in B} x_i = \emptyset$. In this vein, it turns out to be natural, given sets of keys $A_1, \dots, A_\ell \subseteq \Sigma^c$,

to bound the number of tuples $x_1 \in A_1, \dots, x_\ell \in A_\ell$ with $\bigoplus_{i=1}^{\ell} x_i = \emptyset$. This is the content of Lemma 3 of [16]. We prove the following generalization of this result, which deals with weighted keys. Note that the statements would be identical if each key was assigned the weight 1.

LEMMA 4.3. *Let $\ell \in \mathbb{N}$ be even, $w_1, \dots, w_\ell: \Sigma^c \rightarrow \mathbb{R}$ be weight functions, and $A_1, \dots, A_\ell \subseteq \Sigma^c$ be sets of keys. Then*

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

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

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

We proceed by induction on c .

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

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

Thus, by the Cauchy-Schwartz inequality,

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

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

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

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

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

the induction hypothesis. This yields

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

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

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

which was the desired conclusion. \square

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

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

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

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

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

LEMMA 4.4. *Let $h: \Sigma^c \rightarrow [m]$ be a simple tabulation hash function and $v: \Sigma^c \times [m] \rightarrow \mathbb{R}$ a value function. Let $Q = \{i \in [m] \mid \exists x \in \Sigma^c : v(x, i) \neq 0\}$ be the support of v and write $\ell = |Q|$. Let $n \in \mathbb{N}$ and $A_1, \dots, A_n \subseteq \Sigma^c$. For every*

$i \in \{1, \dots, n\}$ and $j \in [m]$ define the random variable

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

and set

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

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

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

PROOF. We rewrite T as follows

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

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

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

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

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

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

Write $S = A_1 \times \dots \times A_n$ and let $f: S \rightarrow \mathbb{R}$ be the function

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

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

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

Now, for a t -tuple $(s_i)_{i=1}^t \in S^t$, we overload notation by for a subset $T \subseteq \{1, \dots, t\}$ defining $\bigoplus_{i \in T} s_i = \bigoplus_{i \in T} \bigoplus_{j=1}^n (s_i)_j$, where we still think of the keys $(s_i)_j$ as sets of input characters, and where \oplus is the symmetric difference. Let $(s_i)_{i=1}^t \in S^t$ and let $T_1, \dots, T_r \subseteq \{1, \dots, t\}$ be all subsets of indices satisfying $\bigoplus_{i \in T_j} s_i = \emptyset$, $1 \leq j \leq r$. If for some $i \in \{1, \dots, t\}$, $i \notin \bigcup_{j=1}^r T_j$ then by Lemma 4.1, $h(s_i)$ is independent of the joint distribution $(h(s_j))_{j \neq i}$ and uniformly distributed in $[m]$. It follows that $f(s_i)$ is independent of the joint distribution $(f(s_j))_{j \neq i}$. Since it further holds that $\mathbb{E}[f(s_i)] = 0$, this implies

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

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

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

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

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

we always have

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

by the QA-inequality. We now wish to combine this bound with Lemma 4.3 to obtain a bound on the inner sum in (8). For this, we define let $\ell = |T_r|n$ and define sets of keys F_1, \dots, F_ℓ and weight functions $w_1, \dots, w_\ell : \Sigma^c \rightarrow \mathbb{R}$ as follows. Enumerate $T_r = \{i_1, \dots, i_{|T_r|}\}$ such that $\{i_1, \dots, i_{|B_r|}\} = B_r$. Now for $0 \leq k < |B_r|$ and $1 \leq j \leq n$ we define $F_{kn+j} = A_j$. We further define the weight function $w_{kn+j} : \Sigma^c \rightarrow \mathbb{R}$ by $w_{kn+j}(x) = \sqrt{\ell \sum_{b \in Q} v(x, b)^2}$ for $x \in \Sigma^c$. Observe that these weight functions are all identical. Secondly, for $|B_r| \leq k < |T_r|$ and $1 \leq j \leq n$, we define $F_{kn+j} = \{s_{i_k}(j)\}$, and $w_{kn+j} : \Sigma^c \rightarrow \mathbb{R}$ by $w_{kn+j}(x) = 1$ for all $x \in \Sigma^c$. Then,

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

where the last inequality follows from Lemma 4.3. Note that this upper bound does not depend on the choice of $(s_i)_{i \in \{1, \dots, t\} \setminus B_r} \in S^{t-|B_r|}$ in the outer sum in (8). Repeating this argument another $r - 1$ times, and using that $\{1, \dots, t\}$ is the disjoint union of B_1, \dots, B_r , we obtain that

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

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

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

□

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

THEOREM 4.5. *Let $h : \Sigma^c \rightarrow [m]$ be a simple tabulation hash function and $S \subseteq \Sigma^c$ a set of keys. Let $v : \Sigma^c \times [m] \rightarrow [-1, 1]$ be a value function such that the set $Q = \{i \in [m] \mid \exists x \in \Sigma^c : v(x, i) \neq 0\}$ satisfies $|Q| \leq m^\epsilon$, where $\epsilon < \frac{1}{4}$ is a constant. For $j \in [m]$ define the random variable $V_j = \sum_{x \in S} v(x, h(x) \oplus j)$ and let $\mu = \mathbb{E}[V_j]$, noting that this is independent of j .*

For any $\gamma \geq 1$,

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

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

PROOF. First, note that we may write

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

Now, define $v'(x) = \sum_{k \in Q} v(x, k)^2$ and for $X \subseteq \Sigma^c$ we let $v'(X) = \sum_{x \in X} v'(x)$ and define $v'_\infty(X) = \max_{x \in X} v'(x)$. Now applying Lemma 4.2 with respect to v' we get position characters $\alpha_1, \dots, \alpha_r$ with corresponding groups G_1, \dots, G_r , such that, $\cup_{i=1}^r G_i = S$ and for every $i \in \{1, \dots, r\}$, $v'(G_i) \leq v'(S)^{1-1/c} v'_\infty(S)^{1/c}$. For $i \in \{1, \dots, r\}, j \in [m]$ we define the random variables

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

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

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

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

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

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

and bound V_1 and V_2 separately starting with V_1 .

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

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

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

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

By a union bound,

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\Pr \left[V_2 \geq \frac{C_Y^c}{2} v'(S) \right] \leq 2 \Pr \left[\sum_{i=1}^r Z'_i > \frac{C_Y^c}{2} v'(S) \right] + \Pr [A_r^c] + \Pr [B_r^c]. \quad (13)$$

We now wish to apply Corollary 3.2 to the martingale difference $(Z'_i, \mathcal{F}_i)_{i=0}^r$. Thus, we have to bound $|Z'_i|$ as well as the conditional variances $\text{Var} [Z'_i \mid \mathcal{F}_{i-1}]$. For the bound on Z'_i , observe that by the Cauchy-Schwarz inequality,

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

If A_i occurs we obtain

$$\sum_{j \in [m]} (Y_i^{(j)})^2 \leq C_Y^{c-1} v'(G_i) \leq C_Y^{c-1} v'(S)^{1-1/c} v'_\infty(S)^{1/c},$$

by Lemma 4.2 and if A_i, B_i , and C_i all occur then

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

In conclusion

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

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

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

It follows that $\sum_{i=1}^r \text{Var} [Z'_i \mid \mathcal{F}_{i-1}] \leq m^{-1/2} v'(S)^2$. Letting

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

and

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

in Corollary 3.2 we thus obtain

$$\Pr \left[\sum_{i=1}^r Z'_i > \frac{C_Y^c}{2} v'(S) \right] \leq \exp \left(- \frac{v'(S)^{1/c}}{3C_Y^{2c-1} \cdot \sqrt{m} \cdot v'_\infty(S)^{1/c}} C \left(\frac{(C_Y)^{2c-1/2} \cdot \sqrt{3} \cdot \sqrt{m} \cdot v'_\infty(S)^{1/2c}}{2v'(S)^{1/2c}} \right) \right).$$

Applying Lemma 3.4 first with $b = \left(\frac{v'_\infty(S)}{v'(S)}\right)^{1/(2c)} \leq 1$ and then with $b = \sqrt{m} > 1$ yields

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

We then use Lemma 3.3 to get

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

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

$$\Pr \left[\sum_{i=1}^r Z'_i > \frac{C_Y^c}{2} v'(S) \right] \leq m^{-\gamma}. \quad (14)$$

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

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

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

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

Call this quantity T_i . It follows from Lemma 4.4 and Markov's inequality that

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

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

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

4.3 Establishing the Concentration Bound

With the results of the previous subsection at hand, we proceed to prove the first part of Theorem 2.1. We show that for a value function of support bounded in size by m^ϵ for some $\epsilon < 1/4$, simple tabulation supports Chernoff-style bounds with added error probability inversely polynomial in m . For convenience, we restate the first part of Theorem 2.1

as Theorem 4.6. The statement is equivalent to Theorem 2.1 but for precision, we have chosen to write out the statement more explicitly.

THEOREM 4.6. *Let $h: \Sigma^c \rightarrow [m]$ be a simple tabulation hash function and $S \subseteq \Sigma^c$ be a set of keys. Let $v: \Sigma^c \times [m] \rightarrow [-1, 1]$ be a value such that the set $Q = \{i \in [m] \mid \exists x \in \Sigma^c : v(x, i) \neq 0\}$ satisfies $|Q| \leq m^\epsilon$, where $\epsilon < \frac{1}{4}$ is a constant. Define the random variable $W = \sum_{x \in S} v(x, h(x))$ and write $\mu = \mathbb{E}[W]$ and $\sigma^2 = \text{Var}[W]$. Then for any constant $\gamma \geq 1$,*

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

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

PROOF. First, akin to the proof of Theorem 4.5, we may write

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

and note that

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

We proceed by induction on c . For $c = 1$ we have full randomness and it follows immediately from Corollary 3.2 that

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

Now assume that $c > 1$ and inductively that the result holds for smaller values of c . We define $v'(x) = \sum_{k \in Q} v(x, k)^2$ and for $X \subseteq \Sigma^c$ we let $v'(X) = \sum_{x \in X} v'(x)$ and define $v'_\infty(X) = \max_{x \in X} v'(x)$. Now, applying Lemma 4.2 with respect to $w = v'$ we get position characters $\alpha_1, \dots, \alpha_r$ with corresponding groups G_1, \dots, G_r , such that $\cup_{i=1}^r G_i = S$ and for every $i \in \{1, \dots, r\}$, $v'(G_i) \leq v'(S)^{1-1/c} v'_\infty(S)^{1/c}$. For a bin $j \in [m]$ and an $i \in \{1, \dots, r\}$ we again define

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

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

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

According to Theorem 4.5 there exists a constant $K = 3 \cdot 2^6 \cdot \gamma^2$ such that

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

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

$$A_i = \bigcap_{k \leq i} \left(\sum_{j \in [m]} (X_k^{(j)})^2 \leq K^{c-1} v'(G_k) \right),$$

$$B_i = \left(\max_{\substack{k \leq i \\ j \in [m]}} |X_k^{(j)}| \leq C_{\gamma+1, c-1} M \right),$$

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

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

Setting $\sigma_0^2 = \frac{K^{c-1} v'(S)}{m}$ and applying Corollary 3.2 we obtain

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

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

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

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

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

so using (17) we obtain

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

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

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

where $\sigma_i^2 = \text{Var} \left[Y_i^{(j)} \right] \leq v'(G_i)/m$. By the initial assumption on the groups, this implies $\sigma_i^2 \leq v'(S)^{1-1/c} v'_\infty(S)/m$ and we denote the latter quantity τ^2 . Combining with Lemma 3.5 we obtain by a union bound that

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

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

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

First since $\frac{C_{\gamma, c} C_{\gamma+1, c-1}}{3K^{c-1}} \geq 1$ Lemma 3.4 give us that

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

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

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

So we have reduced the problem to showing that

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

For that we have to check a couple of cases.

Case 1. $\frac{tM}{\sigma^2} \leq 1$: Using Lemma 3.3 twice and the fact that $(C\gamma)^3 \geq \frac{3}{2}$, we get that

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

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

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

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

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

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

We then get that

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

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

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

where we have used Lemma 3.3 and that $\gamma \geq 1$. Now we have that $\sigma^2 \geq \frac{v'(S)}{3m} > m^{\frac{1-2\varepsilon}{2c-1}}/3$ and $\tau^2 \leq \frac{v'(S)^{1-1/c} v'_\infty(S)^{1/c}}{m}$. Combining this we get that

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

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

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

□

5 GENERAL VALUE FUNCTIONS – ARBITRARY BINS

The goal of this section is to prove Theorem 2.2, the second step towards Theorem 1.2. Again, we postpone the argument that our concentration bounds are query invariant to Section 7. Recall that Theorem 2.2 is concerned with a hash function of the form $h = \tau \circ g$, where $g : \Sigma^c \rightarrow [m]$ is a simple tabulation hash function and τ is a uniformly random permutation. Our goal is to prove that for any value function $v : \Sigma^c \times [m] \rightarrow [-1, 1]$, the sum $\sum_{x \in \Sigma^c} v(x, h(x))$ is strongly concentrated with high probability in m . This result follows by combining the distributional properties of g with the randomness of τ .

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

LEMMA 5.1. *Let $m \geq 2$ be an integer and $C, T \in \mathbb{R}^+$ positive reals. Furthermore, let $\mathcal{V} : [m] \times [m] \rightarrow \mathbb{R}$ be a value function satisfying $\sum_{i \in [m]} \mathcal{V}(i, j) = 0$ for every $j \in [m]$ and such that*

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

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

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

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

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

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

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

Thus, we shall prove (20) for Y_1 – the proof is completely analogous for Y_2 . Define the filtration $(\mathcal{F}_i)_{i=0}^{\lceil m/2 \rceil}$ by $\mathcal{F}_i = \sigma \left((\tau(j))_{j \in [i]} \right)$ and let $X_i = \mathbb{E}[Y_1 | \mathcal{F}_i]$ such that $(X_i, \mathcal{F}_i)_{i=0}^{\lceil m/2 \rceil}$ is a martingale, $X_0 = \mathbb{E}[Y_1]$, and $X_{\lceil m/2 \rceil} = Y_1$. Towards applying Corollary 3.2, we bound $|X_i - X_{i-1}|$ and $\sum_{i=1}^{\lceil m/2 \rceil} \text{Var}[X_i - X_{i-1} | \mathcal{F}_{i-1}]$.

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

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

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

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

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

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

Hence, it follows that

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

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

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

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

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

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

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

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

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

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

The assumption on \mathcal{V} implies that $\mathbb{E}[\mathcal{V}(\tau(i), i)] = 0$ for each $i \in [m]$, so also $\mathbb{E}[Y_1] = 0$. Applying Corollary 3.2 then yields,

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

The goal is now to show that

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

Because if this is the case, then as desired

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

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

Case 1. $M \leq \frac{10}{D}$: In this case, $\frac{DM}{10} \leq 1$. Thus, by Lemma 3.4,

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

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

Case 2. $\frac{10}{D} \leq M \leq C$: In this case, $\frac{DM}{10} \geq 1$. Thus, by Lemma 3.4,

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

using that $D \geq 8C$.

Case 3. $M \leq \frac{\sigma^2}{T}$: In this case, recall that $D \geq 12$ such that $\frac{D}{10} \geq 1$ and we may apply Lemma 3.4, yielding

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

By Lemma 3.3,

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

So finally,

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

where we have used Lemma 3.3 and the fact that $D \geq 12$. □

With this result in hand we are ready to prove Theorem 2.2. We restate it here in a more technically explicit version. For a more intuitive understanding, please refer back to the original statement. Note that we only require the hash function h of the theorem to be 2-independent, whereas Theorem 2.2 requires the hash function to be 3-independent. The difference lies in that the statement of Theorem 2.2 is slightly stronger, guaranteeing query invariance. Having deferred the treatment of query invariance until later, we only need 2-independence for now.

THEOREM 5.2. *Let $\varepsilon \in (0, 1]$ and $m \geq 2$ be given. Let $h: A \rightarrow [m]$ be a 2-independent hash function satisfying the following. For every $\gamma > 0$ and every value function $\tilde{v}: A \times [m] \rightarrow [-1, 1]$ such that $Q = \{i \in [m] \mid \exists x \in A: \tilde{v}(x, i) \neq 0\}$ has size $|Q| \leq m^\varepsilon$, the random variables $W = \sum_{x \in A} \tilde{v}(x, h(x))$ and $W_j = \sum_{x \in A} \tilde{v}(x, h(x) \oplus j)$, $j \in [m]$ with mean $\mu_W = \mathbb{E}[W] = \mathbb{E}[W_j]$ and variance $\sigma_W^2 = \text{Var}[W]$ satisfy the inequalities*

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

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

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

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

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

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

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

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

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

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

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

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

As $\sigma' \leq \sigma$ and $V = 2V'$ the theorem then follows with $E = 2E'$ by applying Lemma 3.5.

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

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

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

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

and note that $V_1 + V_2 = V'$. Defining value functions $v'_1, v'_2 : A \times [m] \rightarrow [-1, 1]$ by

$$v'_1(x, i) = \begin{cases} v'(x, i), & \text{if } i \in [m^\varepsilon] \\ 0, & \text{otherwise} \end{cases} \quad \text{and} \quad v'_2(x, i) = \begin{cases} v'(x, i), & \text{if } i \in [m] \setminus [m^\varepsilon] \\ 0, & \text{otherwise} \end{cases},$$

we observe that

$$V_1 = \sum_{x \in A} v'_1(x, \tau(h(x))) - \mathbb{E} \left[\sum_{x \in A} v'_1(x, \tau(h(x))) \right] \quad \text{and} \quad V_2 = \sum_{x \in A} v'_2(x, \tau(h(x))) - \mathbb{E} \left[\sum_{x \in A} v'_2(x, \tau(h(x))) \right]$$

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

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

By a union bound,

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

and we proceed to bound each of these terms individually.

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

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

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

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

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

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

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

We now bound $\Pr[|\mathcal{A} \cdot \mathcal{B} \cdot V_2| \geq E't/2]$. It suffices to condition on $h = h_0$ for some h_0 satisfying that $[\mathcal{A}] = [\mathcal{B}] = 1$ and make the bound over the randomness of τ . For this we may use Lemma 5.1. Indeed if $h = h_0$ for some h_0 such that $[\mathcal{A}] = [\mathcal{B}] = 1$, then $\sum_{i \in [m]} \sum_{j \in [m]} \frac{1}{m} \mathcal{V}(i, j)^2 \leq D\sigma'^2$. Here we used the conditioning on \mathcal{A} . Define the function $\mathcal{V}_0 : [m] \times [m] \rightarrow \mathbb{R}$ by $\mathcal{V}_0(i, j) = \mathcal{V}(i, j)$ when $j \in [m] \setminus [m^\epsilon]$ and $\mathcal{V}_0(i, j) = 0$ otherwise. Then also $\sum_{i \in [m]} \sum_{j \in [m]} \frac{1}{m} \mathcal{V}_0(i, j)^2 \leq D\sigma'^2$ and further, for each $j \in [m]$, $\sum_{i \in [m]} \mathcal{V}_0(i, j) = 0$. Finally, the conditioning on \mathcal{B} gives that $\max_{i, j \in [m]} \mathcal{V}_0(i, j) \leq M$. Note that $V_2 = \sum_{j \in [m]} \mathcal{V}_0(\tau^{-1}(j), j)$. Applying Lemma 5.1 to \mathcal{V}_0 , noting that the bound obtained in that lemma is increasing in σ , we obtain that

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

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

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

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

\tilde{v} has support $A \times \{i\}$,

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

and $\text{Var}[\mathcal{V}(i, j)] \leq \frac{1}{m} \sum_{x \in A} v_2'(x, j)^2 = \sigma_j^2 \leq \sigma'^2/m^\epsilon$. The last inequality follows from our assumption that $\sigma_0^2 \geq \dots \geq \sigma_{m-1}^2$ and $j \geq m^\epsilon$.

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

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

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

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

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

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

Suppose first that $\sigma'^2 \leq m^{\epsilon/2}$. In that case we use Lemma 3.3 to conclude that

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

so if $C \geq 4 \frac{\gamma+2}{D'\epsilon}$ this is at least $\frac{(\gamma+2) \log m}{D'}$. Now suppose $m^{\epsilon/2} < \sigma'^2 \leq m^{2\epsilon}/(2D\gamma \log m)$. In that case we recall that

$M = \max\left\{C, \frac{\sigma'}{\sqrt{2D\gamma \log m}}\right\}$ and use the bound

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

If m is larger than some constant, this is certainly at least $\frac{(\gamma+2) \log m}{D'}$. Finally suppose that $\sigma'^2 > m^{2\epsilon}/(2D\gamma \log m)$.

Using the inequality $\log(1+x) \geq \frac{x}{2}$ holding for $0 \leq x \leq 1$ we find that

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

Again it holds that if m is greater than some constant, this is at least $\frac{(\gamma+2) \log m}{D'}$. It follows that if C is large enough, then

$\Pr[|\mathcal{V}(i, j)| \geq M] = O(|A|m^{-\gamma-2})$. Union bounding over $(i, j) \in [m] \times ([m] \setminus [m^\epsilon])$ we find that $\Pr[\mathcal{A}^c] = O(|A|m^{-\gamma})$.

Combining the bounds we find that

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

which completes the proof. \square

6 EXTENDING THE HASH RANGE

This section is dedicated to proving Theorem 2.3, which we will restate shortly. Again, we will postpone the argument that our concentration bounds are query invariant to Section 7. First, we prove the following technical lemma.

LEMMA 6.1. Let $\sigma^2 > 0$ and $t > 0$. Writing $s = \max\{\sigma^2, \sqrt{t\sigma^2}\}$,

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

PROOF. For $t \leq \sigma^2$ the inequality is trivial, so suppose $t > \sigma^2$. We note that for $x \geq 0$, $1 + \sqrt{x} \geq \sqrt{1+x}$, such that $\lg(1 + \sqrt{x}) \geq \lg(1+x)/2$ for every $x \geq 0$. Using this fact in between two applications of Lemma 3.3, we find that

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

□

Next, we recall the law of total variance.

LEMMA 6.2 (LAW OF TOTAL VARIANCE). For every pair of random variables X, Y ,

$$\text{Var}[Y] = \mathbb{E}[\text{Var}[Y | X]] + \text{Var}[\mathbb{E}[Y | X]].$$

In particular, $\text{Var}[Y] \geq \text{Var}[\mathbb{E}[Y | X]]$.

We are now ready to prove the main theorem of the section, which informally states that concatenating the output values of hash functions preserves the property of having Chernoff-style bounds. Note that the following is a much more explicit and elaborate statement of Theorem 2.3. The purpose of this restatement is to make a formal proof more readable. The reader is encouraged to refer back to Theorem 2.3 for intuition regarding the theorem statement. Again, we highlight that we have left out the part of Theorem 2.3 concerning query independence. How query independence is obtained will be discussed in Section 7

THEOREM 6.3. Let A be a finite set. Let $(X_a)_{a \in A}$ and $(Y_a)_{a \in A}$ be pairwise independent families of random variables taking values in B_X and B_Y , respectively, and satisfying that the distributions of $(X_a)_{a \in A}$ and $(Y_a)_{a \in A}$ are independent. Suppose that there exist universal constants $D_X, D_Y \geq 1$, $K_X, K_Y > 0$, and $R_X, R_Y \geq 0$ such that for every choice of value functions $v_X: A \times B_X \rightarrow [0, 1]$ and $v_Y: A \times B_Y \rightarrow [0, 1]$ and for every $t > 0$,

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

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

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

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

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

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

define

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

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

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

Now, the triangle inequality and a union bound yields

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

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

For bounding $\Pr [|T_1| > t/3]$, define the value function $v_X^{(1)} : A \times B_X \rightarrow [0, 1]$ by $v_X^{(1)}(a, x) = \mathbb{E} [V_a | X_a = x]$. Note that $\mathbb{E} [\mathbb{E} [V_a | X_a]] = \mu_a$ and $\text{Var} [\mathbb{E} [V_a | X_a]] \leq \sigma_a^2$, by the law of total variance, such that $\text{Var} \left[\sum_{a \in A} v_X^{(1)}(a, X_a) \right] \leq \sigma_{XY}^2$. Thus, by Equation (25) and Lemma 3.4,

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

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

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

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

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

Applying Equation (25) with the value function $v_X^{(2)} : A \times B_X \rightarrow [0, 1]$ given by $v_X^{(2)}(a, X_a) = L_a$ to $\Pr[|Z - \mathbb{E}[Z]| > t/6]$ yields

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

where the second follows from Lemma 6.1 and the third inequality follows from Lemma 3.4.

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

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

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

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

where the second follows from Lemma 3.4 and the third inequality follows from Lemma 6.1. In conclusion,

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

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

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

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

$$\mathbb{E}[\text{Var}[T_3 \mid (X_a)_{a \in A}]] \leq \text{Var}[T_3] \leq \sigma_{XY}^2,$$

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

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

Applying Equation (25) with $v_X^{(3)}$, Lemma 6.1, and Lemma 3.4, we obtain

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

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

7 QUERY INVARIANCE

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

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

Tabulation-Permutation. In the proof of Theorem 2.2 we consider some specific value function w . We proceed by considering separately the m^ϵ bins $S \subseteq [m]$ of largest contribution to the variance, σ^2 , and then the remaining bins, $[m] \setminus S$. The contribution of each subset of bins is then individually bounded. In the first case, we simply use the assumption on the hash function h that we received in a black box manner and use no properties of the permutation. Now, say towards query invariance that we require that $\tau \circ h(q) = i$. To support this, we instead chose S to have size $|S| = m^\epsilon/2$. This does not change the proof by more than constant factors and simply adding i to S yields a set $S' = S \cup \{i\}$ of size $|S'| < m^\epsilon$, such that the assumption on h directly yields the result. In conclusion, the proof goes through exactly as before.

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

8 TIGHTNESS OF CONCENTRATION: SIMPLE TABULATION INTO FEW BINS

Recall the result of Theorem 1.1. If $h: [u] \rightarrow [m]$ is a simple tabulation hash function with $[u] = \Sigma^c$ and $c = O(1)$, and $S \subseteq [u]$ is a set of hash keys of size $n = |S|$ where each key $x \in S$ is given a weight $w_x \in [0, 1]$. Then for arbitrary $y \in [m]$ and a constant $\gamma > 0$ the total weight of the balls landing in bin y , given by the random variable $X = \sum_{x \in S} w_x [h(x) = y]$, satisfies the concentration bound

$$\Pr[|X - \mu| \geq t] \leq 2 \exp(-\Omega(\sigma^2 C(t/\sigma^2))) + n/m^\gamma, \quad (27)$$

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

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

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

THEOREM 8.1. *Let $m \leq |\Sigma|^{1-\epsilon}$ for some constant $\epsilon > 0$ and $h: [u] \rightarrow [m]$ be a simple tabulation hash function. Let $0 < \epsilon' < \epsilon$ be a constant and suppose that $C: \mathbb{R}^+ \rightarrow \mathbb{R}^+$ is a function such that for all $0 \leq \gamma \leq |\Sigma|^{\epsilon'/c}$, all sets $S \subseteq [u]$, all*

choices of weights $w_x \in [0, 1]$, $x \in S$, and every $y \in [m]$, the random variable $X = \sum_{x \in S} w_x [h(x) = y]$ satisfies

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

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

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

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

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

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

where we have applied Lemma 3.3.

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

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

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

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

where the last inequality used Lemma 3.3 and $\sqrt{\log(m)} k^{c-1} < \sigma$. Let $k = \frac{\gamma}{2c}$ and combine the above inequalities to conclude that $2m^{-\frac{(\gamma/(2c))^{c-1}}{C(\gamma)}} + m^{-\gamma} = \Omega(m^{-\gamma/2})$. It follows that indeed, $C(\gamma) = \Omega(\gamma^{c-2})$. \square

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

ACKNOWLEDGEMENT

Anders Aamand, Jakob B. T. Knudsen, Peter M. R. Rasmussen, and Mikkel Thorup are partly supported by Thorup's Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.

REFERENCES

- [1] Anders Aamand, Debarati Das, Evangelos Kipouridis, Jakob Bæk Tejs Knudsen, Peter M. R. Rasmussen, and Mikkel Thorup. 2020. No Repetition: Fast Streaming with Highly Concentrated Hashing. *CoRR* (2020). arXiv:2004.01156 arxiv.org/abs/2004.01156.
- [2] Arne Andersson, Peter Bro Miltersen, Søren Riis, and Mikkel Thorup. 1996. Static Dictionaries on AC^0 RAMs: Query Time $\Theta(\sqrt{\log n / \log \log n})$ is Necessary and Sufficient. In *37th Annual Symposium on Foundations of Computer Science (FOCS)*. 441–450. <https://doi.org/10.1109/SFCS.1996.548503>
- [3] Austin Appleby. 2016. *MurmurHash3*.
- [4] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. 2013. BLAKE2: Simpler, Smaller, Fast as MD5. In *Applied Cryptography and Network Security*, Michael Jacobson, Michael Locasto, Payman Mohassel, and Reihaneh Safavi-Naini (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 119–135.
- [5] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. 2002. Counting Distinct Elements in a Data Stream. In *International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*. 1–10.
- [6] George Bennett. 1962. Probability Inequalities for the Sum of Independent Random Variables. *J. Amer. Statist. Assoc.* 57, 297 (1962), 33–45. <https://doi.org/10.1080/01621459.1962.10482149> arXiv:<https://www.tandfonline.com/doi/pdf/10.1080/01621459.1962.10482149>
- [7] Sergei Natanovich Bernstein. 1924. On a modification of Chebyshev's inequality and of the error formula of Laplace. *Ann. Sci. Inst. Sav. Ukraine, Sect. Math.* 1 (1924), 38–49.
- [8] Andrei Z. Broder. 1997. On the resemblance and containment of documents. In *Compression and Complexity of Sequences (SEQUENCES)*. 21–29.
- [9] Larry Carter and Mark N. Wegman. 1979. Universal classes of hash functions. *J. Comput. System Sci.* 18, 2 (1979), 143–154. Announced at STOC'77.
- [10] L. Elisa Celis, Omer Reingold, Gil Segev, and Udi Wieder. 2011. Balls and Bins: Smaller Hash Families and Faster Evaluation. In *52nd Annual Symposium on Foundations of Computer Science (FOCS)*. 599–608.
- [11] Ashok K. Chandra, Larry J. Stockmeyer, and Uzi Vishkin. 1984. Constant Depth Reducibility. *SIAM J. Comput.* 13, 2 (1984), 423–439. <https://doi.org/10.1137/0213028>
- [12] Herman Chernoff. 1952. A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the sum of Observations. *Annals of Mathematical Statistics* 23, 4 (1952), 493–507.
- [13] Tobias Christiani and Rasmus Pagh. 2014. Generating k -Independent Variables in Constant Time. In *55th Annual Symposium on Foundations of Computer Science (FOCS)*. 196–205.
- [14] Tobias Christiani, Rasmus Pagh, and Mikkel Thorup. 2015. From independence to expansion and back again. In *Proceedings of the 47rd ACM Symposium on Theory of Computing (STOC)*.
- [15] Kai-Min Chung, Michael Mitzenmacher, and Salil Vadhan. 2013. Why simple hash functions work: Exploiting the entropy in a data stream. *Theory of Computing* 9, 1 (2013), 897–945.
- [16] Søren Dahlgaard, Mathias Bæk Tejs Knudsen, Eva Rotenberg, and Mikkel Thorup. 2015. Hashing for Statistics over K -Partitions. In *56th Annual Symposium on Foundations of Computer Science (FOCS)*. 1292–1310. <https://doi.org/10.1109/FOCS.2015.83>
- [17] Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Mikkel Thorup. 2017. Practical Hash Functions for Similarity Estimation and Dimensionality Reduction. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS)*. Curran Associates Inc., 6618–6628. <http://dl.acm.org/citation.cfm?id=3295222.3295407>
- [18] Martin Dietzfelbinger. 1996. Universal hashing and k -wise independent variables via integer arithmetic without primes. In *Proceedings of the 13th Symposium on Theoretical Aspects of Computer Science (STACS)*. 569–580.
- [19] Martin Dietzfelbinger and Friedhelm Meyer auf der Heide. 1992. Dynamic Hashing in Real Time. In *Informatik, Festschrift zum 60. Geburtstag von Günter Hotz*. 95–119. https://doi.org/10.1007/978-3-322-95233-2_7
- [20] Martin Dietzfelbinger and Michael Rink. 2009. Applications of a Splitting Trick. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP)*. 354–365.
- [21] Martin Dietzfelbinger and Christoph Weidling. 2007. Balanced allocation and dictionaries with tightly packed constant size bins. *Theor. Comput. Sci.* 380 (06 2007), 47–68. <https://doi.org/10.1016/j.tcs.2007.02.054>
- [22] Martin Dietzfelbinger and Philipp Woelfel. 2003. Almost Random Graphs with Simple Hash Functions. In *Proceedings of the 25th ACM Symposium on Theory of Computing (STOC)*. 629–638.
- [23] A. I. Dumey. 1956. Indexing for rapid random access memory systems. *Computers and Automation* 5, 12 (1956), 6–9.
- [24] Xiequan Fan, Ion Grama, and Quansheng Liu. 2012. Hoeffding's inequality for supermartingales. *Stochastic Processes and their Applications* 122, 10 (2012), 3545 – 3559. <https://doi.org/10.1016/j.spa.2012.06.009>
- [25] Dimitris Fotakis, Rasmus Pagh, Peter Sanders, and Paul Spirakis. 2005. Space Efficient Hash Tables with Worst Case Constant Access Time. *Theory of Computing Systems* 38, 2 (01 Feb 2005), 229–248. <https://doi.org/10.1007/s00224-004-1195-x>

- [26] Parikshit Gopalan, Daniel M. Kane, and Raghu Meka. 2018. Pseudorandomness via the Discrete Fourier Transform. *SIAM J. Comput.* 47, 6 (2018), 2451–2487. <https://doi.org/10.1137/16M1062132>
- [27] Torben Hagerup and Torsten Tholey. 2001. Efficient Minimal Perfect Hashing in Nearly Minimal Space. In *Proceedings of the 18th Symposium on Theoretical Aspects of Computer Science (STACS)*. 317–326.
- [28] John L. Hennessy and David A. Patterson. 2012. *Computer Architecture - A Quantitative Approach, 5th Edition*. Morgan Kaufmann.
- [29] Donald E. Knuth. 1963. Notes on open addressing. (1963). Unpublished memorandum. See <http://citeseer.ist.psu.edu/knuth63notes.html>.
- [30] Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang, and Yan Chen. 2003. Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the 3rd Internet Measurement Conference (IMC)*. 234–247. <https://doi.org/10.1145/948205.948236>
- [31] Daniel Lemire and Owen Kaser. 2016. Faster 64-bit universal hashing using carry-less multiplications. *Journal of Cryptographic Engineering* 6, 3 (01 Sep 2016), 171–185. <https://doi.org/10.1007/s13389-015-0110-5>
- [32] Yishay Mansour, Noam Nisan, and Prasoos Tiwari. 1993. The Computational Complexity of Universal Hashing. *Theor. Comput. Sci.* 107, 1 (1993), 121–133. [https://doi.org/10.1016/0304-3975\(93\)90257-T](https://doi.org/10.1016/0304-3975(93)90257-T)
- [33] Raghu Meka, Omer Reingold, Guy N. Rothblum, and Ron D. Rothblum. 2014. Fast Pseudorandomness for Independence and Load Balancing - (Extended Abstract). In *Proceedings of the 41st International Colloquium on Automata, Languages and Programming (ICALP)*. 859–870.
- [34] Peter Bro Miltersen. 1996. Lower Bounds for Static Dictionaries on RAMs with Bit Operations But No Multiplication. In *Proceedings of the 23rd International Colloquium on Automata, Languages and Programming (ICALP)*. 442–453. https://doi.org/10.1007/3-540-61440-0_149
- [35] Michael Mitzenmacher and Salil P. Vadhan. 2008. Why simple hash functions work: exploiting the entropy in a data stream. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 746–755.
- [36] Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized Algorithms*. Cambridge University Press.
- [37] Anna Pagh and Rasmus Pagh. 2008. Uniform Hashing in Constant Time and Optimal Space. *SIAM J. Comput.* 38, 1 (2008), 85–96.
- [38] Mihai Patrăşcu and Mikkel Thorup. 2012. The Power of Simple Tabulation-Based Hashing. *J. ACM* 59, 3 (2012), Article 14. Announced at STOC’11.
- [39] Mihai Patrăşcu and Mikkel Thorup. 2016. On the k -Independence Required by Linear Probing and Minwise Independence. *ACM Trans. Algorithms* 12, 1 (2016), 8:1–8:27.
- [40] Geoff Pike and Jyrki Alakuijala. 2011. Introducing cityhash. <https://opensource.googleblog.com/2011/04/introducing-cityhash.html>.
- [41] Mihai Patrăşcu and Mikkel Thorup. 2013. Twisted Tabulation Hashing. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 209–228.
- [42] René Schilling. 2005. *Measures, Integrals and Martingales*. Cambridge University Press. <https://doi.org/10.1017/CBO9780511810886>
- [43] Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. 1995. Chernoff-Hoeffding bounds for applications with limited independence. *SIAM Journal on Discrete Mathematics* 8, 2 (1995), 223–250. Announced at SODA’93.
- [44] Alan Siegel. 2004. On Universal Classes of Extremely Random Constant-Time Hash Functions. *SIAM J. Comput.* 33, 3 (2004), 505–543. Announced at FOCS’89.
- [45] Mikkel Thorup. 2013. Simple Tabulation, Fast Expanders, Double Tabulation, and High Independence. In *54th Annual Symposium on Foundations of Computer Science (FOCS)*. 90–99.
- [46] Mikkel Thorup. 2015. High Speed Hashing for Integers and Strings. *CoRR* (2015). arxiv.org/abs/1504.06804.
- [47] Mikkel Thorup and Yin Zhang. 2012. Tabulation-Based 5-Independent Hashing with Applications to Linear Probing and Second Moment Estimation. *SIAM J. Comput.* 41, 2 (2012), 293–331. Announced at SODA’04 and ALENEX’10.
- [48] Mark N. Wegman and Larry Carter. 1981. New Classes and Applications of Hash Functions. *J. Comput. System Sci.* 22, 3 (1981), 265–279. Announced at FOCS’79.
- [49] Albert Lindsey Zobrist. 1970. *A New Hashing Method with Application for Game Playing*. Technical Report 88. Computer Sciences Department, University of Wisconsin, Madison, Wisconsin.

A EXPERIMENTS

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

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

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

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

hash values. Aside from having strong theoretical guarantees, our experiments show that tabulation-permutation and tabulation-1permutation are very fast in practice.

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

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

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

First, we consider simple bad instances for Multiply-Shift and 2-independent PolyHash. These are analyzed in detail in [39, Appendix B]. The specific instance we consider is that of hashing the arithmetic progression $A = \{a \cdot i \mid i \in [50000]\}$ into 16 bins, where we are interested in the number of keys from A that hashes to a specific bin. We performed this experiment 5000 times, with independently chosen hash functions. The cumulative distribution functions on the number of keys from A hashing to a specific bin is presented in Figure 2. We see that most of the time 2-independent PolyHash and Multiply-Shift distribute the keys perfectly with exactly $1/16$ of the keys in our bin. Since

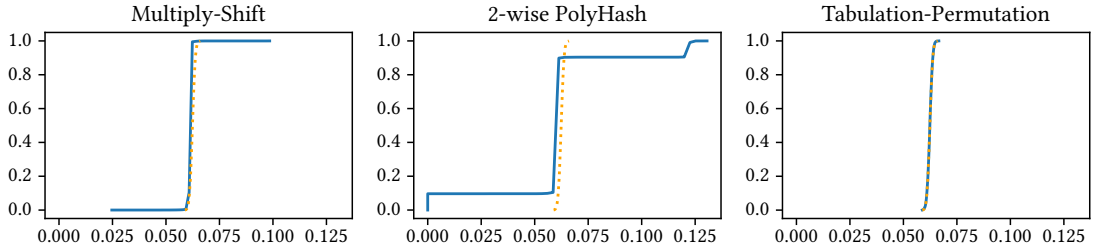


Fig. 2. Hashing the arithmetic progression $\{a \cdot i \mid i \in [50000]\}$ to 16 bins for a random integer a . The dotted line is a 100-independent PolyHash.

the variance is the same as with fully random hashing, this should suggest a much heavier tail, which is indeed what our experiments show. For contrast, we see that the cumulative distribution function with our tabulation-permutation hash function is almost indistinguishable from that of 100-independent Poly-Hash. We note that our experiments with tabulation-permutation is only a sanity check: No experiment can prove good performance on all possible inputs.

Our second set of experiments shows bad instances for simple tabulation and twisted tabulation. We already know theoretically from Section 8 that these bad instances exist, but we shall now see that, in a sense, things can be even worse than described in Section 8 for certain sets of keys. The specific instance we consider is hashing the discrete cube $Q = [2]^7 \times [2]^6$ to $m = 2$ bins using simple tabulation, twisted tabulation, and tabulation-permutation. We performed this experiment 5000 times, with independently chosen hash functions, and again we were interested in the number of keys from Q hashing to one of the bins. The cumulative distribution functions of the number of such keys is presented in Figure 3. Let us explain the appearance of the curves for simple and twisted tabulation. In general, if we hash the set of keys $[2] \times R$ to $[2]$ with simple tabulation, then if $h_1(0) \neq h_1(1)$, each bin will get exactly the same amount of keys. When we hash the set of keys $[2]^7 \times [2]^6$ this happens with probability $1 - 2^{-7}$. If on the other hand $h_i(0) = h_i(1)$ for each $i = 1, \dots, 7$, which happens with probability 2^{-7} , the distribution of the balls in the bins is the same as that when 2^6 balls, each of weight 2^7 , are distributed independently and uniformly at random into the two bins. If this happens, the variance of the number of balls in a bin is a factor of 2^7 higher, so we expect a much heavier tail than in the completely independent case. These observations agree with the results in Figure 3. Most of the time, the distribution is perfect, but the tail is very heavy. We believe that this instance is also one of the worst instances for tabulation-permutation hashing. We would therefore expect to see that on this instance it performs slightly worse than 100-independent PolyHash, which is indeed what our experiments show. We note that that no amount of experimentation can prove that tabulation-permutation always works well for all inputs. We do, however, have mathematical concentration guarantees, and the experiments performed here give us some idea of the impact of the constant delay hidden in the exponential decrease in the bounds of Theorem 1.2. For completeness, we note that the situation with mixed tabulation is unresolved. Neither do we have strong concentration bounds, nor any bad instances showing that such bounds do not hold. Running experiments is not expected to resolve this issue since mixed tabulation, as any other 2-independent hashing scheme, performs well on almost all inputs [35].

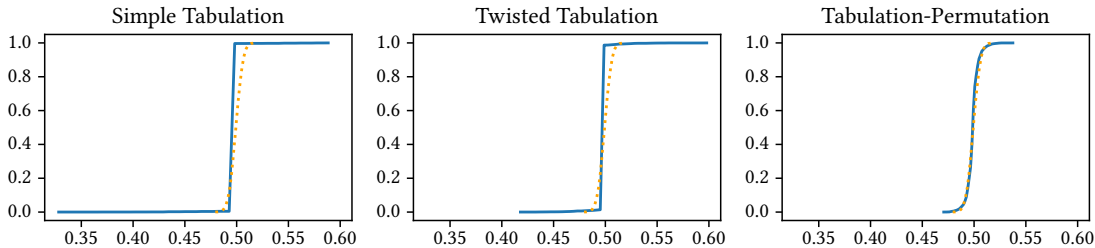


Fig. 3. Hashing the discrete cube $[2]^7 \times [2]^6$ to 2 bins. The dotted line is a 100-independent PolyHash.

Appendix B

No Repetition: Fast Streaming with Highly Concentrated Hashing

No Repetition: Fast Streaming with Highly Concentrated Hashing

Anders Aamand
BARC, University of Copenhagen
Copenhagen, Denmark
aa@di.ku.dk

Debarati Das
BARC, University of Copenhagen
Copenhagen, Denmark
debaratix710@gmail.com

Evangelos Kipouridis
BARC, University of Copenhagen
Copenhagen, Denmark
kipouridis@di.ku.dk

Jakob B. T. Knudsen
BARC, University of Copenhagen
Copenhagen, Denmark
jkn@di.ku.dk

Peter M. R. Rasmussen
BARC, University of Copenhagen
Copenhagen, Denmark
pmrr@di.ku.dk

Mikkel Thorup
BARC, University of Copenhagen
Copenhagen, Denmark
mikkel2thorup@gmail.com

ABSTRACT

Working with huge amounts of data, for instance in the setting of streaming algorithms, it is common to replace precise statistics with stochastic estimators. Such estimators typically rely on hashing-based algorithms. In that vein, Aamand et al. (STOC'20) recently proposed a fast and practical hashing scheme with strong concentration bounds, Tabulation-1Permutation, the first of its kind. In this paper, we demonstrate the benefits of applying such a hash function to obtain reliable estimators.

In previous work, a common strategy to get estimators that work within a certain error bound with high probability is to design one that works with constant probability, and then boost the probability using r independent repetitions. Important examples of applications are small space algorithms for estimating the number of distinct elements in a stream, or estimating the similarity between large sets. We consider these problems and prove that using a hash function with strong concentration bounds, akin to those of Tabulation-1Permutation, we may achieve the same high probability bounds without any need for repetitions. Using the same amount of space, we thus save a factor r in time, and simplify the overall algorithm.

We validate our approach experimentally on both real and synthetic data, focusing on the application of counting distinct elements. We compare Tabulation-1Permutation with other hash functions such as strongly universal hash functions and various other hash functions such as MurmurHash3 and BLAKE3, both with and without resorting to repetitions. We see that if we want reliability in terms of small error probabilities, then Tabulation-1Permutation is significantly faster.

1 INTRODUCTION

Recent years have brought with them a huge demand for algorithms that can process and compute statistics on large streams of data. Common statistics of interest include the number of distinct elements of a stream and the set similarity between two large sets. The sheer volume of the data makes storing a complete copy of the stream and performing exact computations an impossible task, and so, if the data is not processed in time, the information is lost. To solve this problem, we therefore have to resort to estimation algorithms. For instance, instead of precisely counting the number of unique visitors to a website, we may settle for a good estimate. In simple terms, the goal of these estimation algorithms are as follows:

For a data stream S and some statistic $\mathcal{F} = \mathcal{F}(S)$, we want an estimator $\hat{\mathcal{F}}$ such that $\hat{\mathcal{F}} \in (1 \pm \epsilon)\mathcal{F}$ with some high probability at least $1 - \delta$.

To get such an estimator, a common strategy is to design one that works with constant probability, and then boost the probability using independent repetitions. A classic example of this approach is the algorithm of Bar-Yossef et al. [3] to estimate the number of distinct elements in a stream. Using strongly universal hashing to process each element, we obtain an estimator such that the probability of getting too large an error is at most a constant, e.g., $1/4$. By performing r independent repetitions and taking the median of the estimators, the error probability falls exponentially in r . However, running r independent experiments increases the processing time by a factor r .

Here we make the point that if we have a hash function with strong concentration bounds (to be defined in Section 2.2), then we get the same high probability bounds without any need for repetitions. Instead of r independent sketches, we have a single sketch that is $\Theta(r)$ times bigger, so the total space is essentially the same. However, we only apply a single hash function, processing each element in constant time regardless of r , and the overall algorithms just get simpler. The idea is generic and can be applied to other algorithms. We will also apply it to Broder's original min-hash algorithm [6] to estimate set similarity.

Fast practical hash functions with strong concentration bounds were recently proposed by Aamand *et al.* [1]. Using their hashing scheme, Tabulation-1Permutation hashing, we obtain a very fast implementation of the above streaming algorithms, suitable for online processing of high volume data streams.

To illustrate a streaming scenario where the constant in the processing time is critical, consider the Internet. Suppose we want to process packets passing through a high-end Internet router. Each application only gets very limited time to look at the packet before it is forwarded. If it is not done in time, the information is lost. Since processors and routers use some of the same technology, we never expect to have more than a few instructions available. Slowing down the Internet is typically not an option. The papers of Krishnamurthy *et al.* [13] and Thorup and Zhang [17] explain in more detail how high speed hashing is necessary for their Internet traffic analysis. Incidentally, the hash function we use from [1] is a bit faster than the ones from [13, 17], which do not provide strong concentration bounds.

1.1 Experiments

To demonstrate that our approach performs well in practice, we perform experiments showing the strength of Tabulation-1Permutation for counting distinct elements in a stream. We compare with the fastest known strongly universal hash functions and with other commonly used hash functions such as MurmurHash3 [2] and BLAKE3 [12]. Without the use of independent repetitions, we demonstrate that Tabulation-1Permutation provides more reliable estimates than the fast strongly universal hash functions. Moreover, the implementation with Tabulation-1Permutation is faster than when using MurmurHash3 and BLAKE3. In fact, BLAKE3 was approximately 150 times slower than Tabulation-1Permutation, so we disregard it in our experiments. On the other hand, the implementation with Tabulation-1Permutation is both faster and provides better estimates than when implementing the algorithm with the strongly universal hash functions and independent repetitions.

We include two figures from Section 5 (which will be explained in more detail in that section). Figure 1a shows the relative error incurred by different hash functions over 5×10^4 experiments on a synthetic data set. We see that when implementing the algorithms with classic strongly universal hash functions like Multiply-Shift and Multiply-Mod-Prime, some of the estimates are significantly off. Figure 1b shows the running time per experiment for each of the hash functions tested. We see that Multiply-Shift is the only hashing scheme that outperforms Tabulation-1Permutation in regards to speed. However, in order to eliminate the outliers of Multiply-Shift seen in Figure 1a, we need to run independent repetitions, making Tabulation-1Permutation much faster. See Section 5 for details.

In this paper, we have chosen bottom- k sampling and threshold sampling for our algorithms, see Section 3 for details. There are several other estimators and algorithms for counting distinct elements. For more details, a thorough survey by Harmouch and Naumann [11] provides experimental data on the choice of algorithm for counting distinct elements.

Remark. It is important to note that no amount of experiments can prove that a hashing scheme performs well on all possible data, and finding the problematic data sets for a given hash family is often a non-trivial task¹. The results from [1] show that Tabulation-1Permutation performs well on *any* possible data set with high probability. In other words, if implementing the above streaming algorithms with Tabulation-1Permutation, we no longer have to cross our fingers that the data sets encountered does not have hidden structure which interacts badly with the hashing scheme. Furthermore, Tabulation-1Permutation is very fast, so this new guarantee comes with no compromise on the speed of the algorithms.

Remark. An interesting statistical artifact appearing in our experiments is the following. Looking at the accuracy of the estimators when implemented with Multiply-Shift and Multiply-Mod-Prime, it appears that for certain structured data sets, they provide even better estimates than one would obtain with fully random hashing. This could lead to the false impression that their performance is *always* better than implementing the estimators with something

¹For Multiply-Shift and Multiply-Mod-Prime, we have concrete examples of data sets on for which they fail, as is evident from Figure 1a. Moreover, in [5] the authors provide concrete bad data sets for MurmurHash3.

like Tabulation-1Permutation. Since the variance of the estimators are the same for all the seeded hash functions, it follows that if a hashing scheme yields estimators that are 'too good', it must inevitably fail occasionally and provide estimates that are far off. We will see examples of this in Section 5 and discuss it further in Section 5.5.

2 HASHING AND CONCENTRATION

In the present paper, the crucial component of our algorithms is a random hash function $h: U \rightarrow [0, 1)$ mapping some key universe U , e.g. 64-bit keys, uniformly into $R = [0, 1)$. The application is the following. Let $S \subset U$ be a set of keys and $p \in [0, 1)$ a threshold value. We wish to compute the number X of keys of S that hash below p , i.e.

$$X = |\{s \in S \mid h(s) < p\}|.$$

Here p could be an unknown function of S , but p should be independent of the random hash function h . Since the mean μ of x is $\mathbb{E}[X] = |S|p$, we may estimate the size of S by X/p with precision increasing in the concentration of X around its mean. In particular, we are interested in the probability δ that X deviates from μ by more than a factor $\varepsilon > 0$, i.e., the probability $\delta = \Pr[|X - \mu| \geq \varepsilon\mu]$.

If the hash function h is fully random, we get the classic Chernoff concentration bounds on X (see, e.g. [14]):

$$\Pr[X \geq (1 + \varepsilon)\mu] \leq \exp(-\varepsilon^2\mu/3) \text{ for } 0 \leq \varepsilon \leq 1, \quad (1)$$

$$\Pr[X \leq (1 - \varepsilon)\mu] \leq \exp(-\varepsilon^2\mu/2) \text{ for } 0 \leq \varepsilon \leq 1. \quad (2)$$

Unfortunately, we cannot implement fully random hash functions as it requires space as big as the universe.

2.1 Strongly Universal Hashing

To get something implementable in practice, Wegman and Carter [18] proposed strongly universal hashing.

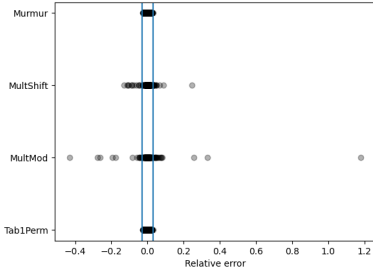
Definition 2.1 (Strongly Universal Hashing). A hash function $h: U \rightarrow R$ is *strongly universal* if for every pair of distinct keys $x, y \in U$, the distribution of $(h(x), h(y))$ is uniform on R^2 .

Many common hash functions are strongly universal. Worth mentioning is the Multiply-Shift hash function [8]. The textbook example of a strongly universal hash function hashing into $[0, 1)$ is the Multiply-Mod-Prime hash function [7]. The Multiply-Mod-Prime hash function picks a large prime \wp and two uniformly random numbers $a, b \in \mathbb{Z}_\wp$. Then $h_{a,b}(x) = ((ax + b) \bmod \wp) / \wp$ is strongly universal from $U \subseteq \mathbb{Z}_\wp$ to $R = \{i/\wp \mid i \in \mathbb{Z}_\wp\} \subset [0, 1)$. Obviously this hash function is not uniform on $[0, 1)$ as we considered above, but for any $p \in [0, 1)$, we have $\Pr[h(x) < p] \approx p$ with equality if $p \in R$. Below we ignore this deviation from uniformity on $[0, 1)$.

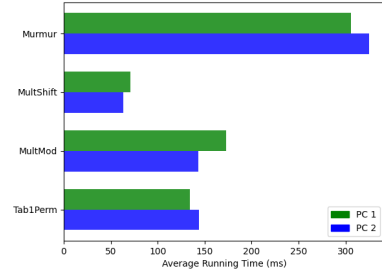
Assuming we have a strongly universal hash function $h: U \rightarrow [0, 1)$, we again let X be the number of elements from S that hash below p . Then $\mu = \mathbb{E}[X] = |S|p$ and because the hash values are 2-independent, we have $\text{Var}[X] \leq \mathbb{E}[X] = \mu$. Therefore, by Chebyshev's inequality,

$$\Pr[|X - \mu| \geq \varepsilon\mu] \leq 1/(\varepsilon^2\mu). \quad (3)$$

As $\varepsilon^2\mu$ gets large, we see that the error probability of strongly universal hashing is much higher than the Chernoff bounds with fully random hashing. However, as described in Section 3.4, it is



(a) The relative error when estimating the number of distinct elements with various hash functions using no repetitions.



(b) The average running time per experiment with the different hash functions.

Figure 1: Relative error and timing when estimating the number of distinct elements using various hash functions on synthetic data. These experiments, did not use independent repetitions.

still possible to guarantee high concentration by aiming for a constant error probability like $\delta = 1/4$ and then using the median over independent repetitions of the estimation to reduce the error probability.

2.2 Strongly Concentrated Hashing

In this paper we discuss the benefits of hash functions with strong concentration akin to that of fully random hashing.

Definition 2.2. A hash function $h : U \rightarrow [0, 1)$ is *strongly concentrated with added error probability* \mathcal{E} if for any key set $S \subseteq U$, threshold $p \in [0, 1)$, and $0 < \epsilon \leq 1$, the number X of elements from S hashing below p satisfies

$$\Pr[|X - \mu| \geq \epsilon\mu] = 2 \exp(-\Omega(\epsilon^2\mu)) + \mathcal{E},$$

where $\mu = p|S|$. If $\mathcal{E} = 0$, we say that h is *strongly concentrated*.

It is worth noting that a fully random hash function is strongly concentrated. Another way of viewing the added error probability \mathcal{E} is as follows. We have strong concentration as long as we do not aim for error probabilities below \mathcal{E} , so if \mathcal{E} is sufficiently low, we can simply ignore it. In other words, except for an error term of \mathcal{E} , the hash function performs asymptotically as well as a random hash function.

What makes this definition interesting in practice is that Aamand et al. [1] recently presented a practical hash function with small constant running time that for a universe $U = [u] = \{0, \dots, u-1\}$ is strongly concentrated with added error probability $u^{-\gamma}$ for any constant γ . For our applications, this term is so small that we can ignore it as the universe sets we consider are huge.

2.3 Tabulation-1Permutation

The hash function introduced by Aamand et al. [1] we consider in this paper is Tabulation-1Permutation. For the applications of this paper it enjoys the benefits of being strongly concentrated with a negligible added error probability, comparable in speed to the fastest hash functions on the market, and simple to implement.

We note that [1] also presents a slightly slower scheme, Tabulation-Permutation, which offers far more general concentration bounds than those for Tabulation-1Permutation in Theorem

```

1  uint64_t T[4][65536];
2  uint64_t P[65536];
3
4  uint64_t Tab1Perm(uint64_t x) {
5      uint64_t y; int i;
6      y = 0;
7      for (i = 0; i < 4; ++i) {
8          y ^= R[i][(uint16_t) x];
9          x = x >> 16;
10     }
11     return y ^ P[(uint16_t) y];
12 }

```

Figure 2: The C-code for the evaluation of Tabulation-1Permutation with 4 characters for 64-bit keys.

2.3. However, Tabulation-1Permutation is faster and sufficient for the strong concentration needed for our streaming applications.

2.3.1 Implementation. Tabulation-1Permutation obtains its power and speed using certain character tables in fast cache. The scheme views a keys from a universe $U = [u] = \{0, \dots, u-1\}$ as consisting of a small, constant number c of characters from some alphabet Σ , that is, $U = \Sigma^c$. For 64-bit keys, this could be $c = 4$ characters of 16 bits each. For our applications, we only consider the case where the hash values belong to the same universe U . Tabulation-1Permutation needs $c + 1$ character tables mapping characters to hash values. To compute the hash value of a key, we make one lookup for each of the $c + 1$ tables and perform $O(c)$ fast AC⁰ operation to extract the characters and XOR the hash values. The character tables can be populated with an $O(\log u)$ -independent pseudo-random number generator, needing a random seed of $O(\log^2 u)$ bits. As such, Tabulation-1Permutation is simple to implement and evaluate (see Figure 2). It takes up a dozen lines of code and is extremely fast. For further details on the inner workings of Tabulation-1Permutation see [1]. For details regarding running time of Tabulation-1Permutation in practice see Section 5 as well as [1, Section 1.7].

2.3.2 Strong Concentration. The exact concentration results regarding Tabulation-1Permutation [1, Theorem 1.3] are far too general for our purposes. We instead state a version simplified for the purposes of this exposition. Here we identify a hash value from $[u]$ as a fraction in $[0, 1)$.

THEOREM 2.3. *Let $h: [u] \rightarrow [0, 1)$ be a Tabulation-1Permutation hash function with $[u] = \Sigma^c$, $c = O(1)$, and let $\gamma > 0$ be fixed. For any key set $S \subset [u]$, threshold $p \in [0, 1)$, and $0 < \varepsilon \leq 1$, the number X of elements from S hashing below p satisfies*

$$\Pr[|X - \mu| \geq \varepsilon\mu] = 2 \exp(-\Omega(\mu\varepsilon^2)) + 1/u^\gamma.$$

Thus, for our applications, we are theoretically guaranteed that Tabulation-1Permutation will perform asymptotically as well as a random hash function except for an error term of $1/u^\gamma$. However, it is a crucial difficulty in said applications that the universe size is huge, so the error term is indeed negligible.

2.3.3 Computer Dependent Versus Problem Dependent Resources. We view the resources used for Tabulation-1Permutation as computer dependent rather than problem dependent. More precisely, you should pick the number of characters, c , and the size of the character alphabet, $|\Sigma|$, depending on the computer's architecture. For hashing 64-bit keys one often picks either $c = 4$ or $c = 8$ which yields a space usage of $9 \times 2^8 \times 8$ bytes (less than 20 KB) and $5 \times 2^{16} \times 8$ bytes (less than 3 MB), respectively. An important property is that once the tables are populated they will never be overwritten. This means that the cache does not get dirty, that is, different computer cores can access the tables and not worry about consistency.

3 COUNTING DISTINCT ELEMENTS WITH STRONGLY CONCENTRATED HASHING

Consider a sequence (or stream) of keys $x_1, \dots, x_s \in U$ from some universe U , where each element may appear multiple times. The problem of *counting distinct elements* in such a stream is the following. Using only little space, we wish to estimate the number n of distinct keys in the stream. We are given parameters $0 < \varepsilon, \delta < 1$, and the goal is to create an estimator, \hat{n} such that $(1 - \varepsilon)n \leq \hat{n} \leq (1 + \varepsilon)n$ with probability at least $1 - \delta$. It is assumed that the space required to store all n distinct elements of the stream is much too large. In practice we often want ε to be a small constant, e.g. $\varepsilon = 1\%$, while δ should be so small that we are all but guaranteed that $(1 - \varepsilon)n \leq \hat{n} \leq (1 + \varepsilon)n$, e.g. $\delta = 2^{-30}$.

In all previous work, hash functions have been a vital tool to create such estimators. Typically, strongly universal hash functions or hash functions with no theoretical guarantees have been used. In this section, we demonstrate that with strongly concentrated hash functions, we achieve both strong theoretical guarantees and simpler, faster algorithms for the counting distinct elements problem.

3.1 Bottom- k Sampling

In our estimation, we follow the classic approach of Bar-Yossef et al. [3], which was later revised by Beyer et al. [4] to introduce their unbiased version of the estimator. We proceed as follows. Let $h: U \rightarrow [0, 1)$ be a random hash function, and assume for simplicity that h is collision-free over U . For some $k > 1$, assumed to be

significantly smaller than n , we process each element x_i in order, maintaining the k smallest distinct hash values $h(x_i)$ of the stream. Let $x_{(k)}$ be the key with the k th smallest hash value under h , and let $h_{(k)} = h(x_{(k)})$. As in [3], our estimator for n is then $\hat{n} = k/h_{(k)}$. It is worth noting that [3] suggests several other estimators, but the points we will make below apply to all of them. We call this the bottom- k estimator for counting distinct elements.

The point of using a hash function h is that all occurrences of a given key x in the stream get the same hash value. Thus, if S is the set of distinct keys, $h_{(k)}$ is the k th smallest hash value from S . In particular, \hat{n} depends only on S , not on the frequencies of the elements of the stream.

We would like $\hat{n} = k/h_{(k)}$ to be concentrated around n . For any probability $p \in [0, 1]$, let $X^{<p}$ denote the number of elements from S that hash below p . Let $p_- = k/((1 + \varepsilon)n)$ and $p_+ = k/((1 - \varepsilon)n)$. Note that both p_- and p_+ are independent of the random hash function h . Now,

$$\hat{n} = k/h_{(k)} \leq (1 - \varepsilon)n \iff X^{<p_+} < k = (1 - \varepsilon)\mathbb{E}[X^{<p_+}], \quad (4)$$

$$\hat{n} = k/h_{(k)} > (1 + \varepsilon)n \iff X^{<p_-} \geq k = (1 + \varepsilon)\mathbb{E}[X^{<p_-}]. \quad (5)$$

These observations form a good starting point for applying probabilistic tail bounds using different types of hash functions as we describe in the following.

3.2 Strongly Universal Hashing and Independent Repetitions

Suppose we instantiate the bottom- k estimator with a strongly universal hash function $h: U \rightarrow [0, 1)$, perhaps the most common strategy for getting theoretical guarantees on the performance. In that case, the hash values of any two distinct keys are independent, so as noted in Section 2.1, $\text{Var}[X^{<p}] \leq \mathbb{E}[X^{<p}] = np$ for every $0 < p < 1$. Thus, applying Chebyshev's inequality (as we did in (3)) and (4),

$$\begin{aligned} \Pr[\hat{n} \leq (1 - \varepsilon)n] &= \Pr[X^{<p_+} \leq (1 - \varepsilon)\mathbb{E}[X^{<p_+}]] \\ &\leq 1/(np\varepsilon^2) = (1 - \varepsilon)/(k\varepsilon^2). \end{aligned}$$

Similarly, using (5) we get

$$\Pr[\hat{n} \geq (1 + \varepsilon)n] \leq (1 + \varepsilon)/(k\varepsilon^2).$$

Assuming $\varepsilon < 1$, it follows that

$$\Pr[|n - \hat{n}| \geq \varepsilon n] \leq 2/(k\varepsilon^2).$$

To get the desired error probability δ , we could now set $k = 2/(\delta\varepsilon^2)$. However, if δ is small, e.g., $\delta = 1/u$, k becomes much too large. To solve this issue, the standard approach is, as in [3], to apply classic *median trick*. Instead of aiming for a small error probability right away, we start by aiming for a constant error probability δ_0 . Here we use $\delta_0 = 1/4$ for simplicity. Then it suffices to maintain the $k_0 = 2/(\delta_0\varepsilon^2) = 8/\varepsilon^2$ smallest hash values. With these parameters, k_0 and δ_0 , we now sample r independent estimators for n , $\hat{n}_1, \dots, \hat{n}_r$, for some r to be determined later, by repeating the algorithm r times with fresh randomness. For our final estimator \hat{n} we return the median of $\hat{n}_1, \dots, \hat{n}_r$.

Now, for each $1 \leq i \leq r$, $\Pr[|\hat{n}_i - n| \geq \varepsilon n] \leq 1/4$ and these events are independent. If $|\hat{n} - n| \geq \varepsilon n$, then $|\hat{n}_i - n| \geq \varepsilon n$ for at

least half of the $1 \leq i \leq r$. By the standard Chernoff bound (1), this probability can be bounded by

$$\Pr[|\hat{n} - n| \geq \varepsilon n] \leq \exp(-r/4/3) = \exp(-r/12).$$

Setting $r = 12 \ln(1/\delta)$, we get the desired error probability $1/\delta$. The total number of hash values stored is then

$$k_0 r = 96 \ln(1/\delta)/\varepsilon^2 = \Theta(\ln(1/\delta)/\varepsilon^2).$$

3.3 Utopia: Fully Random Hashing

Suppose that we could implement a fully random hash function $h: U \rightarrow [0, 1]$. In that case, instantiating the bottom- k estimator with h would yield excellent guarantees. Combining (4) and (5) with the standard Chernoff bounds (2) and (1), respectively, with $0 < \varepsilon \leq 1$ yield the bounds

$$\begin{aligned} \Pr[\hat{n} \leq (1 - \varepsilon)n] &< \exp\left(-\frac{k\varepsilon^2}{2(1 - \varepsilon)}\right), \\ \Pr[\hat{n} \geq (1 + \varepsilon)n] &\leq \exp\left(-\frac{k\varepsilon^2}{3(1 + \varepsilon)}\right). \end{aligned}$$

Hence,

$$\Pr[|\hat{n} - n| \geq \varepsilon n] \leq 2 \exp(-k\varepsilon^2/6). \quad (6)$$

Thus, to get error probability δ , we just use $k = 6 \ln(2/\delta)/\varepsilon^2$. There are several reasons why this is much better than the above approach using 2-independence and independent repetitions. It avoids the independent repetitions, so instead of applying $r = \Theta(\log(1/\delta))$ hash functions to each key we just need one, and with independent repetitions, we are tuning the algorithm depending on ε and δ , whereas with a fully-random hash function, we get the concentration from (6) for every $0 < \varepsilon \leq 1$.

The only caveat is that fully-random hash functions cannot be implemented in practice. In some applications, cryptographic hash functions, or popular hash functions such as MurmurHash3 [2] have been applied. On datasets which have high entropy, such hash functions will perform well and appear “random”, however, there is no guarantee that this will hold for every choice of dataset. Importantly, it is impossible to predict how such a hash function will do on a particular structured dataset.

3.4 Strongly Concentrated Hashing

Abandoning the infeasible fully random hashing for strongly concentrated hashing, let $h: U \rightarrow [0, 1]$ be a strongly concentrated hash function with error term \mathcal{E} . Instantiating the bottom- k estimator with h , we may apply the same calculations as above. It then follows that for $0 < \varepsilon \leq 1$,

$$\begin{aligned} \Pr[\hat{n} \leq (1 - \varepsilon)n] &= \exp\left(-\Omega\left(\frac{k\varepsilon^2}{1 - \varepsilon}\right)\right) + \mathcal{E}, \\ \Pr[\hat{n} \geq (1 + \varepsilon)n] &= \exp\left(-\Omega\left(\frac{k\varepsilon^2}{1 + \varepsilon}\right)\right) + \mathcal{E}, \end{aligned}$$

so

$$\Pr[|\hat{n} - n| \geq \varepsilon n] = 2 \exp\left(-\Omega\left(k\varepsilon^2\right)\right) + O(\mathcal{E}). \quad (7)$$

To obtain the error probability $\delta = \omega(\mathcal{E})$, we again need to store $k = O(\log(1/\delta)/\varepsilon^2)$ hash values. Within a constant factor this means that we use the same total number using 2-independence

and independent repetitions, and we still retain the following advantages from the fully random case.

- By avoiding repetitions, we save a factor $\Theta(\log(1/\delta))$ in running time.
- We avoid tuning the algorithm for a given ε and δ . Instead we get the concentration from (7) for every $0 < \varepsilon \leq 1$.

3.4.1 Instantiating with Tabulation-1Permutation. Let us relate the above discussion to the Tabulation-1Permutation hash function by Aamand et al. [1]. It follows by Theorem 2.3 that implementing the bottom- k estimator with Tabulation-1Permutation would imply that for any $\gamma > 0$,

$$\Pr[|\hat{n} - n| \geq \varepsilon n] = 2 \exp\left(-\Omega\left(k\varepsilon^2\right)\right) + O(1/u^\gamma).$$

Since the universe size u is usually huge, the error term $O(1/u^\gamma)$ is negligible. Hence, from a practical perspective, the concentration is as good as fully random up to constant factors in the exponent of the exponential tail.

3.5 Implementing Bottom- k and an Alternative

Implementing the bottom- k estimator in practice requires maintaining the k smallest hash values. The most obvious and widely used approach is to use a priority queue. For instance, this is used in the survey of Harmouch and Naumann [11]. If the input arrives in random order then we need to update the priority queue $O(k \log n)$ times which gives a total running time of $O(n + k \log k \log n)$, and with the reasonable assumption that $k = O(n/\log(n)^2)$ we get a running time of $O(n)$.

Unfortunately, it is not reasonable to assume that the data arrives in random order and thus, for hash functions that are not strongly concentrated we might need to update the priority queue much more! But for strongly concentrated hash functions we can show that the priority queue only needs to be updated $O(k \log n)$ times no matter the input. This difference can cost as much as a factor of $O(\log k)$.

3.5.1 Threshold Estimator. Alternatively, we could use a related, but different sketch of Var-Yossef et al. [3], which is more efficient. The algorithm identifies the smallest b such that the number $X^{<1/2^b}$ of keys hashing below $1/2^b$ is at most k . For the online processing of the stream, this means that we increment b whenever $X^{<1/2^b} > k$. At the end, we return $2^b X^{<1/2^b}$. The analysis of this estimator is similar to the analysis of the bottom- k estimator. Using strongly concentrated hashing, we get the same advantage of avoiding independent repetitions: In [3] they achieve error probability δ by running $\Theta(\log(1/\delta))$ independent experiments, each storing up to $k = \Theta(1/\varepsilon^2)$ hash values, whereas we achieve the same error probability by running only a single experiment with a strongly concentrated hash function storing $k = O(\log(1/\delta)/\varepsilon^2)$ hash values. The total number of hash values stored is the same, but asymptotically, we save a factor $\Theta(\log(1/\delta))$ in time.

4 SET SIMILARITY WITH STRONGLY CONCENTRATED HASHING

Another setting where the k smallest hash values of a key set is often used is for the problem of *set similarity*. Consider two subsets $A, B \subset U$ of a universe $U = [u] = \{0, 1, \dots, u - 1\}$. A common

metric for similarity between two such sets is the *Jaccard similarity*, $J(A, B) = |A \cap B| / |A \cup B|$. In this section, we estimate the Jaccard similarity using strongly concentrated hashing, applying similar techniques to the previous section.

We consider Broder's [6] original algorithm for set similarity which we sketch for completeness. As in the previous section, let $h: [u] \rightarrow [0, 1)$ be a hash function assumed to be collision free. We define the bottom- k sample $\text{MIN}_k(S)$ of a set $S \subseteq [u]$ to be the k elements of S with the smallest hash values under h . We assume here that $k \leq n = |S|$. If h is fully random, then $\text{MIN}_k(S)$ is a uniformly random subset of S of size k , and if $T \subseteq S$, we may estimate the frequency $f = |T|/|S|$ as $|\text{MIN}_k(T) \cap \text{MIN}_k(S)|/k$. In [6], Broder uses this observation to estimate the Jaccard similarity between two sets $A, B \subseteq [u]$ as follows. Given the bottom- k samples from A and B , the bottom- k sample of their union may be constructed as $\text{MIN}_k(A \cup B) = \text{MIN}_k(\text{MIN}_k(A) \cup \text{MIN}_k(B))$. Then the Jaccard similarity is estimated as $\hat{J}(A, B) = |\text{MIN}_k(A \cup B) \cap \text{MIN}_k(A) \cap \text{MIN}_k(B)|/k$.

For the hash function, h , Broder [6] first considers fully random hashing and this particular case is very well understood. In this case $\text{MIN}_k(S)$ is a fully random sample of k distinct elements from S . However, fully random hashing is not implementable in practice. Broder also sketches some alternatives with realistic hash functions. Continuing this line of work, Thorup [16] showed that using strongly universal hashing, we get the same expected error as with fully random hashing. Writing $f = J(A, B)$ and $\hat{f} = \hat{J}(A, B)$, his precise result is that $\mathbb{E}[|f - \hat{f}|] = O(1/\sqrt{fk})$. Similarly to the problem of counting distinct elements, in order to obtain an estimator satisfying a similar bound with some high probability at least $1 - \delta$, we have to perform $O(\log(1/\delta))$ independent repetitions and use the median of the estimators as the final estimate. Our next result shows that when implementing the hashing using a scheme with strong concentration bounds, like Tabulation-1Permutation, we obtain the a similar error bound by running the algorithm just a single time, but with a sketch which is $O(\log(1/\delta))$ times larger. Again, the total space usage is the same, but we effectively eliminate the need for independent repetitions, thus saving a factor of $O(\log(1/\delta))$ in time.

Our analysis follows the simple union-bound approach from [16]. It is simpler to study the case where we are sampling from a set S and want to estimate the frequency $f = |T|/|S|$ of a subset $T \subseteq S$. Let $h_{(k)}$ be the k th smallest hash value from S as in the above algorithm for estimating distinct elements. For any p let $Y^{\leq p}$ be the number of elements from T with hash value at most p . Then $|T \cap \text{MIN}_k(S)| = Y^{\leq h_{(k)}}$ which is our estimator for fk .

THEOREM 4.1. *For $\varepsilon \leq 1$, if h is strongly concentrated with added error probability \mathcal{E} , then*

$$\Pr \left[|Y^{\leq h_{(k)}} - fk| > \varepsilon fk \right] = 2 \exp(-\Omega(fk\varepsilon^2)) + O(\mathcal{E}). \quad (8)$$

PROOF. Let $n = |S|$. We already saw in (7) that for any $\varepsilon_S \leq 1$, $P_S = \Pr \left[|1/h_{(k)} - n/k| \geq \varepsilon_S n/k \right] \leq 2 \exp(-\Omega(k\varepsilon_S^2)) + O(\mathcal{E})$. Thus, with $p_- = k/((1 + \varepsilon_S)n)$ and $p_+ = k/((1 - \varepsilon_S)n)$, we have $h_{(k)} \in [p_-, p_+]$ with probability $1 - P_S$, and in that case, $Y^{\leq p_-} \leq Y^{\leq h_{(k)}} \leq Y^{\leq p_+}$.

Let $\mu^- = \mathbb{E}[Y^{\leq p_-}] = fk/(1 + \varepsilon_S) \geq fk/2$. By strong concentration, for any $\varepsilon_T \leq 1$, we get that

$$\begin{aligned} P_T^- &= \Pr \left[Y^{\leq p_-} \leq (1 - \varepsilon_T)\mu^- \right] \leq 2 \exp(-\Omega(\mu^- \varepsilon_T^2)) + \mathcal{E} \\ &= 2 \exp(-\Omega(fk\varepsilon_T^2)) + \mathcal{E}. \end{aligned}$$

Thus

$$\Pr \left[Y^{\leq h_{(k)}} \leq \frac{1 - \varepsilon_T}{1 + \varepsilon_S} fk \right] \leq P_T^- + P_S.$$

Likewise, with $\mu^+ = \mathbb{E}[Y^{\leq p_+}] = fk/(1 - \varepsilon_S)$, for any ε_T , we get that

$$\begin{aligned} P_T^+ &= \Pr \left[Y^{\leq p_+} \geq (1 + \varepsilon_T)\mu^+ \right] \leq 2 \exp(-\Omega(\mu^+ \varepsilon_T^2)) + \mathcal{E} \\ &= 2 \exp(-\Omega(fk\varepsilon_T^2)) + \mathcal{E}. \end{aligned}$$

and

$$\Pr \left[Y^{\leq h_{(k)}} \geq \frac{1 + \varepsilon_T}{1 - \varepsilon_S} fk \right] \leq P_T^+ + P_S.$$

To prove the theorem for $\varepsilon \leq 1$, we set $\varepsilon_S = \varepsilon_T = \varepsilon/3$. Then $\frac{1 + \varepsilon_T}{1 - \varepsilon_S} \leq 1 + \varepsilon$ and $\frac{1 - \varepsilon_T}{1 + \varepsilon_S} \geq 1 - \varepsilon$. Therefore,

$$\begin{aligned} \Pr \left[|Y^{\leq h_{(k)}} - fk| \geq \varepsilon fk \right] &\leq P_T^- + P_T^+ + 2P_S \\ &\leq 8 \exp(-\Omega(fk\varepsilon_T^2)) + O(\mathcal{E}) \\ &= 2 \exp(-\Omega(fk\varepsilon_T^2)) + O(\mathcal{E}). \end{aligned}$$

This completes the proof of (8). \square

Similarly to the discussion in Section 3.4.1, if we implement the hashing as Tabulation-1Permutation, it follows from Theorem 2.3 and Theorem 4.1 that

$$\Pr \left[|Y^{\leq h_{(k)}} - fk| > \varepsilon fk \right] = 2 \exp(-\Omega(fk\varepsilon^2)) + O(u^{-\gamma}). \quad (9)$$

for any $\gamma = O(1)$. Ignoring the negligible term $O(u^{-\gamma})$, this is as good as with fully random hashing up to the constant delay the the exponential decrease.

As for the problem of counting distinct elements in a stream, in the online setting we may again modify the algorithm above to obtain a more efficient sketch. Assuming that the elements from S arrive in a stream, we again identify the smallest b such that the number of keys from S hashing below $1/2^b$, $X^{\leq 1/2^b}$, is at most k . We increment b by one whenever $X^{\leq 1/2^b} > k$ and in the end we return $Y^{\leq 1/2^b} / X^{\leq 1/2^b}$ as an estimator for f . The analysis of this modified algorithm is similar to the analysis provided above.

Remark. The case of set similarity illustrates the crucial importance of using a common hash function h as a source of randomness. In a distributed setting, different entities may generate the samples $\text{MIN}_k(A)$ and $\text{MIN}_k(B)$. As long as they agree on h , they only need to communicate the samples to estimate the Jaccard similarity of A and B . As noted before, for Tabulation-1Permutation h can be shared by exchanging a random seed of $O((\log u)^2)$ bits.

Brief synopsis of our C++ experiments								
Dataset	Algorithm	Measurement	Desired Error	Cardinality	Sketch-size	Experiments	Type	Plots
Synthetic	Bottom- k	Error	0.03	10^6	24,500	2×10^3	Single Rep.	3a
				5×10^5	24,500	5×10^4	Single Rep.	3b, 3c
				5×10^5	24,500	3×10^4	Median-Trick	6a
		Time	–	5×10^7	3500	10	Single Rep.	4a
5×10^7	3500			10	Median-Trick	4b		
Synthetic	Threshold sampling	Error	0.01	5×10^6	8×10^5	5×10^4	Single Rep.	7a, 7b
				2.5×10^5	1.6×10^5	5×10^5	Single Rep.	7c
				2.5×10^5	1.6×10^5	3×10^4	Median-Trick	6b
		Time	–	10^9	8×10^5	10	Single Rep.	8a
5×10^6	8×10^5			10	Median-Trick	8b		
Real-world	Threshold sampling	Error	0.03	1.8×10^6	17,500	10^4	Both	9a, 10a
				6.5×10^5	28,000	3×10^4	Both	9b, 10b
		Time	–	1.8×10^6	17,500	10	Both	9c, 10c
6.5×10^5	28,000			10	Both	9d, 10d		

Table 1: Overview of our experiments. All experiments are designed to compare Tabulation-1Permutation with various other hashing schemes when estimating distinct elements. The type of the experiment indicates whether the other hashing schemes use a single sketch, or 5 independent repetitions and the median trick. Tabulation-1Permutation always uses a single sketch. When using the median trick, the size of each sketch is reduced by a factor of 5, so the total space usage is always the same.

5 EXPERIMENTAL EVALUATION

In this section we experimentally evaluate hash functions with strong concentration bounds in the context of cardinality estimation. In particular, we use the recently announced Tabulation-1Permutation [1]. Our experiments address the following questions:

- Is it possible, in practice, to avoid independent repetitions by using hash functions with strong concentration bounds and still get reliable results?
- What are the implications on the running time of cardinality estimation algorithms, when using Tabulation-1Permutation?

In our experiments, we restrict the space usage of our algorithms to some parameter k . When we run our algorithm for counting distinct elements using the fast Multiply-Mod-Prime and Multiply-Shift hash functions, we report both the results of performing a single repetition per experiment with a sketch of size k , and the results of performing r independent repetitions per experiment each with a sketch of size k/r and outputting the median of the r estimations. The reason we use space k/r for each repetition, instead of “reusing” the space k , is that in a streaming setting the independent repetitions are being run simultaneously. We expect the first type of experiment to have about r times better running time, and the second type of experiment to produce more reliable results. Throughout our experiments, we adopt the widely used practice of performing $r = 5$ repetitions. On the other hand, when using Tabulation-1Permutation we only perform 1 repetition per experiment, effectively demonstrating that the estimation is still reliable without the need to spend more time for independent repetitions.

A brief synopsis of our experiments is presented in Table 1. In what follows, we first discuss our experimental setup and the implementation details. Then, we present our results for synthetic

data and the bottom- k algorithm discussed in previous sections. Then we present similar results for synthetic data and a standard threshold-sampling cardinality estimation algorithm. Finally, we present our results on real-world datasets.

5.1 Experimental Setup

Hardware. We performed our experiments on two computers. The first has an Intel(R) Core(TM) i7-8665U CPU @ 1.90GHz, 16 GB RAM, and is running on 64bit Windows 10. The second has an Intel(R) Core(TM) i5-8350U CPU @ 1.7 GHz, 8 GB RAM, and is running on 64bit Ubuntu 18.04.4 LTS.

Datasets. We use both synthetic and real-world datasets. Notice that theoretically we guarantee that, when using hash functions with strong concentration bounds, the probability of getting a big error is negligible (“big” is quantified in relation to the sketch size). In order to test such a claim, one needs to run hundreds of thousands of experiments and show that the result is always reliable. Therefore, the quality of the results is tested against small datasets, so that we can repeat the same experiment sufficiently many times. Big datasets with cardinality in the order of 10^9 are also employed, but only to test the running time.

Our synthetic datasets for testing accuracy are, therefore, sets with cardinalities ranging between 2.5×10^5 and 5×10^6 , while the synthetic datasets for testing running times have cardinalities ranging between 5×10^6 and 10^9 . We use both structured synthetic datasets, containing consecutive integers, and random synthetic datasets containing random 64-bit integers.

Our real-world datasets are geometric datasets; we are using the Openaddresses dataset, which is a public database connecting the

geographical coordinates with their postal addresses². To motivate cardinality estimation in this case, we partition the space in squares of fixed size, and compute the number of non-empty squares. We also experiment directly on the initial dataset. The cardinality of these datasets varies between 6.5×10^5 and 1.8×10^6 .

5.1.1 Algorithms. We use two different algorithms for cardinality estimation. First, we have the bottom- k algorithm [4] discussed in previous sections; in this case, we expect that the k -th smallest hash value is approximately $\frac{k}{n}$. The second algorithm, threshold-sampling, can be seen as the dual approach; that is, we keep all elements with hash values at most $p = \frac{n}{k}$, and expect their cardinality to be approximately k . Generally we do not know n in advance, so determining p is not straightforward; the way to solve this problem is discussed in Subsection 3.5. In our experiments we know n in advance, so we directly use the proper value of p .

5.1.2 Hash-Functions. For each algorithm, we test it using some of the most popular and fast hash functions employed in practice. Their output is always 64-bit unsigned integers. In particular, we use Tabulation-1Permutation [1], Multiply-Mod-Prime [7], Multiply-Shift [9], and MurmurHash3 [2]. We use Multiply-Mod-Prime and Multiply-Shift as they are very fast strongly universal hash functions. MurmurHash3 is also used as it is known to perform well in practice, even though it does not provide theoretical guarantees (e.g. in [5] the authors show how to break MurmurHash3).

We did not experiment with random polynomials of degree 100, or the cryptographic hash function BLAKE3 because they proved to be more than 60 and 155 times slower than any other method, respectively.

5.1.3 Randomness. The random seed needed for all hash functions was drawn from <https://www.random.org/>. For tabulation based methods, the seed was filled using a random polynomial of degree 100.

5.1.4 Implementation details. Most experiments are implemented in C++ 11. Some are implemented in Java and used the Metanome data profiling framework [15]. Metanome is a standard framework decoupled from the algorithms, which provides basic functionalities, such as input parsing and performance measurement³, having already been used in the context of cardinality estimation [10].

5.1.5 Evaluation Metrics. As the measure of estimation accuracy, we report the relative error. The relative error of an estimate \hat{n} of a quantity n is defined as $\frac{n-\hat{n}}{n}$. For measuring running time, we report the average time per experiment.

5.2 The Bottom- k Algorithm on Synthetic Data

In this subsection, we present our results on applying the bottom- k algorithm to synthetic data. These experiments are of three different kinds. The first uses one repetition per experiment for all hash functions, the second uses the previously mentioned Metanome framework, and the third uses the median trick with the Multiply-Shift and Multiply-Mod-Prime as described at the beginning of the section.

²<https://openaddresses.io/>

³www.metanome.de

5.2.1 One Repetition Per Experiment. We performed experiments on implementations of the Bottom- k algorithm, where we used only a single sketch for each hash function. The experiments are listed as rows 1, 2, and 4 of Table 1. All experiments were implemented in C++ and had $k = 24,500$, aiming at a relative error of at most 3% in all of the experiments (a suitable choice of k for a given relative error ϵ and a desired low error probability δ is suggested in [4]). The datasets used are structured datasets containing consecutive integers.

Testing for accuracy, for the dataset with cardinality 10^6 (row 1 of Table 1) almost all experiments were within the acceptable 3% error (Figure 3a). Only Multiply-Shift – the fastest hash function tested – had some experiments with relative error above 3%. It is to be expected that not too many outliers were observed since 2×10^3 experiments per hash function is not sufficient to consistently observe big errors. On the other hand, for the dataset with cardinality 5×10^5 (row 2 of Table 1) both Multiply-Mod-Prime and Multiply-Shift have some experiments with huge errors (about 120% and 25%, respectively). See Figure 3b.

Testing running times, we tested on a dataset of cardinality 5×10^7 (row 4 of Table 1) with $k = 3500$ (Figure 4a). We notice that the implementation with Tabulation-1Permutation is significantly faster than the other methods except for Multiply-Shift.

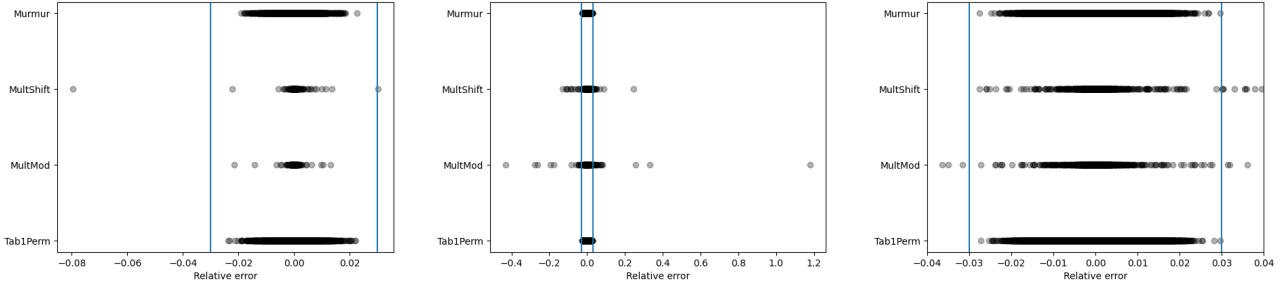
5.2.2 Metanome. Using Java and the Metanome data profiling framework, we also tested Tabulation-1Permutation against Multiply-Mod-Prime and MurmurHash3. The experiments were performed on very large datasets (size 5×10^8), with sketch size $k = 3.2 \times 10^5$, and desired error 1%. We ran 400 experiments for each hash-function.

The error results are shown in Figure 5. Once again we verify that in order to test reliability, a lot more experiments are needed; just 400 experiments were not enough to show any unreliable behavior of Multiply-Mod-Prime. However 400 experiments were enough to show that MurmurHash3 gives results that are not within the 1% error.

Concerning running times, MurmurHash3 required 14.15 seconds per experiment, Multiply-Mod-Prime required 14.4 seconds, and Tabulation-1Permutation required 15.65 seconds per experiment. A possible explanation for the slightly longer running time of Tabulation-1Permutation is that tabulation methods work best when their look-up tables can be efficiently stored in the L1-cache. When running on top of the Metanome framework, it may be the case that there was not enough space in the L1-cache to completely fit the look-up tables.

It is also interesting to notice that the running time for each experiment was not very stable. For Tabulation-1Permutation the time varied between 13.5 and 19 seconds, for Multiply-Mod-Prime between 12.5 and 17.5 seconds, and for MurmurHash3 the range was much larger (between 12 and 30 seconds). For Tabulation-1Permutation this is likely due to cache misses, while for the other two methods it is most likely due to the fact that their output did not look “random” enough and thus values had to be inserted in the bottom- k priority queue often; it can theoretically be proven that Tabulation-1Permutation does not suffer from such problems.

The MurmurHash3 function we used was the one from the official implementation of MurmurHash3 [2]. Additionally, we also

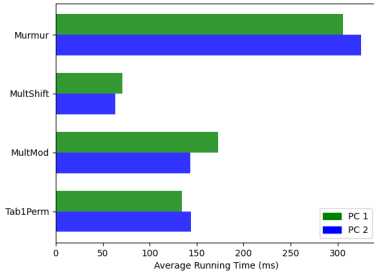


(a) Cardinality: 10^6
Experiments per hash-function: 2×10^3

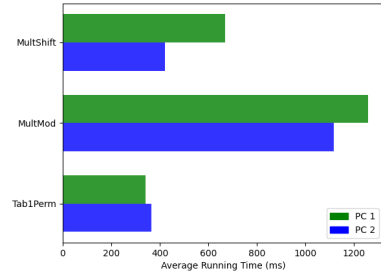
(b) Cardinality: 5×10^5
Experiments per hash-function: 5×10^4

(c) Cardinality: 5×10^5
Experiments per hash-function: 5×10^4

Figure 3: Relative error of the single-repetition experiments using the bottom- k algorithm ($k = 24,500$). Each dot represents an experiment, and the x -coordinate is the relative error. The more opaque the dots are, the more experiments had the corresponding relative error. The vertical blue lines indicate the desired 3% relative error.



(a) Single-repetition experiments with structured synthetic datasets (consecutive integers). Cardinality = 5×10^7 , $k = 3500$.



(b) Experiments on random synthetic datasets. With Multiply-Mod-Prime and Multiply-Shift, $(r, k) = (5, 700)$. With Tabulation-1Permutation, $(r, k) = (1, 3500)$. Cardinality = 5×10^7

Figure 4: Timing of bottom- k experiments (synthetic data).

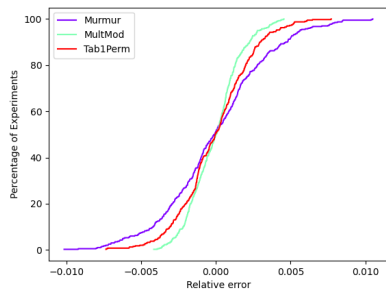


Figure 5: Cumulative plot of the relative errors of 400 experiments per hash-function, using the bottom- k algorithm on Metanome. Cardinality = 5×10^8 and $k = 3.2 \times 10^5$.

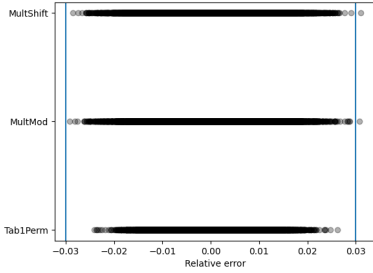
experimented in the same setting, using the implementation of MurmurHash3 used in a survey for cardinality estimation [10]. In

this case, we had some experiments with 15% errors. The reason is possibly that the two implementations use different constants for computing the hash value. This fact highlights the need for hash functions with strong theoretical guarantees.

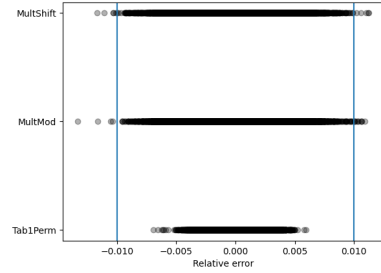
5.2.3 Applying the Median Trick. We performed experiments where independent repetitions and the median trick were used for the instantiations with Multiply-Shift and Multiply-Mod-Prime, while the implementation with Tabulation-1Permutation still used just a single sketch.

For accuracy, we performed 3×10^4 experiments on a dataset of cardinality 5×10^5 (row 3 of Table 1). The results are shown in Figure 6a. We observe that the estimations, when using Tabulation-1Permutation, are always within the 3% bound, and are generally more concentrated around the actual cardinality.

For speed, we ran the algorithms on datasets of cardinality 5×10^7 with $k = 3500$ (row 5 of Table 1). Here Tabulation-1Permutation was significantly faster, as seen in Figure 4b. Its average running time was 365ms, while Multiply-Shift required 1118ms and Multiply-Mod-Prime required 422ms.

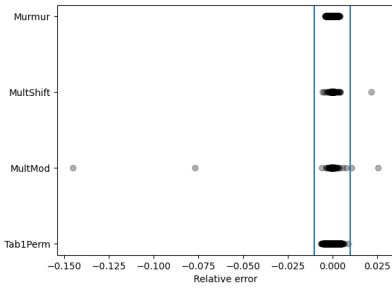


(a) **Bottom- k : Cardinality = 5×10^5**
 For Multiply-Mod-Prime and Multiply-Shift we have $(r, k) = (5, 4900)$. For Tabulation-1Permutation $(r, k) = (1, 24, 500)$.

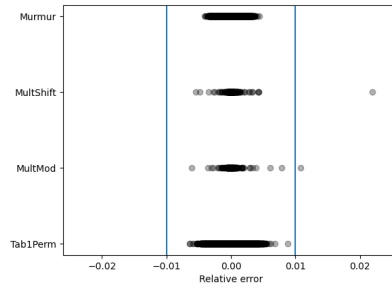


(b) **Threshold-sampling: Cardinality = 2.5×10^5**
 For Multiply-Mod-Prime and Multiply-Shift we have $(r, p) = (5, 0.128)$. For Tabulation-1Permutation $(r, p) = (1, 0.64)$.

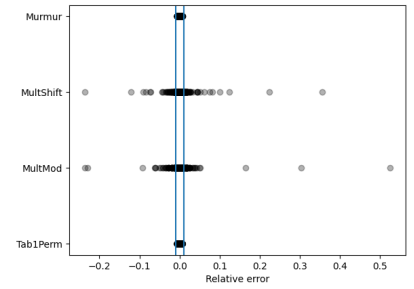
Figure 6: Relative error of 3×10^5 experiments for each hash-function, using the bottom- k and the threshold-sampling algorithm on random synthetic data. Multiply-Mod-Prime and Multiply-Shift used 5 repetitions per experiment, whereas Tabulation-1Permutation ran just once per experiment with a 5 times larger sketch. The plots should be interpreted as those in Figure 3.



(a) **Cardinality: 5×10^6**
 Experiments per hash-function: 5×10^4



(b) **Cardinality: 5×10^6**
 Experiments per hash-function: 5×10^4



(c) **Cardinality: 2.5×10^5**
 Experiments per hash-function: 5×10^5

Figure 7: Relative error of the single-repetition experiments using the threshold-sampling algorithm, using structured synthetic datasets (consecutive integers). The plots should be interpreted as those in Figure 3.

5.3 Threshold-Sampling on Synthetic data

In this subsection, we present our experiments on the threshold-sampling algorithm when using synthetic data. All experiments were performed in C++11. We first consider an implementation with one repetition per experiment for all hash functions. Secondly, we consider using the median trick with Multiply-Shift and Multiply-Mod-Prime.

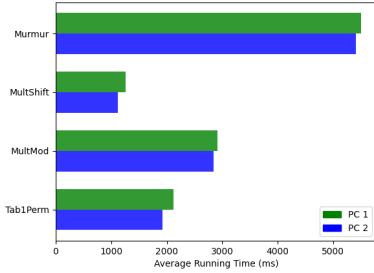
5.3.1 One Repetition Per Experiment. We performed experiments on implementations of the threshold-sampling algorithm using only a single sketch for each hash functions. Details of the experiments are listed as rows 6, 7, and 9 of Table 1. All of them were aiming for a relative error of at most 1%.

For accuracy, we performed experiments on a datasets with cardinality 5×10^6 (row 6 of Table 1). The results are plotted in Figure 7a. Only Multiply-Mod-Prime and Multiply-Shift gave results with relative error worse than 1% error (about 15% and 2%, respectively). If we zoom in (Figure 7b) we see that within the acceptable error

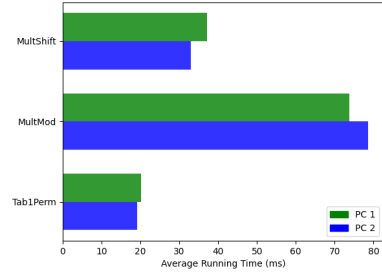
1% these two hash functions are far more accurate than all other. This is actually not at all surprising, as we remark in Section 5.5. Rather it is a direct consequence of the estimators having the same variance. We also performed experiments for datasets with cardinality 2.5×10^5 (row 7 of Table 1). The results are plotted in (Figure 7c). Here, we see a similar picture, but the errors are much higher. Namely, Multiply-Mod-Prime has experiments with more than 50% relative error, and Multiply-Shift has experiments with more than 35% relative error.

Concerning the running time, we run datasets of cardinality 10^9 (row 9 of Table 1). The results can be seen in Figure 8a, and it is clear that implementing the algorithm with Tabulation-1Permutation is significantly faster than with any other hash function except Multiply-Shift.

5.3.2 Applying the Median Trick. We performed experiments where independent repetitions and the median trick were used for the instantiations with Multiply-Shift and Multiply-Mod-Prime,



(a) Single repetition experiments on structured synthetic data (consecutive integers). Cardinality = 10^9 , $p = 8 \times 10^{-4}$.



(b) Experiments on random synthetic datasets. With Multiply-Mod-Prime and Multiply-Shift, $(r, p) = (5, 0.032)$. With Tabulation-1Permutation $(r, p) = (1, 0.16)$. Cardinality = 5×10^6 .

Figure 8: Timing of threshold-sampling experiments (synthetic data).

while the implementation with Tabulation-1Permutation still used just a single sketch.

For accuracy, we performed 3×10^4 experiments with a dataset of cardinality 2.5×10^5 (row 8 of Table 1). We used the threshold $p = 0.128$ for Multiply-Mod-Prime and Multiply-Shift and the threshold $p = 0.64$ for Tabulation-1Permutation. The results are shown in Figure 6b. Notice that the threshold p when performing 5 repetitions is 5 times smaller. This is to ensure that the elements of each of the 5 independent sketches only use a fifth of the total allowed space k . We observe that the estimations are much better concentrated when using Tabulation-1Permutation, always keeping within the 1% bound.

Testing for speed, we ran experiments on datasets of cardinality 5×10^6 (row 10 of Table 1). Here Tabulation-1Permutation is much faster with an average running time of 19ms, while Multiply-Shift required 33ms and Multiply-Mod-Prime required 78.5ms. The results can be seen in Figure 8b.

5.4 Experiments on Real-world Data

In this subsection, we present our experiments with real-world data. For these experiments, we used geometric datasets, namely the Openaddresses dataset. We experiment on this dataset in two different ways. First, we estimate the distinct elements of the dataset directly. Then, in order to better motivate counting distinct elements on this dataset, we partition the geometric space in squares of fixed size. Each geometric item is assigned to its corresponding square, and we count the number of non-empty squares. All the experiments aimed at error probability 3%. The experiments are listed as rows 11 through 14 of Table 1.

5.4.1 One Repetition Per Experiment. In the setting where we were only using a single sketch for all the hash functions, the results were as follows.

For accuracy, when running on both the initial dataset and the preprocessed dataset, Tabulation-1Permutation and MurmurHash3 were always within the 3% relative error bound on the initial dataset, while Multiply-Mod-Prime and Multiply-Shift deviated from this error bound in both cases. See Figure 9a and Figure 9b, respectively.

For the preprocessed data set some of these deviations were as big as 80% and 40% for Multiply-Mod-Prime and Multiply-Shift, respectively.

For running times, Tabulation-1Permutation is comparable with Multiply-Mod-Prime (in the original dataset it is slightly slower and in the preprocessed dataset it is slightly faster). It is also significantly slower than Multiply-Shift and significantly faster than MurmurHash3, in both cases. See Figures 9c and 9d for the results.

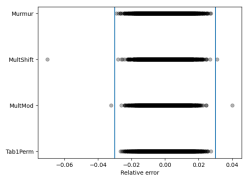
5.4.2 Applying the Median Trick. In the setting where we use the median trick when instantiating the estimator with the median trick, the results were as follows.

For accuracy, Tabulation-1Permutation (with a single repetition per experiment) is always within the 3% relative error bound, and is generally better concentrated. Multiply-Mod-Prime is within the error bounds only in the case of the preprocessed input, and Multiply-Shift is off the 3% error bound in both cases. See Figure 10a for the results.

For speed, Tabulation-1Permutation also outperforms the other two hash functions when we measure running time in this setting. In the original dataset, Tabulation-1Permutation needs 9.5ms per experiment, Multiply-Shift needs 16ms, and Multiply-Mod-Prime needs 36ms. See Figure 10c for details. In the preprocessed dataset, the situation is similar: Tabulation-1Permutation needs 5.5ms per experiment, Multiply-Shift needs 7ms, and Multiply-Mod-Prime needs 14ms. See Figure 10d for details.

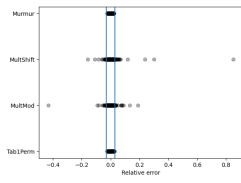
5.5 Remark on Concentration

An interesting observation related to the errors when running only a single sketch for all hash functions is the following: If we only focus our attention on the successful experiments, i.e., the outcomes with relative error less than ϵ , then on some data sets, Multiply-Mod-Prime and Multiply-Shift appear more accurate than Tabulation-1Permutation (see for instance Figure 3a and Figure 3c). Running only a few experiments, this could lead to the false impression that these hash functions are in fact *always* better than hash functions like Tabulation-1Permutations. However, the variance of the estimators are approximately the same for all the seeded hash



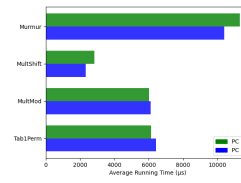
(a) Accuracy on real-world dataset.

Cardinality: 1.8×10^6
Experiments: 10^4



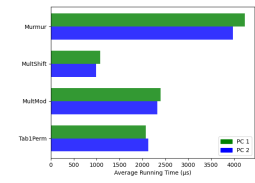
(b) Accuracy on real-world preprocessed dataset.

Cardinality: 6.5×10^5
Experiments: 3×10^4



(c) Timing on real-world dataset.

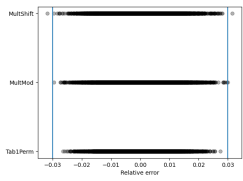
Cardinality: 1.8×10^6
Experiments: 10^4



(d) Timing on real-world preprocessed dataset.

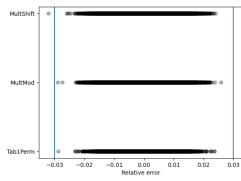
Cardinality: 6.5×10^5
Experiments: 3×10^4

Figure 9: Single repetition experiments on real-world datasets. Plots 9a and 9b should be interpreted as Figure 3.



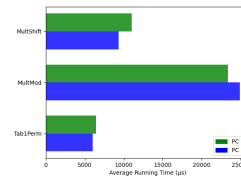
(a) Accuracy on real-world dataset.

Cardinality: 1.8×10^6
Experiments: 10^4



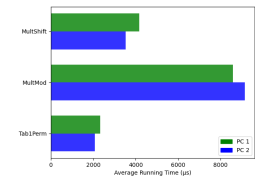
(b) Accuracy on real-world preprocessed dataset.

Cardinality: 6.5×10^5
Experiments: 3×10^4



(c) Timing on real-world dataset.

Cardinality: 1.8×10^6
Experiments: 10^4



(d) Timing on real-world preprocessed dataset.

Cardinality: 6.5×10^5
Experiments: 3×10^4

Figure 10: Median-trick (5 repetitions) experiments on real-world datasets. Plots 10a and 10b should be interpreted as Figure 3.

functions. This means that if we obtain these 'too good to be true' estimates most of the time, we must inevitably have some cases where the estimates are far off. This is precisely the behaviour that we see with Multiply-Mod-Prime and Multiply-Shift. On the other hand, as seen in Figure 3a and Figure 3c, Tabulation-1Permutation provides estimates that reliably lie within the 3% error margin, not extremely close to the precise cardinality, yet with no wild outliers.

6 CONCLUSION

In this paper we showed that the use of hash functions with strong concentration bounds, like Tabulation-1Permutation, can speed up streaming algorithms by avoiding time consuming independent repetitions, and still provide accurate statistical estimates with high probability. Specifically, we studied algorithms for estimating the number of distinct elements in a stream and the similarity between two large sets.

Our results are backed up by experiments which show that widely used hash functions like Multiply-Mod-Prime and Multiply-Shift do not exhibit similar behavior. On the other hand, when boosting the success probability of Multiply-Shift and Multiply-Mod-Prime using independent repetitions, the implementation with Tabulation-1Permutation both becomes faster and it provides more reliable estimates. Finally, the running time of Tabulation-1Permutation is better than that of other commonly used hash functions like MurmurHash3, which provided reliable estimates in our experiments but which have no similar general theoretical guarantees.

Evaluating the strength of a hashing scheme experimentally is an impossible task. The hash function may behave nicely on most data, but when certain badly structured data sets are encountered, it may fail miserably. This is an important reason why hash functions with strong theoretical guarantees are desirable. This paper demonstrates that when implementing the above streaming algorithms with Tabulation-1Permutation, we not only obtain such strong theoretical guarantees for the estimators; the algorithms also experience a significant speed-up.

ACKNOWLEDGMENTS

Research of all authors partly supported by Thorup's Investigator Grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation. Evangelos Kipouridis has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 801199.

REFERENCES

- [1] 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, STOC 2020*, pages 1265–1278. ACM, 2020.
- [2] Austin Appleby. Murmurhash3. <https://github.com/aappleby/smhasher/wiki/MurmurHash3>, 2016. Available at <https://github.com/aappleby/smhasher/wiki/MurmurHash3>.
- [3] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 1–10, 2002.

- [4] Kevin S. Beyer, Peter J. Haas, Berthold Reinwald, Yannis Sismanis, and Rainer Gemulla. On synopses for distinct-value estimation under multiset operations. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*, pages 199–210, 2007. doi: 10.1145/1247480.1247504. URL <https://doi.org/10.1145/1247480.1247504>.
- [5] Martin Bořlet. Breaking murmur: Hash-flooding dos reloaded. <https://embooss.github.io/blog/2012/12/14/breaking-murmur-hash-flooding-dos-reloaded/>, 2012. Available at <https://embooss.github.io/blog/2012/12/14/breaking-murmur-hash-flooding-dos-reloaded/>.
- [6] Andrei Z. Broder. On the resemblance and containment of documents. In *Proc. Compression and Complexity of Sequences (SEQUENCES)*, pages 21–29, 1997.
- [7] Larry Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979. Announced at STOC’77.
- [8] Martin Dietzfelbinger and Friedhelm Meyer auf der Heide. Dynamic hashing in real time. In Johannes Buchmann, Harald Ganzinger, and Wolfgang J. Paul, editors, *Informatik, Festschrift zum 60. Geburtstag von Günter Hotz*, volume 1 of *Teubner-Texte zur Informatik*, pages 95–119. Teubner / Springer, 1992. ISBN 978-3-8154-2033-1. doi: 10.1007/978-3-322-95233-2_7. URL https://doi.org/10.1007/978-3-322-95233-2_7.
- [9] Martin Dietzfelbinger, Torben Hagerup, Jyrki Katajainen, and Martti Penttonen. A reliable randomized algorithm for the closest-pair problem. *Journal of Algorithms*, 25(1):19–51, 1997.
- [10] Hazar Harmouch and Felix Naumann. Cardinality estimation: An experimental survey. *Proc. VLDB Endow.*, 11(4):499–512, 2017. doi: 10.1145/3186728.3164145. URL <http://www.vldb.org/pvldb/vol11/p499-harmouch.pdf>.
- [11] Hazar Harmouch and Felix Naumann. Cardinality estimation: An experimental survey. *Proceedings of the VLDB Endowment*, 11:499–512, 12 2017. doi: 10.1145/3164135.3164145.
- [12] Samuel Neves Jack O’Connor, Jean-Philippe Aumasson and Zooko Wilcox-O’Hearn. Blake3. <https://github.com/BLAKE3-team/BLAKE3>, 2020. Available at <https://github.com/BLAKE3-team/BLAKE3>.
- [13] Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang, and Yan Chen. Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the 3rd ACM SIGCOMM Internet Measurement Conference, IMC 2003, Miami Beach, FL, USA, October 27-29, 2003*, pages 234–247, 2003. doi: 10.1145/948205.948236. URL <https://doi.org/10.1145/948205.948236>.
- [14] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
- [15] Thorsten Papenbrock, Tanja Bergmann, Moritz Finke, Jakob Zwiener, and Felix Naumann. Data profiling with metanome. *Proc. VLDB Endow.*, 8(12):1860–1863, 2015. doi: 10.14778/2824032.2824086. URL <http://www.vldb.org/pvldb/vol8/p1860-papenbrock.pdf>.
- [16] 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.
- [17] 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.
- [18] Mark N. Wegman and Larry Carter. New classes and applications of hash functions. *Journal of Computer and System Sciences*, 22(3):265–279, 1981. Announced at FOCS’79.

Appendix C

Expander Graphs are Non-Malleable Codes

Expander Graphs Are Non-Malleable Codes

Peter Michael Reichstein Rasmussen

Basic Algorithms Research Copenhagen, University of Copenhagen, Denmark
pmrr@di.ku.dk

Amit Sahai

UCLA, Los Angeles, CA, USA
sahai@cs.ucla.edu

Abstract

Any d -regular graph on n vertices with spectral expansion λ satisfying $n = \Omega(d^3 \log(d)/\lambda)$ yields a $O\left(\frac{\lambda^{3/2}}{d}\right)$ -non-malleable code for single-bit messages in the split-state model.

2012 ACM Subject Classification Theory of computation \rightarrow Cryptographic primitives; Mathematics of computing \rightarrow Spectra of graphs

Keywords and phrases Non-Malleable Code, Expander Graph, Mixing Lemma

Digital Object Identifier 10.4230/LIPIcs.ITC.2020.6

Funding *Peter Michael Reichstein Rasmussen*: Supported in part by grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.

Amit Sahai: Supported in part from a DARPA/ARL SAFEWARE award, NSF Frontier Award 1413955, and NSF grant 1619348, BSF grant 2012378, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through the ARL under Contract W911NF-15-C- 0205. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

Acknowledgements A significant effort was made to simplify our proof as much as possible, which eventually resulted in the approximately 2-page proof of our main result presented here; we thank Anders Aamand and Jakob Bæk Tejs Knudsen for suggestions and insights regarding the main theorem that helped simplify and improve the results presented. Furthermore, we thank Aayush Jain, Yuval Ishai, and Dakshita Khurana for early discussions regarding simple constructions of split-state non-malleable codes not based on expander graphs.

1 Introduction

A key goal in theoretical computer science is the identification of structures that exhibit resilience to adversarial tampering. The classical notion in this space is that of an error-detection or error-correction code, where we seek to ensure that tampering caused by an adversary that can modify a *bounded* number of symbols in a codeword can be detected or corrected.

But what if the number of errors that an adversary can introduce is unbounded? The objective of error detection or correction is clearly impossible to achieve in this setting – the adversary can simply replace the transmitted codeword with an encoding of some other fixed value. Thus, the main question of study in this context concerns the notion of *malleability*: informally speaking, our core goal must be to prevent the adversary from replacing an encoding of a value x with an encoding of some other related value $\tilde{x} \neq x$.

The central information-theoretic object in this setting is called a split-state non-malleable code [5]. Since their introduction in 2010 [5], split-state non-malleable codes have been the subject of intense study within theoretical computer science [5, 4, 1, 3, 2, 6]. Here,



© Peter Michael Reichstein Rasmussen and Amit Sahai;
licensed under Creative Commons License CC-BY

1st Conference on Information-Theoretic Cryptography (ITC 2020).

Editors: Yael Tauman Kalai, Adam D. Smith, and Daniel Wichs; Article No. 6; pp. 6:1–6:10

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

we consider the most basic form of a split-state non-malleable code, namely a code for encoding a single bit. A split-state non-malleable code [5] for single-bit messages consists of randomized encoding and decoding algorithms (enc, dec). A message $m \in \{0, 1\}$ is encoded as a pair of strings $(L, R) \in \{0, 1\}^k \times \{0, 1\}^k$, such that $\text{dec}(L, R) = m$. An adversary then specifies an arbitrary pair of functions $g, h : \{0, 1\}^k \rightarrow \{0, 1\}^k$. The code is said to be non-malleable if, intuitively, the message obtained as $\text{dec}(g(L), h(R))$ is “unrelated” to the original message m . In particular, to be ε -non-malleable, it is enough [4] to guarantee that when the message m is chosen uniformly at random and encoded into (L, R) , the probability that $\text{dec}(g(L), h(R)) = 1 - m$ is at most $\frac{1}{2} + \varepsilon$.

1.1 Previous Work

All known constructions and proofs of security for explicit split-state non-malleable codes have required complex mathematical proofs, and all known such proofs either directly or indirectly used the mathematics behind constructions of two-source extractors [4, 1, 3, 2, 6]. In fact, after constructing the first non-malleable code in the split-state model Dziembowski, Kazana, and Obremski wrote: “This brings a natural question if we could show some relationship between the extractors and the non-malleable codes in the split-state model. Unfortunately, there is no obvious way of formalizing the conjecture that non-malleable codes need to be based on extractors” [4].

1.2 Our Contribution

In this work, we seek to establish new, simpler, foundations for the construction of single-bit split-state non-malleable codes. We do so by answering in the negative the implicit conjecture of [4]; we show that it is not necessary to base constructions of non-malleable codes on the theory of extractors.

Specifically, we show that expander graphs immediately give rise to split-state non-malleable codes for single-bit messages. We prove that any d -regular graph on $n = 2^k$ nodes with spectral expansion λ satisfying $n = \Omega(d^3 \log(d)/\lambda)$ yields a $O\left(\frac{\lambda^{3/2}}{d}\right)$ -non-malleable code for single-bit messages in the split-state model. Our proof is elementary, requiring a little more than two pages to prove, having at its heart two nested applications of the Expander Mixing Lemma. Furthermore, we only need expanders of high degree (e.g., $d = n^{1/3}$), which can be constructed and analyzed easily (see, e.g., [7] or Appendix C), yielding $2^{-\Omega(k)}$ -non-malleable codes. It is worth noting that the manner in which we construct a single-bit code from an expander graphs is similar to how [4] constructs a single-bit code from a two-source extractor. Thus, our main discovery is that expander graphs suffice for such a construction to succeed.

Our construction of non-malleable codes from expander graphs thus opens up a new line of attack in the study of split-state non-malleable codes. It is important to keep in mind that current constructions of non-malleable codes supporting messages of arbitrary length use many ideas pioneered in the construction of [4], in particular the use of extractors. While we do not yet know how to generalize our results beyond single-bit messages, we speculate that further investigation building upon our work will reveal a deeper connection and more powerful simple constructions based on expanders.

It should be noted that two-source extractors are well-known to exhibit expansion properties; however, in all previous proofs, much more than mere expansion was used to argue non-malleability. Indeed previous proofs apply extractors repeatedly; for instance the proof of [4] uses the extractor property multiple times (e.g., in equation (22) and using

equation (43) in [4]). We also note that it is not surprising that 1-bit non-malleable codes will exhibit some sort of expansion properties. Our contribution is the converse: that good expansion is *sufficient* for the construction of non-malleable codes.

1.3 Parameters and a Comparison with DKO13

For completeness we include an analysis of the concrete parameters of our resulting code. Let $\gamma > 0$ be given. Our construction yields a 1-bit γ -non-malleable split-state code where each part of the message is a vertex of a d -regular graph G on n vertices. The graph G must have expansion λ and satisfy $n = \Omega(d^3 \log(d)/\lambda)$ and $\lambda^{3/2}/d = O(\gamma)$. Suppose that G has expansion $\Theta(\sqrt{d})$, which is the case for the instantiation in Appendix C. We may then set $d = \Theta((1/\gamma)^4)$ and $n = \tilde{\Theta}((1/\gamma)^{10})$. Thus, our code uses space $20 \log(1/\gamma) + O(\log \log(1/\gamma))$ to encode a single bit. The instantiation from Appendix C is not able to choose n as flexibly as suggested here and uses space $24 \log(1/\gamma)$. The time taken to encode and decode a message in this instantiation is $O(\log(1/\gamma))$. In comparison, the instantiation of [4] uses space around $90 \log(1/\gamma)$ and the time to encode and decode is $O(\log(1/\gamma) \log^2(\log(1/\gamma)))$. It should be noted, however, that the construction of [4] supports leakage as well, something that we are not considering in this paper.

1.4 Intuition behind our construction and analysis

Every graph, $G = (V, E)$ yields a single-bit split-state code in the following straightforward manner: To encode 1, pick an edge $(v, u) \in E$ uniformly at random, and set the left encoding to be v and the right encoding to be u . To encode a 0, do the same with a uniformly random non-edge in the graph.

Our analysis proceeds in two parts. First, in Proposition 6, using only elementary manipulations, we give an exact characterization of the success probability of any particular tampering split-state adversary against the code associated with any graph. The split-state adversary uses two functions, g and h , to tamper with the left and right encodings, respectively. The significant term of the probability to be analyzed is the quantity

$$\sum_{(v,u) \in E} \left(\frac{d |g^{-1}(v)| \cdot |h^{-1}(u)|}{n} - |E(g^{-1}(v), h^{-1}(u))| \right).$$

To bound this expression, we make the following observations. First, sparsity of the graph allows us to bound many of the terms immediately. Second, the term in the parentheses above immediately suggests a bound using the Expander Mixing Lemma, applied to the number of edges from $g^{-1}(v)$ to $h^{-1}(u)$. Third, we observe that the sum itself is over edges $(v, u) \in E$, and furthermore, the remaining sum of problematic terms are a sum over edges of the form $(v, u) \in E \cap (T \times S)$ for some vertex subsets $S, T \subset V$. This allows us to apply the Expander Mixing Lemma a *second* time, effectively bounding the number of “error terms” that accumulate through the initial use of the Expander Mixing Lemma. The actual analysis of this bound is just over a page of calculation. The analysis follows the intuition above, modulo a partitioning of terms into sets of appropriate size for the analysis to work.

2 Preliminaries

We shall assume familiarity with the basics of codes and non-malleable codes. A cursory review of relevant definitions can be found in the appendix.

6:4 Expander Graphs Are Non-Malleable Codes

► **Notation 1** (Graphs). A graph $G = (V, E)$ consists of vertices V and edges $E \subset V \times V$. In this exposition every graph is undirected and $n = |V|$ always denotes the number of vertices of the graph in question.

- For any $v \in V$ we denote by $N(v)$ the set of neighbors of v in G .
- For any two subsets $S, T \subseteq V$ we denote by $E(S, T)$ the set of (directed) edges from S to T in G . I.e. $E(S, T) = \{(v, u) \in S \times T \mid (v, u) \in E\}$.

► **Definition 2** (Spectral Expander). Let $G = (V, E)$ be a d -regular graph, A_G be its adjacency matrix, and $\lambda_1 \geq \dots \geq \lambda_n$ be the eigenvalues of A_G . We say that G is a λ spectral expander if $\lambda \geq \max\{|\lambda_2|, \dots, |\lambda_n|\}$.

► **Theorem 3** (Expander Mixing Lemma). Suppose that $G = (V, E)$ is a λ spectral expander. Then for every pair of subsets $S, T \subset V$ we have

$$\left| |E(S, T)| - \frac{d \cdot |S| \cdot |T|}{n} \right| \leq \lambda \sqrt{|S| \cdot |T|}.$$

Our results will rely on the following characterization of 1-bit non-malleable codes by Dziembowski, Kazana, and Obremski found in [4].

► **Theorem 4.** Let (enc, dec) be a coding scheme with $\text{enc}: \{0, 1\} \rightarrow \mathcal{X}$ and $\text{dec}: \mathcal{X} \rightarrow \{0, 1\}$. Further, let \mathcal{F} be a set of functions $f: \mathcal{X} \rightarrow \mathcal{X}$. Then (enc, dec) is ε -non-malleable with respect to \mathcal{F} if and only if for every $f \in \mathcal{F}$,

$$\Pr_{b \leftarrow \{0, 1\}} (\text{dec}(f(\text{enc}(b))) = 1 - b) \leq \frac{1}{2} + \varepsilon,$$

where the probability is over the uniform choice of b and the randomness of enc .

3 Results

We first formally introduce our candidate code and then prove that it is a non-malleable code.

3.1 Candidate Code

From a graph we can very naturally construct a coding scheme as follows.

► **Definition 5** (Graph Code). Let $G = (V, E)$ be a graph. The associated graph code, $(\text{enc}_G, \text{dec}_G)$, consists of the functions

$$\text{enc}_G: \{0, 1\} \rightarrow V \times V, \quad \text{dec}_G: V \times V \rightarrow \{0, 1\}$$

which are randomized and deterministic, respectively, and given by

$$\text{enc}_G(b) = \begin{cases} (u, v) \leftarrow (V \times V) \setminus E, & b = 0, \\ (u, v) \leftarrow E, & b = 1, \end{cases}$$

$$\text{dec}_G(v_1, v_2) = \begin{cases} 0, & (v_1, v_2) \notin E, \\ 1, & (v_1, v_2) \in E. \end{cases}$$

3.2 Non-Malleability of Expander Graph Codes

Finally, arriving at the core of the matter, we first establish the following lemma casting the expression of Theorem 4 in terms of graph properties.

► **Proposition 6.** *Let $G = (V, E)$ be a graph, functions $g, h: V \rightarrow V$ be given, and $f = (g, h): V \times V \rightarrow V \times V$ satisfy $f(u, v) = (g(u), h(v))$. For the probability that f flips a random bit encoded by enc_G , write*

$$T = \Pr_{b \stackrel{u}{\leftarrow} \{0,1\}} (\text{dec}_G(f(\text{enc}_G(b))) = 1 - b)$$

where the probability is taken over the randomness of enc_G and the sampling of b . Then

$$T = \frac{1}{2} + \frac{1}{2d(n-d)} \sum_{(v,u) \in E} \left(\frac{d|g^{-1}(v)| \cdot |h^{-1}(u)|}{n} - |E(g^{-1}(v), h^{-1}(u))| \right). \quad (1)$$

Proof. For $b \in \{0, 1\}$ denote by Q_b the probability

$$Q_b = \Pr(\text{dec}_G(f(\text{enc}_G(b))) = 1 - b)$$

taken over the randomness of enc_G . It is clear that $T = \frac{Q_0 + Q_1}{2}$ and that by definition

$$Q_0 = \Pr_{(v,u) \stackrel{u}{\leftarrow} V \times V \setminus E} [(g(v), h(u)) \in E], \quad Q_1 = \Pr_{(v,u) \stackrel{u}{\leftarrow} E} [(g(v), h(u)) \notin E].$$

First, for $b = 0$ we see that the number of non-edges that are mapped by f to any given $(v, u) \in E$ is given by $|g^{-1}(v)| \cdot |h^{-1}(u)| - |E(g^{-1}(v), h^{-1}(u))|$. There are $n(n-d)$ non-edges in G so it follows that

$$Q_0 = \frac{\sum_{(v,u) \in E} |g^{-1}(v)| \cdot |h^{-1}(u)| - |E(g^{-1}(v), h^{-1}(u))|}{n(n-d)}.$$

Second, for $b = 1$ the number of edges of G that are mapped to non-edges by f is given by $\sum_{(v,u) \notin E} |E(g^{-1}(v), h^{-1}(u))|$. Since there are dn edges of G to choose from when encoding the bit $b = 1$,

$$Q_1 = \frac{\sum_{(v,u) \notin E} |E(g^{-1}(v), h^{-1}(u))|}{dn}.$$

Now, observing that the number of (directed) edges in the graph is dn and that $\{g^{-1}(v)\}_{v \in V}$ and $\{h^{-1}(u)\}_{u \in V}$ are both partitions of V , we get

$$Q_1 = \frac{dn - \sum_{(v,u) \in E} |E(g^{-1}(v), h^{-1}(u))|}{dn} = 1 - \frac{\sum_{(v,u) \in E} |E(g^{-1}(v), h^{-1}(u))|}{dn}.$$

Putting it all together,

$$\begin{aligned} T &= \frac{\sum_{(v,u) \in E} |g^{-1}(v)| \cdot |h^{-1}(u)| - |E(g^{-1}(v), h^{-1}(u))|}{2n(n-d)} + \frac{1}{2} - \frac{\sum_{(v,u) \in E} |E(g^{-1}(v), h^{-1}(u))|}{2dn} \\ &= \frac{1}{2} + \frac{1}{2d(n-d)} \sum_{(v,u) \in E} \left(\frac{d|g^{-1}(v)| \cdot |h^{-1}(u)|}{n} - |E(g^{-1}(v), h^{-1}(u))| \right). \quad \blacktriangleleft \end{aligned}$$

We proceed immediately with the main theorem, which concludes the exposition. In order to keep this presentation short and to the point, more elaborate calculations, which avoid the log-factors, have been placed in the appendix as Theorem 10.

6:6 Expander Graphs Are Non-Malleable Codes

► **Theorem 7.** *Let $G = (V, E)$ be d -regular with spectral expansion λ satisfying $n = \Omega(d^3 \log(d)^4 / \lambda)$. Then $(\text{enc}_G, \text{dec}_G)$ is an $\tilde{O}\left(\frac{\lambda^{3/2}}{d}\right)$ -non-malleable code in the split-state model.*

Proof. Let $f = (g, h): V \times V \rightarrow V \times V$ be given. By Theorem 4 and Proposition 6 we just need to show that

$$R = \frac{1}{2d(n-d)} \cdot \sum_{(v,u) \in E} \left(\frac{d|g^{-1}(v)| \cdot |h^{-1}(u)|}{n} - |E(g^{-1}(v), h^{-1}(u))| \right)$$

is bounded by $\tilde{O}\left(\frac{\lambda^{3/2}}{d}\right)$. Define the sets

$$\begin{aligned} G_1 &= \left\{ v \in V \mid |g^{-1}(v)| > \frac{n}{d^2} \right\}, & H_1 &= \left\{ u \in V \mid |h^{-1}(u)| > \frac{n}{d^2} \right\}, \\ G_2 &= \left\{ v \in V \mid |g^{-1}(v)| \leq \frac{n}{d^2} \right\}, & H_2 &= \left\{ u \in V \mid |h^{-1}(u)| \leq \frac{n}{d^2} \right\}, \end{aligned}$$

for $i, j \in \{1, 2\}$ write

$$R_{i,j} = \frac{1}{2d(n-d)} \sum_{(v,u) \in E \cap (G_i \times H_j)} \left(\frac{d|g^{-1}(v)| \cdot |h^{-1}(u)|}{n} - |E(g^{-1}(v), h^{-1}(u))| \right),$$

and observe that $R = \sum_{1 \leq i, j \leq 2} R_{i,j}$.

Consider the case when $i = 2$. Simply bounding the terms of the form $|g^{-1}(v)| \cdot |h^{-1}(u)|$ by using that each vertex has only d neighbours, we get

$$\begin{aligned} R_{2,1} + R_{2,2} &\leq \frac{1}{2n(n-d)} \sum_{(v,u) \in E \cap (G_2 \times V)} |g^{-1}(v)| \cdot |h^{-1}(u)| \\ &\leq \frac{1}{2n(n-d)} \cdot d \cdot \sum_{u \in V} \frac{n}{d^2} \cdot |h^{-1}(u)| \\ &= \frac{n}{2(n-d)d}. \end{aligned}$$

Thus, $R_{2,1} + R_{2,2} = O(d^{-1})$. By symmetry, $R_{1,2} = O(d^{-1})$. It only remains to show that $R_{1,1} = \tilde{O}\left(\frac{\lambda^{3/2}}{d}\right)$. To this end, partition G_1 and H_1 , respectively, as

$$G_1^k = \left\{ v \in G_1 \mid \frac{n}{2^{k-1}} \geq |g^{-1}(v)| > \frac{n}{2^k} \right\}, \quad H_1^l = \left\{ u \in H_1 \mid \frac{n}{2^{l-1}} \geq |h^{-1}(u)| > \frac{n}{2^l} \right\}$$

for $1 \leq k, l \leq \lceil \log_2(d^2) \rceil$. Now, focusing on each pair G_1^k and H_1^l , we write

$$S_{k,l} = \frac{1}{2d(n-d)} \sum_{(v,u) \in E \cap (G_1^k \times H_1^l)} \left(\frac{d|g^{-1}(v)| \cdot |h^{-1}(u)|}{n} - |E(g^{-1}(v), h^{-1}(u))| \right)$$

and apply first the mixing lemma then the Cauchy-Schwartz inequality to get

$$\begin{aligned} 2d(n-d)S_{k,l} &= \sum_{v \in G_1^k} \left(\frac{d|g^{-1}(v)| \cdot \sum_{u \in N(v) \cap H_1^l} |h^{-1}(u)|}{n} - \left| E \left(g^{-1}(v), \bigcup_{u \in N(v) \cap H_1^l} h^{-1}(u) \right) \right| \right) \\ &\leq \sum_{v \in G_1^k} \lambda \sqrt{|g^{-1}(v)| \cdot \sum_{u \in N(v) \cap H_1^l} |h^{-1}(u)|} \\ &\leq \lambda \sqrt{\frac{n}{2^{k-1}} \cdot \frac{n}{2^{l-1}}} \cdot \sum_{v \in G_1^k} \sqrt{|N(v) \cap H_1^l|} \\ &\leq 2\lambda n \cdot 2^{-\frac{l+k}{2}} \cdot \sqrt{|G_1^k|} \cdot \sqrt{|E(G_1^k, H_1^l)|}. \end{aligned}$$

We use the fact that $|G_1^k| \leq 2^k$, $|H_1^l| \leq 2^l$, apply the mixing lemma to the last factor, and wield Jensen's inequality on the arising square root to obtain

$$\begin{aligned} d(n-d)S_{k,l} &\leq \lambda n \cdot 2^{-\frac{l+k}{2}} \cdot \sqrt{|G_1^k|} \cdot \sqrt{\frac{d \cdot |G_1^k| \cdot |H_1^l|}{n} + \lambda \sqrt{|G_1^k| \cdot |H_1^l|}} \\ &\leq \lambda \sqrt{2^k d n} + 2^{\frac{k-l}{4}} \lambda^{3/2} n \leq \lambda \cdot \sqrt{d^3 n} + 2^{\frac{k-l}{4}} \lambda^{3/2} n. \end{aligned}$$

By symmetry of k and l , $d(n-d)S_{k,l} \leq \lambda \cdot \sqrt{d^3 n} + 2^{\frac{l-k}{4}} \lambda^{3/2} n$. Thus,

$$\begin{aligned} R_{1,1} &= \sum_{1 \leq k, l \leq \lceil \log_2(d^2) \rceil} S_{k,l} \\ &\leq O\left(\frac{\lambda \log(d)^2 \cdot \sqrt{d}}{\sqrt{n}}\right) + O\left(\frac{\lambda^{3/2}}{d}\right) \cdot \sum_{1 \leq k, l \leq \lceil \log_2(d^2) \rceil} 2^{-\frac{|k-l|}{4}} \\ &= O\left(\frac{\log(d) \lambda^{3/2}}{d}\right). \end{aligned} \quad \blacktriangleleft$$

References

- 1 Divesh Aggarwal, Yevgeniy Dodis, and Shachar Lovett. Non-malleable codes from additive combinatorics. In *Symposium on Theory of Computing, STOC*, 2014.
- 2 Eshan Chattopadhyay, Vipul Goyal, and Xin Li. Non-malleable extractors and codes, with their many tampered extensions. In *Symposium on Theory of Computing, STOC*, 2016.
- 3 Eshan Chattopadhyay and David Zuckerman. Non-malleable codes against constant split-state tampering. In *Foundations of Computer Science, FOCS*, 2014.
- 4 Stefan Dziembowski, Tomasz Kazana, and Maciej Obremski. Non-malleable codes from two-source extractors. In *CRYPTO*, 2013.
- 5 Stefan Dziembowski, Krzysztof Pietrzak, and Daniel Wichs. Non-malleable codes. In *ICS*, 2010.
- 6 Xin Li. Improved non-malleable extractors, non-malleable codes and independent source extractors. In *Symposium on Theory of Computing, STOC*, 2017.
- 7 Luca Trevisan. Luca trevisan's 'in theory' blog. <https://lucatrevisan.wordpress.com/2011/02/28/cs359g-lecture-16-constructions-of-expanders/>. Accessed: 2018-09-27.

A Definitions for Split-State Non-Malleable Codes

Here, we recall the basic definition of a split-state non-malleable code due to [5].

► **Definition 8** (Coding scheme). *We define a coding scheme to be a pair of functions (enc, dec) . The encoding function $\text{enc}: \mathcal{M} \rightarrow \mathcal{X}$ is randomized while the decoding function $\text{dec}: \mathcal{X} \rightarrow \mathcal{M} \cup \{\perp\}$ is deterministic. Further, for all $s \in \mathcal{M}$ the pair satisfies*

$$\Pr[\text{dec}(\text{enc}(s)) = s] = 1$$

where the probability is taken over the randomness of enc .

► **Definition 9** (Split State Non-Malleable Code). *A coding scheme (enc, dec) , $\text{enc}: \mathcal{M} \rightarrow \mathcal{L} \times \mathcal{R}$ and $\text{dec}: \mathcal{L} \times \mathcal{R} \rightarrow \mathcal{M} \cup \{\perp\}$, is ε -non-malleable in the split state model if for every pair of functions $g: \mathcal{L} \rightarrow \mathcal{L}, h: \mathcal{R} \rightarrow \mathcal{R}$ and writing $f = (g, h)$ there exists a distribution D_f*

6:8 Expander Graphs Are Non-Malleable Codes

supported on $\mathcal{M} \cup \{*, \perp\}$ such that for every $s \in \mathcal{M}$ the two random variables defined by the experiments

$$A_f^s = \left\{ \begin{array}{l} (L,R) \leftarrow \text{enc}(s); \\ \text{Output } \text{dec}(g(L), h(R)) \end{array} \right\}$$

$$B_f^s = \left\{ \begin{array}{l} \tilde{s} \leftarrow D_f; \\ \text{If } \tilde{s} = * \text{ output } s \text{ else output } \tilde{s} \end{array} \right\}$$

have statistical distance at most ε .

B Deliver Us from Log Factors

A more thorough analysis of the sums in the proof of Theorem 7 allows us to get slightly better bounds. The technicalities are of little interest to the big picture and were hence omitted in the body of the paper. The addition consists of an alternative ending to the proof of Theorem 7.

► **Theorem 10.** *Let $G = (V, E)$ be d -regular with spectral expansion λ satisfying $n = \Omega(d^3 \log(d)/\lambda)$. Then $(\text{enc}_G, \text{dec}_G)$ is an $O\left(\frac{\lambda^{3/2}}{d}\right)$ -non-malleable code in the split-state model.*

Proof. At the very end of the proof of Theorem 7, we arrived at

$$d(n-d)S_{k,l} \leq 2^{-\frac{l+k}{2}} \lambda n \cdot \sqrt{|G_1^k|} \cdot \sqrt{\frac{d \cdot |G_1^k| \cdot |H_1^l|}{n}} + \lambda \cdot \sqrt{|G_1^k| \cdot |H_1^l|}.$$

Applying Jensen's inequality, we get

$$S_{k,l} \leq O\left(\frac{\lambda}{\sqrt{dn}}\right) \cdot 2^{-\frac{l+k}{2}} \cdot |G_1^k| \cdot \sqrt{|H_1^l|} + O\left(\frac{\lambda^{3/2}}{d}\right) \cdot 2^{-\frac{l+k}{2}} \cdot \sqrt[4]{|G_1^k|^3 \cdot |H_1^l|} \quad (2)$$

with the functions hidden by the O -notation being independent of k, l .

Now, note that

$$|g^{-1}(G_1^k)| \geq \frac{n \cdot |G_1^k|}{2^k} \qquad |h^{-1}(H_1^l)| \geq \frac{n \cdot |H_1^l|}{2^l} \quad (3)$$

and for all $k \leq \lceil \log_2(d^2) \rceil$ we have $\frac{|G_1^k|}{2^{k/2}} \leq 2d$. We shall bound each of the terms of (2) separately.

First, write

$$L = \sum_{1 \leq k, l \leq \lceil \log_2(d^2) \rceil} \left(2^{-\frac{l+k}{2}} \cdot |G_1^k| \cdot \sqrt{|H_1^l|} \right).$$

Using the Cauchy-Schwartz inequality in the second inequality,

$$\begin{aligned} L &\leq 2d \cdot \sum_{1 \leq l \leq \lceil \log_2(d^2) \rceil} \sqrt{2^{-l} |H_1^l|} \\ &\leq O\left(d \cdot \sqrt{\log(d)}\right) \cdot \sqrt{\sum_{1 \leq l \leq \lceil \log_2(d^2) \rceil} 2^{-l} \cdot |H_1^l|} \\ &\leq O\left(d \cdot \sqrt{\log(d)}\right) \cdot \sqrt{\sum_{1 \leq l \leq \lceil \log_2(d^2) \rceil} \frac{|h^{-1}(H_1^l)|}{n}} \\ &= O\left(d \cdot \sqrt{\log(d)}\right) \end{aligned}$$

since the H_1^l are disjoint subsets of V . In conclusion,

$$\begin{aligned} O\left(\frac{\lambda}{\sqrt{dn}}\right) \cdot \sum_{1 \leq k, l \leq \lceil \log_2(d^2) \rceil} 2^{-\frac{l+k}{2}} \cdot |G_1^k| \cdot \sqrt{|H_1^l|} &= O\left(\frac{\lambda \cdot \sqrt{d \cdot \log(d)}}{\sqrt{n}}\right) \\ &= O\left(\frac{\lambda^{3/2}}{d}\right). \end{aligned}$$

Second, let $k \leq l$ and write $t = l - k$. We now bound the sum using (3). Write

$$K = \sum_{1 \leq k < l \leq \lceil \log_2(d^2) \rceil} 2^{-\frac{l+k}{2}} \cdot \sqrt[4]{|G_1^k|^3 \cdot |H_1^l|}.$$

Then

$$\begin{aligned} K &\leq \sum_{1 \leq k < l \leq \lceil \log_2(d^2) \rceil} \left(\frac{2^{\frac{k-l}{4}}}{n} \cdot \sqrt[4]{|g^{-1}(G_1^k)|^3 \cdot |h^{-1}(H_1^l)|} \right) \\ &\leq \sum_{t=0}^{\lceil \log_2(d^2) \rceil} \left(\frac{2^{-\frac{t}{4}}}{n} \sum_{l=t}^{\lceil \log_2(d^2) \rceil} \sqrt[4]{|g^{-1}(G_1^{l-t})|^3 \cdot |h^{-1}(H_1^l)|} \right) \\ &\leq \sum_{t=0}^{\lceil \log_2(d^2) \rceil} \left(\frac{2^{-\frac{t}{4}}}{n} \left(\sum_{l=t}^{\lceil \log_2(d^2) \rceil} |g^{-1}(G_1^{l-t})| \right)^{3/4} \cdot \left(\sum_{l=t}^{\lceil \log_2(d^2) \rceil} |h^{-1}(H_1^l)| \right)^{1/4} \right) \\ &\leq \sum_{t=0}^{\lceil \log_2(d^2) \rceil} 2^{-\frac{t}{4}} = O(1), \end{aligned}$$

where the third inequality is established using Hölder's inequality.

It now follows that

$$\sum_{1 \leq k \leq l \leq \lceil \log_2(d^2) \rceil} S_{k,l} = O\left(\frac{\lambda^{3/2}}{d}\right).$$

By symmetry of k and l ,

$$R_{1,1} = \sum_{1 \leq k, l \leq \lceil \log_2(d^2) \rceil} S_{k,l} = O\left(\frac{\lambda^{3/2}}{d}\right),$$

which completes the proof. ◀

C Instantiating Our Construction

Using our results to instantiate an efficient, secure split-state non-malleable code, we require a family of graphs $\{G_k\}_{k \in \mathbb{N}}$, where each $G_k = (V_k, E_k)$ is d_k -regular with spectral expansion λ_k , satisfying the following:

1. The function $\varepsilon(k) = \frac{\lambda_k^{3/2}}{d_k}$ is negligible.
2. We have $n_k = |V(G_k)| = \Omega(d_k^3 \log(d_k)/\lambda_k)$
3. Both sampling an edge $(u, v) \xleftarrow{u} E_k$ and sampling a non-edge $(u, v) \xleftarrow{u} (V_k \times V_k) \setminus E_k$ can be done in time polynomial in k .

6:10 Expander Graphs Are Non-Malleable Codes

4. Determining membership of a pair $(u, v) \in V \times V$ in $E(G_k)$ can be done deterministically in time polynomial in k .

Given such a family of graphs it is clear that the corresponding graph code $(\text{enc}_{G_k}, \text{dec}_{G_k})$ is an efficiently computable non-malleable code.

C.1 Instantiation with High-Degree Cayley Graphs

Explicit constructions of such families of graphs do indeed exist. We shall here give an example from [7] from the class of graphs known as Cayley graphs. The construction is as follows.

► **Definition 11.** For p a prime and $1 \leq t < p$ let the graph $\text{LD}_{p,t}$ have vertex set \mathbb{F}_p^{t+1} and edge set

$$E(\text{LD}_{p,t}) = \{(x, x + (b, ab, a^2b, \dots, a^tb)) \mid x \in \mathbb{F}_p^{t+1}, a, b \in \mathbb{F}_p\},$$

i.e. $x, y \in V(\text{LD}_{p,t})$ are connected by an edge if and only if there exists $a, b \in \mathbb{F}_p$ such that $y = x + (b, ab, a^2b, \dots, a^tb)$.

It is worth noting that the graph $\text{LD}_{p,t}$ is p^2 -regular and that it is undirected as x is connected to y if and only if y is connected to x .

Now, let $t = 5$ and for each $k \in \mathbb{N}$ let p_k be some k -bit prime. We consider the family of graphs $\{\text{LD}_{p_k,5}\}_{k \in \mathbb{N}}$ for our instantiation. In the following, we shall check the criteria from the beginning of the section point by point.

1. The family of graphs $\text{LD}_{p,t}$ has great expander properties.

► **Theorem 12** (explicit in Trevisan [7]). For $1 < t < p$, the graph $\text{LD}_{p,t}$ is a pt -spectral expander.

This fact allows us to note that for our particular choice of graphs, $\varepsilon(k) = \frac{(p_k t)^{3/2}}{p_k^2} < \frac{12}{\sqrt{p_k}}$, which in fact is $2^{-\Omega(k)}$ and the representation size is $O(k)$ bits.

2. We have $\Omega\left(\frac{d_k^3 \log(d_k)}{\lambda_k}\right) = \Omega(p^5 \log(p))$ such that indeed,

$$n_k = |V(\text{LD}_{p_k,5})| = p^6 = \Omega\left(\frac{d_k^3 \log(d_k)}{\lambda_k}\right).$$

3. Sampling an edge $(u, v) \stackrel{u}{\leftarrow} E(\text{LD}_{p_k,t})$ is simply a question of picking $x \in \mathbb{F}_{p_k}^{t+1}$, $a, b \in \mathbb{F}_{p_k}$ uniformly at random and then outputting the edge $(x, x + (b, ab, a^2b, \dots, a^tb))$.

To pick a non-edge, simply sample two random vertices $u, v \in \mathbb{F}_{p_k}^{t+1}$ uniformly at random and check (with the procedure to be specified below) whether $(u, v) \in E(\text{LD}_{p_k,t})$. Since for $t > 1$ the probability of hitting an edge with such a random choice is $\leq 1/p_k$, the expected number of repetitions is constant and hence the procedure takes expected polynomial time.

4. To test membership of some $(u, v) \in (\mathbb{F}_{p_k}^{t+1})^2$ in $E(\text{LD}_{p_k,t})$, perform the following operation: Compute $x = u - v$ and write $x = (x_0, \dots, x_t)$. It is now trivial to check whether $(1, \frac{x_1}{x_0}, \dots, \frac{x_t}{x_0})$ is of the form $(1, a, a^2, \dots, a^t)$.

Appendix D

Support of Closed Walks and Second Eigenvalue Multiplicity of Graphs

Support of Closed Walks and Second Eigenvalue Multiplicity of the Normalized Adjacency Matrix

Theo McKenzie*
UC Berkeley

Peter M. R. Rasmussen†
University of Copenhagen

Nikhil Srivastava‡
UC Berkeley

July 9, 2021

Abstract

We show that the multiplicity of the second normalized adjacency matrix eigenvalue of any connected graph of maximum degree Δ is bounded by $O(n\Delta^{7/5}/\log^{1/5-o(1)} n)$ for any Δ , and by $O(n \log^{1/2} d / \log^{1/4-o(1)} n)$ for simple d -regular graphs when $d \geq \log^{1/4} n$. In fact, the same bounds hold for the number of eigenvalues in any interval of width $\lambda_2 / \log_{\Delta}^{1-o(1)} n$ containing the second eigenvalue λ_2 . The main ingredient in the proof is a polynomial (in k) lower bound on the typical support of a closed random walk of length $2k$ in any connected graph, which in turn relies on new lower bounds for the entries of the Perron eigenvector of submatrices of the normalized adjacency matrix.

Contents

1	Introduction	2
1.1	Higher degree regular graphs	4
1.2	Related work	5
1.3	Notation	6
2	Lower Bounds on the Perron Eigenvector	7
3	Support of Closed Walks	10
4	Bound on Eigenvalue Multiplicity	13
5	Examples	15
5.1	Bipartite Ramanujan Graphs	15
5.2	Mangrove Tree	16
6	Open Problems	18

*mckenzie@math.berkeley.edu. Supported by NSF Grant DGE-1752814.

†pmrr@di.ku.dk. Supported in part by grant 16582, Basic Algorithms Research Copenhagen (BARC), from the VILLUM Foundation.

‡nikhil@math.berkeley.edu. Supported by NSF Grants CCF-1553751 and CCF-2009011.

1 Introduction

The eigenvalues of matrices associated with graphs play an important role in many areas of mathematics and computer science, so general phenomena concerning them are of broad interest. In their recent beautiful work on the equiangular lines problem, Jiang, Tidor, Yao, Zhang, and Zhao [JTY⁺19] proved the following novel result constraining the distribution of the adjacency eigenvalues of *all* connected graphs of sufficiently low degree.

Theorem 1.1. *If G is a connected graph of maximum degree Δ on n vertices, then the multiplicity of the second largest eigenvalue of its adjacency matrix A_G is bounded by $O(n \log \Delta / \log \log(n))$.*

For their application to equiangular lines, [JTY⁺19] only needed to show that the multiplicity of the second eigenvalue is $o(n)$, but they asked whether the $O(n / \log \log(n))$ dependence in Theorem 1.1 could be improved, noting a huge gap between this and the best known lower bound of $\Omega(n^{1/3})$ achieved by certain Cayley graphs of $\text{PSL}(2, p)$ (see [JTY⁺19, Section 4]). Apart from Theorem 1.1, there are as far as we are aware no known sublinear upper bounds on the second eigenvalue multiplicity for any general class of graphs, even if the question is restricted to Cayley graphs (unless one imposes a restriction on the spectral gap; see Section 1.2 for a discussion).

Meanwhile, in the theoretical computer science community, the largest eigenvalues of the *normalized* adjacency matrix $\tilde{A}_G := D_G^{-1/2} A_G D_G^{-1/2}$ (for D_G the diagonal matrix of degrees) have received much attention over the past decade due to their relation with graph partitioning problems and the unique games conjecture (see e.g. [Kol11, BRS11, LRTV12, OGT13, LOGT14, ABS15, BGH⁺15, LOG18]); in particular, many algorithmic tasks become easier on graphs with few large normalized adjacency eigenvalues. Thus, it is of interest to know how many of these eigenvalues there can be in the worst case.

In this work, we prove significantly stronger upper bounds than Theorem 1.1 on the second eigenvalue multiplicity for the normalized adjacency matrix. Graphs are undirected and allowed to have multiedges and self-loops, unless specified to be simple. Order the eigenvalues of \tilde{A}_G as $\lambda_1(\tilde{A}_G) \geq \lambda_2(\tilde{A}_G) \geq \dots \geq \lambda_n(\tilde{A}_G)$, and let $m_G(I)$ denote the number of eigenvalues of \tilde{A}_G in an interval I .

Theorem 1.2. *If G is a connected graph of maximum degree Δ on n vertices with $\lambda_2(\tilde{A}_G) = \lambda_2$, then¹*

$$m_G\left(\left[\left(1 - \frac{\log \log_{\Delta} n}{\log_{\Delta} n}\right)\lambda_2, \lambda_2\right]\right) = \tilde{O}\left(n \cdot \frac{\Delta^{7/5}}{\log^{1/5} n}\right). \quad (1)$$

Because of the relationship $\tilde{A}_G = \frac{1}{d} A_G$ when G is regular, (1) gives a substantial improvement on Theorem 1.1 in the regular case (in the non-regular case, the results are incomparable as they concern different matrices). In addition to the stronger $O(n/\text{polylog}(n))$ bound, a notable difference between our result and Theorem 1.1 is that we control the number of eigenvalues in a small interval containing λ_2 . Though we do not know whether the exponents in (1) are sharp, we show in Section

¹All asymptotics are as $n \rightarrow \infty$ and the notation $\tilde{O}(\cdot)$ suppresses polyloglog(n) terms.

5.1 that constant degree bipartite Ramanujan graphs have at least $\Omega(n/\log^{3/2} n)$ eigenvalues in the interval appearing in (1), indicating that $O(n/\text{polylog}(n))$ is the correct regime for the maximum number of eigenvalues in such an interval when Δ is constant.

Theorem 1.2 is nontrivial for all $\Delta = \tilde{o}(\log^{1/7} n)$; as remarked in [JTY⁺19], Paley graphs have degree $\Omega(n)$ and second eigenvalue multiplicity $\Omega(n)$, so some bound on the degree is required to obtain sublinear multiplicity. In Section 1.1, we present a variant of Theorem 1.2 (advertised in the abstract) which yields nontrivial bounds in the special case of simple d -regular graphs with degrees as large as $d = \exp(\log^{1/2-\delta} n)$, which is considerably larger than the regime $d = O(\text{polylog}(n))$ handled by [JTY⁺19].

The main new ingredient in the proof of Theorem 1.2 is a polynomial lower bound on the support of (i.e., number of distinct vertices traversed by) a simple random walk of fixed length conditioned to return to its starting point. The bound holds for any connected graph and any starting vertex and may be of independent interest.

Theorem 1.3. *Suppose G is connected and of maximum degree Δ on n vertices and x is any vertex in G . Let $\gamma_x^{2k} = (x = X_0, X_1, \dots, X_{2k})$ denote a random walk of length $2k < n$ sampled according to the simple random walk on G starting at x . Then*

$$\mathbb{P}(\text{support}(\gamma_x^{2k}) \leq s | X_{2k} = X_0) \leq \exp\left(-\frac{k}{65\Delta^7 s^4}\right) \quad \text{for } s \leq \frac{1}{4} \left(\frac{k}{\Delta^7 \log \Delta}\right)^{1/5}. \quad (2)$$

In particular, this means that for constant Δ , the typical support of a closed random walk of length $2k$ is least $\Omega(k^{1/5})$. It may be tempting to compare Theorem 1.3 with the familiar fact that a random closed walk of length $2k$ on \mathbb{Z} (or in continuous time, a standard Brownian bridge run for time $2k$) attains a maximum distance of $\Omega(\sqrt{k})$ from its origin. However, as seen in Figure 1, there are regular graphs for which a closed walk of length $2k$ from a particular vertex x travels a maximum distance of only $\text{polylog}(k)$ with high probability. Theorem 1.3 reveals that nonetheless the number of *distinct* vertices traversed is always typically $\text{poly}(k)$. We do not know if the specific exponent of $k^{1/5}$ supplied by Theorem 1.3 is sharp, but considering a cycle graph shows that it is not possible to do better than $k^{1/2}$.

Given Theorem 1.3, our proof of Theorem 1.2 follows the strategy of [JTY⁺19]: since most closed walks in G have large support, the number of such walks may be drastically reduced by deleting a small number of vertices from G . By a moment calculation relating the spectrum to self return probabilities and a Cauchy interlacing argument, this implies an upper bound on the multiplicity of $\lambda_2(\tilde{A}_G)$. The crucial difference is that we are able to delete only $n/\text{polylog}(n)$ vertices whereas they delete $n/\text{poly log log}(n)$.

The key ingredient in our proof of Theorem 1.3 is a result regarding the Perron eigenvector (i.e., the unique, strictly positive eigenvector with eigenvalue λ_1) of a submatrix of \tilde{A} .

Theorem 1.4. *For any graph $G = (V, E)$ of maximum degree Δ , take any set of vertices $S \subseteq V$ such that the induced subgraph on S is connected, and let ψ_S be the ℓ_2 -normalized Perron vector of \tilde{A}_S , the principal submatrix of \tilde{A} corresponding to vertices in S . Then there is a vertex $u \in S$ which is adjacent to $V \setminus S$ such that*

$$\psi_S(u) \geq 1/(\Delta^{5/2} \lambda_1(\tilde{A}_S) |S|^{5/2}). \quad (3)$$

When we restrict this result to G being a d -regular graph and pass to the adjacency matrix, we achieve a result about the unnormalized adjacency matrix of irregular graphs that may be of independent interest.

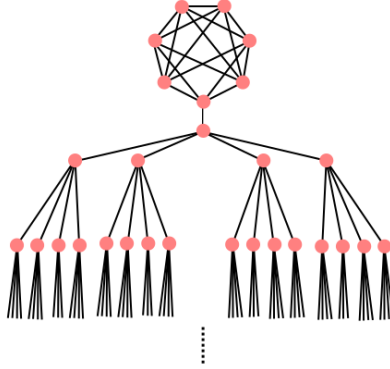


Figure 1: For a regular graph composed of a near-clique attached to an infinite tree, a closed walk of length $2k$ starting from within the near-clique does not typically go deeper than $O(\log k)$ down the tree. However, the support of such a closed walk is typically $k^{\Theta(1)}$. See Section B for a more detailed discussion.

Corollary 1.5. Let $H = (V, E)$ be an irregular connected graph of maximum degree Δ with at least two vertices, and let ϕ_H be the ℓ_2 -normalized Perron vector of A_H . Then there is a vertex $u \in V$ with degree strictly less than Δ satisfying

$$\phi_H(u) \geq 1/(\Delta^2 \lambda_1(A_H) |V|^{5/2}). \quad (4)$$

Corollary 1.5 may be compared with existing results in spectral graph theory on the “principal ratio” between the largest and smallest entries of the Perron vector of a connected graph. The known worst case lower bounds on this ratio are necessarily exponential in the diameter of the graph [CG07, TT15], and it is known that the Perron entry of the highest degree vertex cannot be very small (see e.g. [Ste14, Ch. 2]). Corollary 1.5 articulates that there is always at least one vertex of non-maximal degree for which the ratio is only polynomial in the number of vertices.

The proof of Theorem 1.4 is based on an analysis of hitting times in the simple random walk on G via electrical flows, and appears in Section 2. Combined with a perturbation-theoretic argument, it enables us to show that any small connected induced subgraph S of G can be extended to a slightly larger induced subgraph with significantly larger Perron value $\lambda_1(\tilde{A}_S)$. With some further combinatorial arguments, this implies that closed walks cannot concentrate on small sets, yielding Theorem 1.3 in Section 3, which is finally used to deduce Theorem 1.2 in Section 4.

We show in Section 5.2 via an explicit example (Figure 2) that the exponent of $5/2$ appearing in Corollary 1.5 is sharp up to polylogarithmic factors. We conclude with a discussion of open problems in Section 6.

Remark 1.6 (Higher Eigenvalues). An update of the preprint of [JTY⁺19] generalizes Theorem 1.1 to the multiplicity of the j th eigenvalue. Our results can also be generalized in this manner by some nominal changes to the arguments in Section 4, but for simplicity we focus on λ_2 in this paper.

1.1 Higher degree regular graphs

If $G = (V, E)$ is a simple, d -regular graph, and $S \subsetneq V$ such that $|S| \leq d$, then necessarily all vertices of S are adjacent to vertices in $V \setminus S$. Therefore we can improve the bound from Theorem 1.4

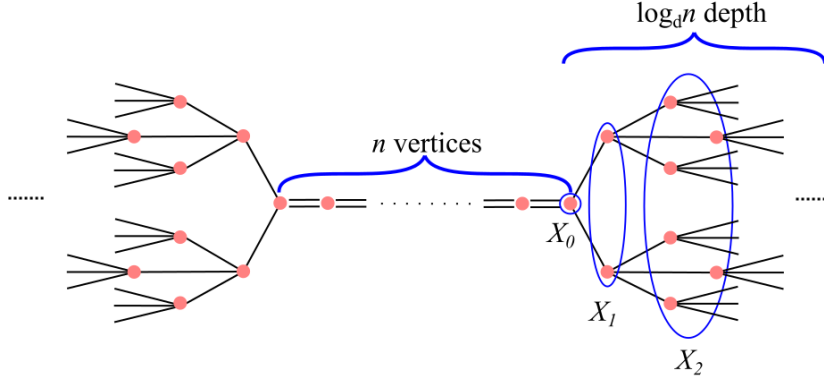


Figure 2: An example of a graph where all vertices u that are not of maximum degree have $\psi(u) = \tilde{O}(n^{-5/2})$. The circled sets X_0 , X_1 and X_2 will be used in the analysis of the graph in Section 5.2.

by assuming the vertex on the boundary is the maximizer of the Perron vector, which has value $\psi_S(u) \geq 1/\sqrt{|S|}$. This leads to the following variants of our main results for simple, regular graphs of sufficiently high degree.

Theorem 1.7. *G is simple, d -regular, and connected with $\lambda_2 = \lambda_2(A_G)$, then*

$$m_G \left(\left[\left(1 - \frac{\log \log_d n}{\log_d n} \right) \lambda_2, \lambda_2 \right] \right) = \begin{cases} \tilde{O} \left(\frac{n}{d} \right) & \text{when } d = o(\log^{1/4} n) \\ \tilde{O} \left(\frac{n \log^{1/2} d}{\log^{1/4} n} \right) & \text{when } d = \Omega(\log^{1/4} n). \end{cases} \quad (5)$$

The above theorem is based on the following corresponding result for closed walks.

Theorem 1.8. *If G is simple, d -regular, and connected on n vertices and γ is a random closed walk of length $2k < n$ started at any vertex in G , then:*

$$\Pr(\text{support}(\gamma) \leq s) \leq \exp \left(-\frac{k}{100s^3} \right) \quad \text{for } s \leq \min \left\{ \frac{1}{8} \left(\frac{k}{\log d} \right)^{1/4}, \frac{d}{2} \right\}. \quad (6)$$

The proofs of both theorems appear in Appendix A

1.2 Related work

Eigenvalue Multiplicity. Despite the straightforward nature of the question, relatively little is known about eigenvalue multiplicity of general graphs. As discussed in [JTY⁺19], if one assumes that G is a bounded degree expander graph, then the bound of Theorem 1.1 can be improved to $O(n/\log n)$. On the other hand, if G is assumed to be a Cayley graph of bounded doubling constant K (indicating non-expansion), then [LM08] show that the multiplicity of the second eigenvalue is at most $\exp(\log^2 K)$. In the context of Cayley graphs, one interesting new implication of Theorem 1.7 is that all Cayley graphs of degree $O(\exp(\log^{1/2-\delta} n))$ have second eigenvalue multiplicity $O(n/\log^{\delta/2} n)$.

Distance regular graphs of diameter D have exactly $D+1$ distinct eigenvalues (see [God93] 11.4.1 for a proof). However, besides the top eigenvalue (which must have multiplicity 1), generic bounds

on the multiplicity of the other eigenvalues are not known. As expanding graphs have diameter $\Theta(\log_d n)$, the average multiplicity of eigenvalues besides λ_1 for expanding distance regular graphs is $\Theta(n/\log_d n)$. It is tempting to see this as a hint that the multiplicity of the second eigenvalue could be $\Omega(n/\log_d n)$.

Sublinear multiplicity does not necessarily hold for eigenvalues in the interior of the spectrum even assuming bounded degree. In particular, Rowlinson has constructed connected d -regular graphs with an eigenvalue of multiplicity at least $n(d-2)/(d+2)$ [Row19] for constant d .

Higher Order Cheeger Inequalities. The results of [LRTV12, LOGT14] imply that if a d -regular graph G has a second eigenvalue multiplicity of m , then its vertices can be partitioned into $\Omega(m)$ disjoint sets each having edge expansion $O(\sqrt{d(1-\lambda_2)\log m})$. Combining this with the observation that a set cannot have expansion less than the reciprocal of its size shows that $m = O(n/\text{polylog}(n))$ whenever $1 - \lambda_2(\tilde{A}_G) \leq 1/\log^c n$ for any $c > 1$, i.e., the graph is sufficiently non-expanding. Our main theorem may be interpreted as saying that this phenomenon persists for all graphs.

Support of Walks. There are as far as we are aware no known lower bounds for the support of a random closed walk of fixed length in a general graph (or even Cayley graph). It is relatively easy to derive such bounds for bounded degree graphs if the length of the walk is sufficiently larger than the mixing time of the simple random walk on the graph; the key feature of Theorem 1.3, which is needed for our application, is that the length of the walk can be taken to be much smaller.

The support of open walks (namely removing the condition that the walk ends at the starting point) is better understood. There are Chernoff-type bounds on the size of the support of a random walk based on the spectral gap [Gil98, Kah97]. Such bounds and their variants are an important tool in derandomization.

Entries of the Perron Vector. There is a large literature concerning the magnitude of the entries of the Perron eigenvector of a graph — see [Ste14, Chapter 2] for a detailed discussion of results up to 2014. Rowlinson showed sufficient conditions on the Perron eigenvector for which changing the neighborhood of a vertex increases the spectral radius [Row90]. Cvetković, Rowlinson, and Simić give a condition which, if satisfied, means a given edge swap increases the spectral radius [CRS93]. Cioabă showed that for a graph of maximum degree Δ and diameter D , $\Delta - \lambda_1 > 1/nD$ [Cio07]. Cioabă, van Dam, Koolen, and Lee then showed that $\lambda_1 \geq (n-1)^{1/D}$ [CVDKL10]. The results of [VMSK⁺11] prove a lemma similar to Lemma 3.2, giving upper and lower bounds on the change in spectral radius from the deletion of edges. However, their result does not quite imply Lemma 3.2, and we prove a slightly different statement.

1.3 Notation

All logarithms are base e unless noted otherwise.

Electrical Flows. We use $\text{Reff}_H(\cdot, \cdot)$ to denote the effective resistance between two vertices in H , viewing each edge of the graph as a unit resistor. See e.g. [DS84] or [Bol13, Chapter IX] for an introduction to electrical flows and random walks on graphs.

Graphs. For a matrix M , we use M_S to denote the principal submatrix of M corresponding to the indices in S . Consider a graph $G = (V, E)$ and a subset $H \subset V$. Let $P := AD^{-1}$ be the transition matrix of the simple random walk matrix on G , where A is the adjacency matrix and D is the diagonal matrix of degrees. We will also use the normalized adjacency matrix $\tilde{A} := D^{-1/2}AD^{-1/2}$. Note that P and \tilde{A} are similar, and that \tilde{A} is symmetric. P_S and \tilde{A}_S are submatrices of P and \tilde{A} ; they are not the transition matrices and normalized adjacency matrices of the induced subgraph on S . Note P_S and \tilde{A}_S are also similar.

Perron Eigenvector. We use ψ_S to denote the ℓ_2 -normalized eigenvector corresponding to $\lambda_1(\tilde{A}_S)$, which is a simple eigenvalue if S is connected. Note that for connected S , ψ_S is strictly positive by the Perron-Frobenius theorem.

A simple graph refers to a graph without multiedges or self-loops. We assume $\Delta \geq 2$ for all connected regular graphs, since otherwise the graph is just an edge, so $\log \Delta > 0$.

2 Lower Bounds on the Perron Eigenvector

In this section we prove Theorem 1.4, which is a direct consequence of the following slightly more refined result. In a graph $G = (V, E)$, define the boundary of S as the set of vertices in S adjacent to $V \setminus S$ in G .

Theorem 2.1 (Large Perron Entry). *Let $G = (V, E)$ be a connected graph of maximum degree Δ and $S \subsetneq V$ such that the induced subgraph on S is connected. Then there is a vertex $u \in S$ on the boundary of S such that*

$$\psi_S(u)/\psi_S(t) \geq 1/(\Delta^{5/2} \lambda_1(\tilde{A}_S) |S|^2) \quad (7)$$

where $t = \arg \max_{w \in S} \psi_S(w)$.

At a high level, the proof proceeds as follows. First, we show that there exists a vertex $x \in S$ adjacent to the boundary of S such that a random walk started at x is somewhat likely to hit t before it hits the boundary of S . Second, we express the ratio of $D_S^{1/2} \psi_S(x)$ and $D_S^{1/2} \psi_S(t)$ as a limit as $k \rightarrow \infty$ of the ratio $\mathbb{P}Y_x^k / \mathbb{P}Y_t^k$, where Y_v^k is the event that the simple random walk started at v remains in S for k steps; we bound this ratio from below using the hitting time estimate from the first step. Third, by the eigenvector equation the ratio of the entries of an eigenvector at two neighboring vertices is bounded. Hence, x is adjacent to some vertex u on the boundary of S satisfying the theorem.

Proof. Write $S = M \sqcup B$, where B is the boundary of S and $M = S \setminus B$. If $t \in B$ then we are done, so assume not. Let $\mathbb{P}_x^G(\cdot)$ denote the law of the simple random walk (SRW) $(X_i)_{i=0}^\infty$ on G started at $X_0 = x$, and for any subset $T \subset V$, let $\tau_T := \{\min i : X_i \in T\}$ denote the hitting time of the SRW to that subset; if $T = \{u\}$ is a singleton we will simply write τ_u .

Step 1. We begin by showing that there is a vertex $x \in M$ adjacent to B for which the random walk started at x is reasonably likely to hit t before B . To do so, we use the well-known connection between hitting probabilities in random walks and electrical flows. Define a new graph $K = (V' = V \setminus B \cup \{s\}, E')$ by contracting all vertices in B to a single vertex s . Let $f : V' \rightarrow [0, 1]$ be the vector of voltages in the electrical flow in K with boundary conditions $f(s) = 0, f(t) = 1$, regarding every edge as a unit

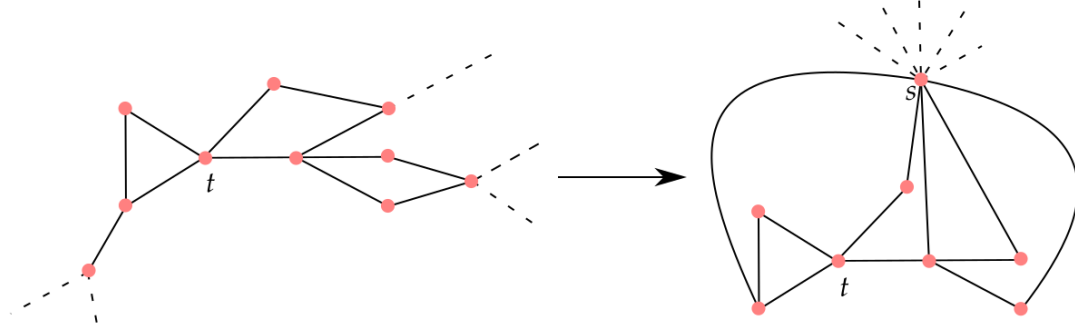


Figure 3: In Step 1 of the proof of Theorem 2.1, we lower bound the probability that a random walk started at a certain vertex x adjacent to B reaches t before reaching B . We do this by contracting B to a vertex s , then lower bounding the current from s to t , which establishes the existence of the desired x . The left graph in the figure is G and the right graph is the contracted graph K , with the dotted lines indicating edges leaving the set of interest $S = M \sqcup B$.

resistor. By Ohm's law, the current flow from s to t is equal to $1/\text{Reff}_K(s, t)$. We have the crude upper bound

$$\text{Reff}_K(s, t) \leq \text{distance}_K(s, t) \leq |S|,$$

since S is connected, so the outflow of current from s is at least $1/|S|$. By Kirchhoff's current law, there must be a flow of at least $1/(|S| \deg_K(s))$ on at least one edge $(s, x) \in E'$. By Ohm's law again, for this particular $x \in V'$ we must have

$$f(x) \geq \frac{1}{|S| \deg_K(s)} = \frac{1}{|S| |\partial_G B|} \geq \frac{1}{\Delta |S|^2}, \quad (8)$$

where $\partial_G B$ denotes the edge boundary of B in G . Appealing to e.g. [Bol13, Chapter IX, Theorem 8], this translates to the probabilistic bound

$$\mathbb{P}_x^G(\tau_t < \tau_B) = \mathbb{P}_x^K(\tau_t < \tau_s) = f(x) \geq \frac{1}{\Delta |S|^2}. \quad (9)$$

Finally, since $f(s) = f(y) = 0$ for every $y \in V \setminus S$, we must in fact have $x \in M$.

Step 2. We now use (9) to show that $\psi_S(x)$ is large. Because $\tilde{A}_S = D_S^{-1/2} P_S D_S^{1/2}$, the top eigenvector of P_S is $D_S^{1/2} \psi_S / \|D_S^{1/2} \psi_S\|$. Let $P' : (P + I)/2$ denote the lazy random walk² on G , and to ease notation let $\mathbb{P}'_x(\cdot) := \mathbb{P}_x^G(\cdot)$ denote the law of the lazy random walk on G started at x . Note that the eigenvectors of P_S , as well as $\mathbb{P}_x(\tau_t < \tau_B)$, do not change when passing to P'_S .

For the lazy random walk, the Perron-Frobenius theorem implies that

$$\frac{(D_S^{1/2} \psi_S)(w)}{\|D_S^{1/2} \psi_S\|} = \lim_{k \rightarrow \infty} \frac{\mathbf{1}_S^T P_S^k e_w}{\|\mathbf{1}_S^T P_S^k\|},$$

²This modification is only to ensure non-bipartiteness; if S is not bipartite we may take the simple random walk

for every $w \in S$, where $\mathbf{1}_S \in \mathbb{R}^S$ is the all ones vector. We further have

$$\mathbf{1}_S^T P_S'^k e_w = \mathbb{P}'_w(\tau_{V \setminus S} > k),$$

namely the probability a random walk of length k starting at w stays in S .

We are interested in the ratio

$$\frac{(D_S^{1/2} \psi_S)(x)}{(D_S^{1/2} \psi_S)(t)} = \lim_{k \rightarrow \infty} \frac{\mathbb{P}'_x(\tau_{V \setminus S} > k)}{\mathbb{P}'_t(\tau_{V \setminus S} > k)}. \quad (10)$$

Fix an integer $k > 0$. The numerator of (10) is bounded as

$$\begin{aligned} \mathbb{P}'_x(\tau_{V \setminus S} > k) &\geq \mathbb{P}'_x(\tau_{V \setminus S} > k | \tau_t < \tau_B) \mathbb{P}'_x(\tau_t < \tau_B) \\ &\geq \frac{1}{\Delta |S|^2} \mathbb{P}'_x(\tau_{V \setminus S} > k | \tau_t < \tau_B) \quad \text{by (9)} \\ &\geq \frac{1}{\Delta |S|^2} \sum_{\theta=0}^{k-1} \mathbb{P}'_x(\tau_{V \setminus S} > k | \tau_t = \theta, \tau_t < \tau_B) \mathbb{P}'_x(\tau_t = \theta | \tau_t < \tau_B) \end{aligned} \quad (11)$$

$$\begin{aligned} &= \frac{1}{\Delta |S|^2} \sum_{\theta=0}^{k-1} \mathbb{P}'_t(\tau_{V \setminus S} > k - \theta) \mathbb{P}'_x(\tau_t = \theta | \tau_t < \tau_B) \\ &\geq \frac{1}{\Delta |S|^2} \sum_{\theta=0}^{k-1} \mathbb{P}'_t(\tau_{V \setminus S} > k) \mathbb{P}'_x(\tau_t = \theta | \tau_t < \tau_B). \end{aligned} \quad (12)$$

Observe that $\mathbb{E}'_x \tau_B < \infty$ since G is connected. Thus,

$$\begin{aligned} \sum_{\theta=0}^{k-1} \mathbb{P}'_x(\tau_t = \theta | \tau_t < \tau_B) &= 1 - \mathbb{P}'_x(\tau_t \geq k | \tau_t < \tau_B) \\ &\geq 1 - \frac{\mathbb{P}'_x(\tau_B \geq k)}{\mathbb{P}'_x(\tau_t < \tau_B)} \\ &\geq 1 - \frac{\mathbb{E}'_x \tau_B}{k} \cdot \Delta |S|^2 \quad \text{by Markov and (9)}. \end{aligned}$$

Combining this bound with (12), we have

$$\mathbb{P}'_x(\tau_{V \setminus S} > k) \geq \frac{1}{\Delta |S|^2} \left(1 - \frac{\mathbb{E}'_x \tau_B}{k} \cdot \Delta |S|^2 \right) \mathbb{P}'_t(\tau_{V \setminus S} > k)$$

Taking the limit as $k \rightarrow \infty$ in (10) yields

$$\frac{(D_S^{1/2} \psi_S)(x)}{(D_S^{1/2} \psi_S)(t)} \geq \frac{1}{\Delta |S|^2}.$$

Step 3. Since x is adjacent to B , we can choose a $u \in B$ adjacent to x . The eigenvector equation and nonnegativity of the Perron vector now imply $\Delta\lambda_1(A_S)\psi_S(u) \geq \psi_S(x)$, whence

$$(D_S^{1/2}\psi_S)(u) \geq \frac{1}{\lambda_1(\tilde{A}_S)\Delta^2|S|^2}(D_S^{1/2}\psi_S)(t). \quad (13)$$

Therefore, as D is a diagonal matrix, and the entries of D range from 1 to Δ , it must be the case that

$$\psi_S(u) \geq \frac{1}{\lambda_1(\tilde{A}_S)\Delta^{5/2}|S|^2}\psi_S(t).$$

□

Remark 2.2. As the proof shows, the right-hand side of (7) may be replaced with $1/\Delta^{3/2}\lambda_1(\tilde{A}_S)|\partial_G B|R$ where B is the boundary of S in G and R is the maximum effective resistance between two vertices in S .

Proof of Corollary 1.5. Given an irregular graph H , construct a Δ -regular graph G containing H as an induced subgraph (it is trivial to do this if we allow G to be a multigraph). Repeating the above proof on G with $S = H$ and observing that $D_S^{1/2}$ is a multiple of the identity since G is regular, (13) yields the desired conclusion. □

3 Support of Closed Walks

In this section we prove Theorem 1.3, which is an immediate consequence of the following slightly stronger result. Let $W^{2k,s}$ denote the event a simple random walk of length $2k$ has support at most s and ends at its starting point.

Theorem 3.1 (Implies Theorem 1.3). *If G is connected and of maximum degree Δ on n vertices, then for every vertex $x \in G$ and $k < n/2$,*

$$\mathbb{P}_x(W^{2k,s}) \leq \exp\left(-\frac{k}{65\Delta^7 s^4}\right)\mathbb{P}_x(W^{2k,2s}) \quad \text{for } s \leq \frac{1}{4}\left(\frac{k}{\Delta^7 \log \Delta}\right)^{1/5}. \quad (14)$$

The proof requires a simple lemma lower bounding the increase in the Perron value of a subgraph upon adding a vertex in terms of the Perron vector.

Lemma 3.2 (Perturbation of λ_1). *Take the normalized adjacency matrix $\tilde{A} := D^{-1/2}AD^{-1/2}$ of a graph $G = (V, E)$ of maximum degree Δ . For any $S \subseteq V$ and vertex $u \in S$, the submatrix which includes the subset $S' = (S \cup \{v\}, E(S) \cup \{(u, v)\})$, which adds a vertex v and the edge (u, v) to S , satisfies*

$$\lambda_1(\tilde{A}_{S'}) \geq \frac{1}{2}\left(\lambda_1(\tilde{A}_S) + \sqrt{\lambda_1(\tilde{A}_S)^2 + \Delta^{-2}\psi_S(u)^2}\right).$$

Proof. The largest eigenvalue of \tilde{A} is at least the quadratic form associated with the unit vectors

$$g_\alpha(x) = \begin{cases} \sqrt{1 - \alpha^2}\psi_S(x) & x \in V \\ \alpha & x = v \end{cases}$$

for $0 \leq \alpha \leq 1$. We have $g_\alpha^T \tilde{A} g_\alpha = (1 - \alpha^2) \lambda_1(\tilde{A}_S) + d_u^{-1/2} d_v^{-1/2} \alpha \sqrt{1 - \alpha^2} \psi_S(u)$, where d_u is the degree of u in G . This quantity is maximized when

$$\alpha = \sqrt{\frac{1}{2} - \frac{\lambda_1(\tilde{A}_S)}{2\sqrt{\lambda_1(\tilde{A}_S)^2 + d_u^{-1} d_v^{-1} \psi_S(u)^2}}},$$

at which

$$g_\alpha^T \tilde{A} g_\alpha = \frac{1}{2} \left(\lambda_1(\tilde{A}_S) + \sqrt{\lambda_1(\tilde{A}_S)^2 + d_u^{-1} d_v^{-1} \psi_S(u)^2} \right).$$

□

Combining Lemma 3.2 and Theorem 2.1 yields a bound on the increase of the top eigenvalue of the submatrix corresponding to an induced subgraph that may be achieved by adding vertices to it.

Lemma 3.3 (Support Extension). *For any connected graph $G = (V, E)$ of maximum degree Δ , consider its normalized adjacency matrix \tilde{A} . For any connected subset $S \subsetneq V$ such that $2 \leq |S| = s < |V|/2$, there is a connected subset $T \subset V$ containing S such that $|T| = 2s$ and*

$$\lambda_1(\tilde{A}_T) \geq \lambda_1(\tilde{A}_S) \left(1 + \frac{5}{128\Delta^7 s^4} \right).$$

Proof. Define $\lambda_1 := \lambda_1(\tilde{A}_S)$ and note that $\lambda_1 \geq 1/\Delta$ since S contains at least one edge. As ψ_S is a normalized vector with s entries, $\psi_S(t) \geq 1/\sqrt{s}$. Therefore $\psi_S(u) \geq 1/(\Delta^{5/2} \lambda_1^{5/2})$. Take v to be any vertex in $V \setminus S$ that neighbors u in G . By Lemma 3.2,

$$\begin{aligned} \lambda_1(\tilde{A}_{S \cup \{v\}}) &\geq \frac{1}{2} \left(\lambda_1 + \sqrt{\lambda_1^2 + \Delta^{-2} \psi_S(u)^2} \right) \\ &\geq \lambda_1 + \frac{\psi_S(u)^2}{4\lambda_1 \Delta^2} - \frac{\psi_S(u)^4}{16\lambda_1^3 \Delta^4} \\ &\geq \lambda_1 + \frac{1}{6\lambda_1^3 \Delta^7 s^5} \quad \text{as } \psi_S(u)^2 / \lambda_1^2 \leq \Delta^2 \\ &\geq \lambda_1 + \frac{1}{6\Delta^7 s^5} \quad \text{since } \lambda_1 \leq 1. \end{aligned} \tag{15}$$

Assuming that $s < |V|/2$, we can iterate this process s times, adding the vertices $\{v_1, \dots, v_s\}$. At each step we add the vertex v_i and increase the Perron eigenvalue of $\tilde{A}_{S \cup \{v_1, \dots, v_{i-1}\}}$ by at least $1/(6\Delta^7(s+i-1)^5)$. Therefore, defining $T = S \cup \{v_1, \dots, v_s\}$, we have

$$\lambda_1(\tilde{A}_T) \geq \lambda_1 + \frac{1}{6\Delta^7} \sum_{i=1}^s \frac{1}{(s+i-1)^5} \geq \lambda_1 + \frac{5}{128\Delta^7 s^4},$$

where the last inequality follows from approximating the sum with the integral. As $\lambda_1 \leq 1$, this translates to the desired multiplicative bound. □

Proof of Theorem 3.1. We begin by showing (14). Let Γ_x^s be the set of connected subgraphs of G with s vertices containing x . Choose S to be the maximizer of $e_x^T \tilde{A}_S^{2k} e_x$ among $S \in \Gamma_x^s$, and let $T \in \Gamma_x^{2s}$ be the extension of S guaranteed by Lemma 3.3 to satisfy

$$\lambda_1(\tilde{A}_T) \geq \left(1 + \frac{5}{128\Delta^7 s^4}\right) \lambda_1(\tilde{A}_S).$$

P_S^{2k} has the same diagonal entries as \tilde{A}_S^{2k} , so

$$\mathbb{P}_x(W^{2k,s}) \leq \sum_{S' \in \Gamma_x^s} e_x^T \tilde{A}_{S'}^{2k} e_x,$$

since each walk of length $2k$ satisfying $W^{2k,s}$ is contained in at least one $S' \in \Gamma_x^s$. Furthermore, $|\Gamma_x^s| \leq \Delta^{2s}$ since each subgraph of Γ_x^s may be encoded by one of its spanning trees, which may in turn be encoded by a closed walk rooted at x traversing the edges of the tree. We then have

$$\begin{aligned} \mathbb{P}_x(W^{2k,s}) &\leq |\Gamma_x^s| e_x^T \tilde{A}_S^{2k} e_x \\ &\leq \Delta^{2s} \lambda_1(\tilde{A}_S)^{2k} \\ &\leq \Delta^{2s} \left(1 + \frac{5}{128\Delta^7 s^4}\right)^{-2k} \lambda_1(\tilde{A}_T)^{2k}. \end{aligned} \quad (16)$$

We will bound the right hand side in terms of $\mathbb{P}_x(W^{2k,2s})$.

We claim that for every $z \in T$,

$$e_x^T \tilde{A}_T^{2k} e_x \geq \Delta^{-4s} e_z^T \tilde{A}_T^{2k-4s} e_z. \quad (17)$$

To see this, let π be a path in T of length $\ell \leq 2s$ between x and z , which must exist since T is connected and has size $2s$. Then every closed walk of length $2k - 2\ell$ in T rooted at z may be extended to a walk of length $2k$ in T rooted at x by attaching π and its reverse. Performing the walk of π twice occurs with probability at least $\Delta^{-2\ell}$. Since all of the walks produced this way are distinct, we have

$$e_x^T \tilde{A}_T^{2k} e_x \geq \Delta^{-2\ell} e_z^T \tilde{A}_T^{2k-2\ell} e_z.$$

By the same argument $e_z^T \tilde{A}_T^{2k-2\ell} e_z \geq \Delta^{-4s+2\ell} e_z^T \tilde{A}_T^{2k-4s} e_z$, and inequality (17) follows.

Choose $z \in T$ to be the maximizer of $e_z^T \tilde{A}_T^{2k-4s} e_z$, for which we have:

$$e_z^T \tilde{A}_T^{2k-4s} e_z \geq \frac{1}{2s} \text{Tr}(P_T^{2k-4s}) \geq \frac{\lambda_1(\tilde{A}_T)^{2k-4s}}{2s}.$$

Combining this with (17) and substituting in (16), we obtain

$$\begin{aligned} \mathbb{P}_x(W^{2k,s}) &\leq \Delta^{6s} \cdot 2s \left(1 + \frac{5}{128\Delta^7 s^4}\right)^{-2k} \lambda_1(\tilde{A}_T)^{4s} e_x^T \tilde{A}_T^{2k} e_x \\ &\leq \Delta^{6s} \cdot 2s \left(1 + \frac{5}{128\Delta^7 s^4}\right)^{-2k} \lambda_1(\tilde{A}_T)^{4s} \mathbb{P}_x(W^{2k,2s}). \end{aligned}$$

Applying the inequality $e^{x/2} \leq 1 + x$ for $0 < x < 1$ and the bound $\lambda_1(\tilde{A}_T) < 1$, we obtain

$$\mathbb{P}_x(W^{2k,s}) \leq \exp\left(6s \log \Delta + \log(2s) - \frac{5k}{128\Delta^7 s^4}\right) \mathbb{P}_x(W^{2k,2s}), \quad (18)$$

which implies

$$\mathbb{P}_x(W^{2k,s}) \leq \exp\left(-\frac{k}{65\Delta^7 s^4}\right) \mathbb{P}_x(W^{2k,2s})$$

whenever

$$s \leq \frac{1}{4} \left(\frac{k}{\Delta^7 \log(\Delta)} \right)^{1/5},$$

establishing (14). □

4 Bound on Eigenvalue Multiplicity

In this section we prove Theorem 1.2, restated below in slightly more detail.

Theorem 4.1 (Detailed Theorem 1.2). *Let G be a maximum degree Δ connected graph on n vertices. If³ $\Delta \leq \log^{1/7} n / \log \log n$ then the spectrum of the normalized adjacency matrix \hat{A} satisfies*

$$m_G\left(\left[1 - \frac{\log \log_{\Delta} n}{\log_{\Delta} n}\right] \lambda_2, \lambda_2\right] = O\left(n \cdot \frac{\Delta^{7/5} (\log^{2/5} \Delta) \log \log n}{\log^{1/5} n}\right). \quad (19)$$

Proof. For now, assume that $|\lambda_n(P)| \leq |\lambda_2(P)|$. Let $\mathbb{P}(\cdot)$ denote the law of an SRW γ of length $2k$ on G , started at a vertex chosen uniformly at random (i.e., *not* from the stationary measure of the SRW). Let $W^{2k} := W^{2k,n}$ denote the event that γ returns to its starting vertex after $2k$ steps. In an abuse of notation, let $W^{2k, \geq s+1} := W^{2k} \setminus W^{2k,s}$ be the event that a walk of length $2k$ is closed and has support at least $s+1$.

Set $k := \frac{1}{3} \log_{\Delta} n$ and $c := 2 \log k$ and let s be a parameter satisfying

$$\mathbb{P}(W^{2k,s}) \leq e^{-c} \mathbb{P}(W^{2k}) \quad (20)$$

to be chosen later. Delete cn/s vertices from G uniformly at random and call the resulting graph H .

If γ has support at least $s+1$, then the probability that none of the vertices of γ are deleted is at most

$$\left(1 - \frac{s}{n}\right)^{\frac{cn}{s}} \leq e^{-c}.$$

Thus,

$$\mathbb{E}_H \mathbb{P}(\gamma \subset H | \gamma \in W^{2k, \geq s+1}) \leq e^{-c},$$

where \mathbb{E}_H is the expectation over H . It then follows by the probabilistic method that there exists a deletion such that the resulting subgraph H of G satisfies

$$\mathbb{P}(W^{2k, \geq s+1} \cap \{\gamma \subset H\}) \leq e^{-c} \mathbb{P}(W^{2k, \geq s+1}).$$

³If $\Delta \geq \log^{1/7} n / \log \log n$ then (1) is vacuously true.

Write $\lambda_2 := \lambda_2(\tilde{A}_G)$ and let m' be the number of eigenvalues of H in the interval $[(1 - \epsilon)\lambda_2, \lambda_2]$ for $\epsilon := c/2 \log_\Delta(n)$. Since $2k$ is even,

$$\begin{aligned}
m'(1 - \epsilon)^{2k} \lambda_2^{2k} &\leq \text{Tr}(\tilde{A}_H^{2k}) \\
&= n\mathbb{P}(W^{2k} \cap \{\gamma \subset H\}) \\
&= n(\mathbb{P}(W^{2k,s} \cap \{\gamma \subset H\}) + \mathbb{P}(W^{2k,\geq s+1} \cap \{\gamma \subset H\})) \\
&\leq n(\mathbb{P}(W^{2k,s}) + e^{-c}\mathbb{P}(W^{2k,\geq s+1})) \quad \text{by our choice of } H \\
&\leq n(e^{-c}\mathbb{P}(W^{2k}) + e^{-c}\mathbb{P}(W^{2k,\geq s+1})) \quad \text{by (20)} \\
&\leq 2e^{-c} \text{Tr}(\tilde{A}_G^{2k}) \\
&\leq 2e^{-c}(n\lambda_2^{2k} + 1).
\end{aligned}$$

We may assume that the diameter of G is at least 10 as otherwise $\Delta \geq n^{1/10}$, making the theorem statement vacuous. Because of the diameter, we can take four edges $(u_1, v_1), (a_1, b_1), (u_2, v_2), (a_2, b_2)$ such that the distance between each pair of edges is at least 2. Then consider the vectors ϕ_1, ϕ_2 such that for $i \in \{1, 2\}$

$$\phi_i(x) = \begin{cases} 1 & x \in \{u_i, v_i\} \\ -1 & x \in \{a_i, b_i\} \\ 0 & \text{otherwise} \end{cases}$$

Choose real numbers α and β such that at least one is nonzero. We have

$$\frac{(\alpha\phi_1 + \beta\phi_2)^T D^{-1/2} A D^{-1/2} (\alpha\phi_1 + \beta\phi_2)}{(\alpha\phi_1 + \beta\phi_2)^T (\alpha\phi_1 + \beta\phi_2)} \geq \frac{\frac{4}{\Delta}(\alpha^2 + \beta^2)}{4(\alpha^2 + \beta^2)} \geq \frac{1}{\Delta}.$$

Therefore by Courant Fisher

$$\lambda_2 \geq \min_{\alpha, \beta} \frac{(\alpha\phi_1 + \beta\phi_2)^T D^{-1/2} A D^{-1/2} (\alpha\phi_1 + \beta\phi_2)}{(\alpha\phi_1 + \beta\phi_2)^T (\alpha\phi_1 + \beta\phi_2)} \geq \frac{1}{\Delta}.$$

By our choice of k , this means $n\lambda_2^{2k} \geq 1$. Moreover,

$$\epsilon \leq \frac{2 \log \log n}{2 \log_\Delta n} \leq \frac{\log \Delta \log \log n}{\log n} < 1/2,$$

based on our assumptions on Δ . Thus, $1 - \epsilon \geq e^{-1.5\epsilon}$. Combining these facts,

$$m' \lambda_2^{2k} \leq 4e^{3k\epsilon - c} n \lambda_2^{2k},$$

yielding

$$m' \leq 4ne^{3k\epsilon - c} \leq 4ne^{-c/2} = O\left(\frac{n}{\log_\Delta n}\right).$$

As we created H by deleting cn/s vertices, it follows by Cauchy interlacing that the number of eigenvalues of \tilde{A} in $[(1 - \epsilon)\lambda_2, \lambda_2]$ is at most

$$\frac{cn}{s} + O\left(\frac{n}{\log_\Delta n}\right).$$

We now show that taking

$$s := \frac{1}{4} \left(\frac{k}{\Delta^7 \log \Delta} \right)^{1/5}$$

satisfies (20). Applying Theorem 3.1 equation (14) to each $x \in G$ and summing, we have

$$\begin{aligned} \frac{\mathbb{P}(W^{2k,s})}{\mathbb{P}(W^{2k})} &\leq \exp\left(-\frac{k}{65\Delta^7 s^4}\right) \\ &\leq \exp\left(-\Omega\left(\frac{\log n \log^{2/5} \Delta}{\Delta^{7/5}}\right)\right) \\ &\ll \exp(-c) = \exp(-\Theta(\log \log_{\Delta} n)), \end{aligned}$$

satisfying (20) for sufficiently large n , and we conclude that

$$m_G\left(\left[\left(1 - \frac{\log \log_{\Delta} n}{\log_{\Delta} n}\right)\lambda_2, \lambda_2\right]\right) = O\left(n \cdot \frac{\Delta^{7/5} \log^{2/5} \Delta \log \log n}{\log^{1/5} n}\right),$$

as desired.

If $|\lambda_n| > |\lambda_2|$, we can do a lazy walk with probability of moving $p = \frac{1}{2}$, therefore making all eigenvalues nonnegative. This is equivalent to doubling the degree of every vertex by adding loops. This is the equivalent of taking the simple random walk on a graph with maximum degree 2Δ , requiring $s \leq \frac{1}{11} \left(\frac{k}{\Delta^7 \log \Delta} \right)^{1/5}$, yielding the same asymptotics. \square

5 Examples

In this section, we consider examples demonstrating some of the points raised in the introduction regarding the tightness of our results. As most of our results in this section are combinatorial rather than probabilistic, we will consider multiplicity in the non-normalized adjacency matrix A . For regular graphs, this is equivalent.

5.1 Bipartite Ramanujan Graphs

We show that bipartite Ramanujan graphs (see [LPS88]; known to exist for every $d \geq 3$ by [MSS15]) have high multiplicity near λ_2 . This means that the bound of $n/\log^{\Theta(1)} n$ of Theorem 1.2 is tight.

Theorem 5.1 (Friedman [Fri91] Corollary 3.6). *Let G be a d -regular graph on n vertices. Then*

$$\lambda_2(A_G) \geq 2\sqrt{d-1}(1 - O(1/\log^2 n)).$$

Lemma 5.2 (McKay [McK81] Lemma 3). *The number of closed walks on the infinite d -regular tree of length $2k$ starting at a fixed vertex is $\Theta\left(\frac{4^k(d-1)^k}{k^{3/2}}\right)$.*

Proposition 5.3. *There exists a constant $\alpha > 0$ such that for fixed d , every bipartite d -regular bipartite Ramanujan graph G on n vertices satisfies*

$$m_G\left(\left[\lambda_2\left(1 - \alpha \frac{\log \log(n)}{\log(n)}\right), \lambda_2\right]\right) = \Omega(n/\log^{3/2}(n)).$$

Proof. By Theorem 5.1,

$$\lambda_2 \left(1 - \alpha \frac{\log \log(n)}{\log(n)} \right) \leq 2 \sqrt{d-1} \left(1 - \frac{1}{2} \alpha \frac{\log \log(n)}{\log(n)} \right),$$

for sufficiently large n . Let $k = \beta \log n$ for some constant β to be set later and suppose that there are m eigenvalues of A_G in the interval $[2 \sqrt{d-1} \left(1 - \frac{1}{2} \alpha \frac{\log \log(n)}{\log(n)} \right), \lambda_2]$. Recall that the spectrum of a bipartite graph is symmetric around 0. From Lemma 5.2 it follows that for some constant C ,

$$\begin{aligned} Cn \left(\frac{4^k (d-1)^k}{k^{3/2}} \right) &\leq \sum_{i=1}^n \lambda_i(A_G)^{2k} \\ &\leq 2d^{2k} + (n-2m) \left(2 \sqrt{d-1} \left(1 - \frac{1}{2} \alpha \frac{\log \log(n)}{\log(n)} \right) \right)^{2k} + 2m(2 \sqrt{d-1})^{2k}. \end{aligned}$$

If we let β be sufficiently small and $\alpha > \frac{3}{2\beta}$, rearranging yields

$$\begin{aligned} \frac{m}{n} &\geq \frac{C \frac{4^k (d-1)^k}{k^{3/2}} - \frac{2d^{2k}}{n} - \left(2 \sqrt{d-1} \left(1 - \frac{1}{2} \alpha \frac{\log \log(n)}{\log(n)} \right) \right)^{2k}}{2(2 \sqrt{d-1})^{2k} \cdot \left(1 - \left(1 - \frac{1}{2} \alpha \frac{\log \log(n)}{\log(n)} \right)^{2k} \right)} \\ &= \Omega \left(\frac{1 - 2n^{2\beta-1}}{k^{3/2}} - \frac{\left(1 - \frac{1}{2} \alpha \frac{\log \log(n)}{\log(n)} \right)^{2k}}{1 - \left(1 - \frac{1}{2} \alpha \frac{\log \log(n)}{\log(n)} \right)^{2k}} \right) \\ &= \Omega \left(\frac{1}{k^{3/2}} - \frac{1}{e^{\alpha\beta \log \log(n)}} \right) \\ &= \Omega \left(\frac{1}{k^{3/2}} \right). \end{aligned}$$

□

5.2 Mangrove Tree

This section shows that the dependence on $|V|$ in Corollary 1.5 is tight up to polylogarithmic factors. Our example begins with a path of multiedges containing n vertices, where each multiedge of the path is composed of $d/2$ edges for some even d . At both ends of the path, we attach a tree of depth $\log_{d-1} n$. The roots have degree $d/2$ and all other vertices (besides the leaves) have degree d . Therefore the only vertices in the graph that are not degree d are the leaves of the two trees. Call this graph Q . An example of this graph is shown in Figure 2.

Proposition 5.4. *For every vertex u of degree less than d , $\psi_Q(u) = \tilde{O}(n^{-5/2})$, where \tilde{O} suppresses dependence on logarithmic factors and d .*

Therefore, we cannot hope to do significantly better than our analysis in Lemma 3.3, in which we find a vertex u of non-maximal degree with $\psi(u) \geq 1/(d\lambda_1 n^{5/2})$.

Proof. For simplicity, call $\lambda_1(A_Q) = \lambda_1$ and $\psi_Q = \psi$. By the symmetry of the graph, the value of ψ at vertices in the tree is determined by the distance from the root. Call the entries of ψ corresponding to the tree r_0, r_1, \dots, r_ℓ , where the index indicates the distance from the root.

By the discussion in the proof of Kahale [Kah95] Lemma 3.3, if we define

$$\theta := \log \left(\frac{\lambda_1}{2\sqrt{d-1}} + \sqrt{\frac{\lambda_1^2}{4(d-1)} - 1} \right),$$

then for $0 \leq i \leq \ell$, entries of the eigenvector must satisfy

$$\frac{r_i}{r_0} = \frac{\sinh((\ell+1-i)\theta)(d-1)^{-i/2}}{\sinh((\ell+1)\theta)}$$

where ℓ is the depth of the tree.

Therefore, $r_\ell/r_0 = \frac{\sinh(\theta)(d-1)^{-\ell/2}}{\sinh((\ell+1)\theta)}$. Examining the various terms, $\sinh(\theta) \leq d$ and $(d-1)^{-\ell/2} = \frac{1}{\sqrt{n}}$. To bound the third term, we use the definition $\sinh(x) = (e^x - e^{-x})/2$, which yields

$$\sinh((\ell+1)\theta) \geq \frac{1 - o_n(1)}{2} \left(\frac{\lambda_1}{2\sqrt{d-1}} + \sqrt{\frac{\lambda_1^2}{4(d-1)} - 1} \right)^{\log_{d-1} n+1}.$$

λ_1 is at least the spectral radius of the path of length n with $d/2$ multiedges between vertices. This spectral radius is $d \cos(\pi/(n+1))$. This gives

$$\begin{aligned} \sinh((\ell+1)\theta) &\geq \frac{1 - o_n(1)}{2(2\sqrt{d-1})^{\log_{d-1} n+1}} \left(d \left(1 - \frac{\pi^2}{2n^2}\right) + \sqrt{d^2 \left(1 - \frac{\pi^2}{2n^2}\right)^2 - 4d + 4} \right)^{\log_{d-1} n+1} \\ &\geq \frac{1 - o_n(1)}{2(2\sqrt{d-1})^{\log_{d-1} n+1}} (d + d - 2)^{\log_{d-1} n+1} \left(1 - O\left(\frac{d}{n^2}\right) \right)^{\log_{d-1} n+1} \\ &\geq \frac{1 - o_n(1)}{2} e^{-O(d \log_{d-1} n/n^2)} \sqrt{n} \geq \frac{\sqrt{n}}{3} \end{aligned}$$

for large enough n . Therefore

$$\frac{r_\ell}{r_0} = \frac{\sinh(\theta)(d-1)^{-\ell/2}}{\sinh((\ell+1)\theta)} \leq \frac{3d}{n}. \quad (21)$$

At this point, we know the ratio between r_ℓ and r_0 , but still need to bound the overall mass of the eigenvector on the tree. A “regular partition” is a partition of vertices $V = \bigsqcup_{i=0}^k X_i$ where the number of neighbors a vertex $u \in X_i$ has in X_j does not depend on u . We can create a *quotient matrix*, where entry i, j corresponds to the number of neighbors a vertex $u \in X_i$ has in X_j . For an overview of quotient matrices and their utility, see Godsil, [God93, Chapter 5]. In our partition, every vertex in the path is placed in a set by itself. The vertices of each of the two trees are partitioned into sets according to their distance from the two roots. Call the matrix according to this partition B_Q . We denote by $B_Q(X_i, X_j)$ the entry in B_Q corresponding to edges from a vertex in X_i to X_j .

Define X_0, \dots, X_ℓ as the sets corresponding to vertices in the first tree of distance $0, \dots, \ell$ from the root. For $1 \leq j \leq \ell - 1$, $B_Q(X_0, X_1) = d/2$. $B_Q(X_j, X_{j+1}) = d - 1$. Moreover, for $0 \leq j \leq \ell - 1$, $B_Q(X_{j+1}, X_j) = 1$. All values between vertices in the path are unchanged at $d/2$.

Consider the diagonal matrix D with $D_{i,i} := |X_i|^{-1/2}$. $D^{-1}B_QD$ is a symmetric matrix. Define $C := D^{-1}B_QD$. We now have $C(X_{j+1}, X_j) = C(X_j, X_{j+1}) = \sqrt{d-1}$ for $1 \leq j \leq \ell - 1$, and $C(X_0, X_1) = C(X_1, X_0) = \sqrt{d}/2$.

If a vector ϕ is an eigenvector of C , then $D\phi$ is an eigenvector of B_Q with the same eigenvalue. By the definition of D this means

$$\psi_C(X_i)^2 = \sum_{u \in X_i} \psi_{A_Q}(u)^2. \quad (22)$$

Define $C_{X_{0:\ell}}$ as the submatrix of C corresponding to the sets $\{X_0, \dots, X_\ell\}$, then extended with zeros to have the same size as C . Every entry of $C + \frac{d/2 - \sqrt{d-1}}{\sqrt{d-1}} C_{X_{0:\ell}}$ is less than or equal to the corresponding entry of the adjacency matrix of a path of length $n + 2 \log_{d-1} n$ with $d/2$ edges between pairs of vertices. Also, ψ_C is a nonnegative vector. Therefore the quadratic form $\psi_C^T (C + \frac{d/2 - \sqrt{d-1}}{\sqrt{d-1}} C_{X_{0:\ell}}) \psi_C$ is at most the spectral radius of this path. Namely

$$\psi_C^T C \psi_C + \frac{d/2 - \sqrt{d-1}}{\sqrt{d-1}} \psi_C^T C_{X_{0:\ell}} \psi_C \leq d \cos(\pi/(n + 2 \log_{d-1} n + 1)).$$

Because C contains the path of length n , $\psi_C^T C \psi_C \geq d \cos(\pi/(n + 1))$. Putting these together yields

$$\psi_C^T C_{X_{0:\ell}} \psi_C \leq \frac{\sqrt{d-1}}{d/2 - \sqrt{d-1}} \cdot d (\cos(\pi/(n + 2 \log_{d-1} n + 1)) - \cos(\pi/(n + 1))) \leq \frac{d \sqrt{d-1}}{d/2 - \sqrt{d-1}} \frac{3\pi^2 \log_d n}{n^3}. \quad (23)$$

Define $\psi_C(X_{1:\ell})$ as the projection of ψ_C on $\{X_1, \dots, X_\ell\}$. $C \psi_C(X_{1:\ell}) = C_{X_{0:\ell}} \psi_C(X_{1:\ell})$, so

$$\psi_C^T C_{X_{0:\ell}} \psi_C \geq \lambda_1 \|\psi_C(X_{1:\ell})\|^2 \geq d (\cos(\pi/(n + 1))) \|\psi_C(X_{1:\ell})\|^2 \quad (24)$$

Combining (23) and (24) yields

$$\|\psi_C(X_{1:\ell})\|^2 \leq \frac{\sqrt{d-1}}{d/2 - \sqrt{d-1}} \left(\frac{3\pi^2 \log_d n}{n^3} \right) / \cos(\pi/n + 1) \leq \left(\frac{21\pi^2 \log_d n}{n^3} \right)$$

assuming $d \geq 4$ and n is sufficiently large.

Using (22) and the eigenvalue equation, we obtain

$$\psi_Q(r_0) = \psi_C(X_0) \leq \lambda_1(A_C) \|\psi_C(X_{1:\ell})\| \leq d \cdot \frac{5\pi \log_d^{1/2} n}{n^{3/2}}.$$

Therefore, according to (21)

$$r_\ell \leq \frac{15d^2 \pi (\log_d^{1/2} n)}{n^{5/2}}.$$

□

6 Open Problems

We conclude with some promising directions for further research.

Beyond the Trace Method: Polynomial Multiplicity Bounds

There is a large gap between our upper bound of $O(n/\log^{1/5} n)$ on the multiplicity of the second eigenvalue and the lower bound of $n^{1/3}$ mentioned after Theorem 1.1. It is very natural to ask, whether the bound of this paper may be improved. To improve the bound beyond $O(n/\text{polylog}(n))$, however, it appears that a very different approach is needed.

Open Problem 1 (Similar to Question 6.3 of [JTY⁺19]). Let $d > 1$ be fixed integer. Does there exist an $\varepsilon > 0$ such that for every connected d -regular graph G on n vertices, the multiplicity of the second largest eigenvalue of A_G is $O(n^{1-\varepsilon})$?

In the present paper, we rely on the trace method to bound eigenvalue multiplicity through closed walks. There are three drawbacks to this approach that stops a bound on the second eigenvalue multiplicity below $n/\text{polylog}(n)$. First, considering walks of length $\omega(\log(n))$ makes the top eigenvalue dominate the trace, leaving no information behind. Second, considering the trace $\text{Tr} A_G^k$ for $k = O(\log(n))$ it is impossible to distinguish eigenvalues that differ by $O(1/\log(n))$. Third, as covered in Section 5.1, there exist graphs such that there are $\Omega(n/\text{polylog}(n))$ eigenvalues in a range of that size around the second eigenvalue. Thus, the trace method reaches a natural barrier at $n/\text{polylog}(n)$.

Eigenvalue Multiplicity for Unnormalized Non-Regular Graphs

Another natural question is whether Theorem 1.2 may be extended to hold for the (non-normalized) adjacency matrix of non-regular graphs.

Open Problem 2. Let $\Delta > 1$ be a fixed integer. Does it hold for every connected graph G on n vertices of maximum degree Δ that the multiplicity of the second largest eigenvalue of A_G is $o(n/\log \log(n))$?

In order to handle unnormalized irregular graphs via the approach in this paper, the key ingredient needed would be an “unnormalized” analogue of Theorem 1.3, showing that a uniformly random closed walk (from the set of all closed walks) in an irregular graph must have large support. We exhibit in Appendix B an irregular “lollipop” graph for which the typical support of a closed walk from a specific vertex is only $O(\text{polylog}(k))$. It remains plausible that when starting from a random vertex, a randomly selected closed walk has $\text{poly}(k)$ support in irregular graphs.

Sharper Bounds for Closed Random Walks

We have no reason to believe that the exponent of $1/5$ appearing in Theorem 1.3 is sharp. In fact, we know of no example where where the answer is $o(k^{1/2})$. An improvement over Theorem 1.3 would immediately yield an improvement of Theorem 1.2.

Open Problem 3. Let $d > 1$ be a fixed integer. Does there exist an $\alpha > 1/5$ such that for every connected d -regular graph G on n vertices and every vertex x of G , a random closed walk of length $2k < n$ rooted at x has support $\Omega(k^\alpha)$ in expectation? Is $\alpha = 1/2$ true? Does such a bound hold for SRW in general?

Acknowledgments

We thank Yufei Zhao for telling us about [JTY⁺19] at the Simons Foundation conference on High Dimensional Expanders in October, 2019. We thank Shirshendu Ganguly for helpful discussions.

References

- [ABS15] Sanjeev Arora, Boaz Barak, and David Steurer. Subexponential algorithms for unique games and related problems. *Journal of the ACM (JACM)*, 62(5):1–25, 2015.
- [BGH⁺15] Boaz Barak, Parikshit Gopalan, Johan Håstad, Raghu Meka, Prasad Raghavendra, and David Steurer. Making the long code shorter. *SIAM Journal on Computing*, 44(5):1287–1324, 2015.
- [Bol13] Béla Bollobás. *Modern graph theory*, volume 184. Springer Science & Business Media, 2013.
- [BRS11] Boaz Barak, Prasad Raghavendra, and David Steurer. Rounding semidefinite programming hierarchies via global correlation. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 472–481. IEEE, 2011.
- [CG07] Sebastian Cioabă and David Gregory. Principal eigenvectors of irregular graphs. *The Electronic Journal of Linear Algebra*, 16, 2007.
- [Cio07] Sebastian M Cioabă. The spectral radius and the maximum degree of irregular graphs. *The Electronic Journal of Combinatorics*, 14(1):R38, 2007.
- [CRS93] Dragoš Cvetković, Peter Rowlinson, and Slobodan Simić. A study of eigenspaces of graphs. *Linear algebra and its applications*, 182:45–66, 1993.
- [CVDKL10] Sebastian M Cioabă, Edwin R Van Dam, Jack H Koolen, and Jae-Ho Lee. A lower bound for the spectral radius of graphs with fixed diameter. *European Journal of Combinatorics*, 31(6):1560–1566, 2010.
- [DS84] Peter G Doyle and J Laurie Snell. *Random walks and electric networks*, volume 22. American Mathematical Soc., 1984.
- [Fri91] Joel Friedman. *Some geometric aspects of graphs and their eigenfunctions*. Princeton University, Department of Computer Science, 1991.
- [Gil98] David Gillman. A chernoff bound for random walks on expander graphs. *SIAM Journal on Computing*, 27(4):1203–1220, 1998.
- [God93] Chris Godsil. *Algebraic combinatorics*, volume 6. CRC Press, 1993.
- [JTY⁺19] Zilin Jiang, Jonathan Tidor, Yuan Yao, Shengtong Zhang, and Yufei Zhao. Equiangular lines with a fixed angle. *arXiv preprint arXiv:1907.12466*, 2019.
- [Kah95] Nabil Kahale. Eigenvalues and expansion of regular graphs. *Journal of the ACM (JACM)*, 42(5):1091–1106, 1995.

- [Kah97] Nabil Kahale. Large deviation bounds for markov chains. *Combinatorics Probability and Computing*, 6(4):465–474, 1997.
- [Kol11] Alexandra Kolla. Spectral algorithms for unique games. *computational complexity*, 20(2):177–206, 2011.
- [LM08] James R Lee and Yury Makarychev. Eigenvalue multiplicity and volume growth. *arXiv preprint arXiv:0806.1745*, 2008.
- [LOG18] Russell Lyons and Shayan Oveis Gharan. Sharp bounds on random walk eigenvalues via spectral embedding. *International Mathematics Research Notices*, 2018(24):7555–7605, 2018.
- [LOGT14] James R Lee, Shayan Oveis Gharan, and Luca Trevisan. Multiway spectral partitioning and higher-order cheeger inequalities. *Journal of the ACM (JACM)*, 61(6):1–30, 2014.
- [LPS88] Alexander Lubotzky, Ralph Phillips, and Peter Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.
- [LRTV12] Anand Louis, Prasad Raghavendra, Prasad Tetali, and Santosh Vempala. Many sparse cuts via higher eigenvalues. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1131–1140, 2012.
- [McK81] Brendan D McKay. The expected eigenvalue distribution of a large regular graph. *Linear Algebra and its Applications*, 40:203–216, 1981.
- [MSS15] Adam W Marcus, Daniel A Spielman, and Nikhil Srivastava. Interlacing families i: Bipartite ramanujan graphs of all degrees. *Annals of Mathematics*, 182:307–325, 2015.
- [OGT13] Shayan Oveis Gharan and Luca Trevisan. A new regularity lemma and faster approximation algorithms for low threshold rank graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 303–316. Springer, 2013.
- [Row90] Peter Rowlinson. More on graph perturbations. *Bulletin of the London Mathematical Society*, 22(3):209–216, 1990.
- [Row19] Peter Rowlinson. Eigenvalue multiplicity in regular graphs. *Discrete Applied Mathematics*, 269:11–17, 2019.
- [Ste14] Dragan Stevanovic. *Spectral radius of graphs*. Academic Press, 2014.
- [TT15] Michael Tait and Josh Tobin. Characterizing graphs of maximum principal ratio. *arXiv preprint arXiv:1511.06378*, 2015.
- [VMSK⁺11] Piet Van Mieghem, Dragan Stevanović, Fernando Kuipers, Cong Li, Ruud Van De Bovenkamp, Daijie Liu, and Huijuan Wang. Decreasing the spectral radius of a graph by link removals. *Physical Review E*, 84(1):016101, 2011.

A Proofs for high degree regular graphs

Theorem A.1 (Detailed Theorem 1.8). *If G is d -regular, has exactly h self-loops at every vertex, and no multi-edges⁴, then*

$$\mathbb{P}_x(W^{2k,s}) \leq \exp\left(-\frac{k}{100s^3}\right) \mathbb{P}_x(W^{2k,2s}) \quad \text{for } s \leq \min\left\{\frac{1}{8}\left(\frac{k}{\log d}\right)^{1/4}, \frac{d-h}{2}\right\}. \quad (25)$$

Proof. We show this via a small modification of the proof of Theorem 3.1. Assume $s \leq (d-h)/2$. The key observation is that each vertex has at least $d-h$ edges in G to other vertices, so in a subgraph of size at most $2s-1$ every vertex has at least one edge in G leaving the subgraph. In this case, we can simply choose $u \in S$ as $u := \arg \max_{w \in S} \psi_S(w)$ in Lemma 3.3. Therefore, considering the adjacency matrix, (15) can be improved to

$$\lambda_1(A_{S \cup \{v\}}) \geq \frac{1}{2} \left(\lambda_1 + \sqrt{\lambda_1^2 + \psi_S(u)^2} \right) \geq \lambda_1 + \frac{\psi_S(u)^2}{6\lambda_1^2} \geq \lambda_1 + \frac{1}{6\lambda_1^2 s}.$$

Therefore, after adding s vertices to S according to the process of Lemma 3.3, we find a set $T \in \Gamma_x^{2s}$ satisfying

$$\lambda_1(A_T) \geq \lambda_1 + \frac{1}{6\lambda_1^2} \sum_{i=1}^s \frac{1}{s+i-1} \geq \lambda_1 + \frac{\log 2}{6\lambda_1^2} \geq \lambda_1 \left(1 + \frac{1}{10\lambda_1^3}\right).$$

Using this improved bound, and keeping in mind that $\lambda_1(A_T) \leq 2s$, we can replicate the argument above to get to the following improvement over (18):

$$\mathbb{P}_x(W^{2k,s}) \leq \exp\left(2s \log d + 4s \log(2s) + \log(2s) - \frac{k}{80s^3}\right) \mathbb{P}_x(W^{2k,2s}).$$

This implies

$$\mathbb{P}_x(W^{2k,s}) \leq \exp\left(7s \log d - \frac{k}{80s^3}\right) \mathbb{P}_x(W^{2k,2s}) \leq \exp\left(-\frac{k}{100s^3}\right) \mathbb{P}_x(W_x^{2k,2s})$$

whenever

$$s \leq \frac{1}{8} \left(\frac{k}{\log d} \right)^{1/4},$$

establishing (25). □

Theorem A.2 (Detailed Theorem 1.7). *If G is simple and d -regular, then*

$$m_G \left(\left[\left(1 - \frac{\log \log_d n}{\log_d n}\right) \lambda_2, \lambda_2 \right] \right) = \begin{cases} O\left(n \cdot \frac{\log d \log \log n}{d}\right) & \text{when } d \log^{1/2} d \leq \alpha \log^{1/4} n \\ O\left(n \cdot \frac{\log^{1/2} d \log \log n}{\log^{1/4} n}\right) & \text{when } d \log^{1/2} d \geq \alpha \log^{1/4} n \end{cases} \quad (26)$$

for all⁵ $d \leq \exp(\sqrt{\log n})$, where $\alpha := \sqrt[4]{3}/4$.

⁴This technical assumption is used to handle the case when $|\lambda_n(A_G)| > \lambda_2(A_G)$ in Theorem A.2. Here we take $h = 0$.

⁵If $d \geq \exp(\sqrt{\log n})$ then (26) is vacuously true.

Proof. The proof is the same as the proof of Theorem 1.2 in Section 4, except we choose different s .

1. If $d \log^{1/2} d < \alpha \log^{1/4} n$ set

$$s := \min \left\{ \frac{1}{8} \left(\frac{k}{\log d} \right)^{1/4}, \frac{d-h}{2} \right\} = \frac{d}{2}$$

with $h = 0$. Applying Theorem A.1 it is easily checked that (20) is satisfied for large enough n , yielding a bound of

$$m_G \left(\left[\left(1 - \frac{\log \log_d n}{\log_d n} \right) \lambda_2, \lambda_2 \right] \right) = O \left(n \cdot \frac{\log d \log \log n}{d} \right).$$

2. If G is simple, d -regular and $d \log^{1/2} d \geq \alpha \log^{1/4} n$, set

$$s := \min \left\{ \frac{1}{8} \left(\frac{k}{\log d} \right)^{1/4}, \frac{d-h}{2} \right\} = \frac{1}{8} \left(\frac{\log n}{\log^2 d} \right)^{1/4}$$

with $h = 0$. Then (20) is again satisfied by applying Theorem 3.1 equation (25), and we conclude that

$$m_G \left(\left[\left(1 - \frac{\log \log_d n}{\log_d n} \right) \lambda_2, \lambda_2 \right] \right) = O \left(n \cdot \frac{\log^{1/2} d \log \log n}{\log^{1/4} n} \right).$$

□

B Lollipop

Here, we show that if we do not assume that our graph is regular, the average support of a uniformly chosen (from the set of all such walks) closed walk of length k from a fixed vertex is no longer necessarily $k^{\Theta(1)}$ (as opposed to the average support of a random walk). We take the lollipop graph, which consists of a clique of $(d+1)$ vertices for fixed $d \geq 3$ and a path of length n $\{u_1, \dots, u_n\}$ attached to a vertex v of the clique, where $n \gg k$. Here $\psi := \psi(A)$ and $\lambda_1 := \lambda_1(A)$ are the Perron eigenvector and eigenvalue of the adjacency matrix of the graph.

Lemma B.1. $\psi(v) \geq 1/\sqrt{d+2}$.

Proof. By symmetry, the value on all entries of the clique besides v are the same. Call this value $\psi(b)$. Then by the eigenvalue equation we have $\lambda_1 \psi(b) = \psi(v) + (d-1)\psi(b)$, so as $\lambda_1 \geq d$, it must be that $\psi(v) \geq \psi(b)$.

Similarly, to satisfy the eigenvalue equation, vertices on the path must satisfy the recursive relation

$$\begin{aligned} \lambda_1 \psi(u_i) &= \psi(u_{i-1}) + \psi(u_{i+1}) \quad 1 \leq i \leq n-1 \\ \lambda_1 \psi(u_n) &= \psi(u_{n-1}) \end{aligned}$$

where we define $v = u_0$. To satisfy this equation, we must have $\psi(u_i) \geq (\lambda_1 - 1)\psi(u_{i+1})$ for each i , so as $\lambda_1 \geq d \geq 3$, $\psi(v) \geq \sum_{i=1}^n \psi(u_k)$. As the Perron vector is nonnegative, $\psi(v)^2 \geq \sum_{i=1}^n \psi(u_k)^2$, and

$$(d+2)\psi(v)^2 \geq \psi(v)^2 + d\psi(b)^2 + \sum_{i=1}^n \psi(u_k)^2 = 1,$$

so $\psi(v) \geq 1/\sqrt{d+2}$.

□

Call γ_v^{2k} the number of closed walks of length $2k$, and $\gamma_v^{2k, \geq \ell+d+1}$ as the subset of these walks with support at least $\ell + d + 1$.

Proposition B.2. For $\ell \geq 2 \log(k) / \log(\lambda_1/2)$,

$$|\gamma_v^{2k, \geq \ell+d+1}| = O(k^{-2}) |\gamma_v^{2k}|.$$

Proof. For a closed walk to have support $\ell + d + 1$, it must contain u_ℓ . For such walks, once the path is entered, at least 2ℓ steps must be spent in the path, as the walk must reach u_ℓ and return. Therefore, closed walks starting at v that reach u_ℓ can be categorized as follows. First, there is a closed walk from v to v . Then there is a closed walk from v to v going down the path containing u_ℓ . On this excursion, the walk can only go forward or backwards, and it spends at least 2ℓ steps within the path. For each of these steps, there are 2 options. If we remain in the path after 2ℓ steps, upper bound the number of choices until returning to v by λ_1 at each step. After returning to v , the remaining steps form another closed walk. The number of closed walks from v of length i is at most λ_1^i . Therefore the number of closed walks with an excursion to u_ℓ is at most

$$\sum_{i=0}^{2k} \lambda_1^i 2^{2\ell} \lambda_1^{2k-2\ell-i} = (2k+1) \lambda_1^{2k-2\ell} 2^{2\ell}.$$

The total number of closed walks starting at v is at least $\psi(v)^2 \lambda_1^{2k}$. Therefore the fraction of closed walks that have support at least ℓ is at most

$$\frac{(2k+1) 2^{2\ell} \lambda_1^{2k-2\ell}}{\lambda_1^{2k} / (d+2)} = \frac{(d+2)(2k+1) 2^{2\ell}}{\lambda_1^{2\ell}}$$

so for $\ell \geq 2(\log k) / \log(\lambda_1/2)$, this is $O(k^{-2})$. □

Remark B.3. Instead of adding a path, we can add a tree (as exhibited in Figure 1). According to the same analysis, the probability a walk reaches depth further than $\Theta(\log k)$ is small. Therefore, in Theorem 1.3 we can not get a sufficient bound on support from passing to depth, but must deal with support itself.

Appendix E

Optimal Decremental Connectivity in Non-Sparse Graphs

Optimal Decremental Connectivity in Non-Sparse Graphs

Anders Aamand
University of Copenhagen
aa@di.ku.dk

Adam Karczmarz
University of Warsaw
a.karczmarz@mimuw.edu.pl

Jakub Łacki
Google Research
jłacki@google.com

Nikos Parotsidis
Google Research
nikosp@google.com

Peter M. R. Rasmussen
University of Copenhagen
pmrr@di.ku.dk

Mikkel Thorup
University of Copenhagen
mikkel2thorup@gmail.com

September 2, 2021

Abstract

We present a dynamic algorithm for maintaining the connected and 2-edge-connected components in an undirected graph subject to edge deletions. The algorithm is Monte-Carlo randomized and processes any sequence of edge deletions in $O(m + n \text{ poly } \log n)$ total time. Interspersed with the deletions, it can answer queries to whether any two given vertices currently belong to the same (2-edge-)connected component in constant time. Our result is based on a general Monte-Carlo randomized reduction from decremental c -edge-connectivity to a variant of fully-dynamic c -edge-connectivity on a sparse graph.

For non-sparse graphs with $\Omega(n \text{ poly } \log n)$ edges, our connectivity and 2-edge-connectivity algorithms handle all deletions in optimal linear total time, using existing algorithms for the respective fully-dynamic problems. This improves upon an $O(m \log(n^2/m) + n \text{ poly } \log n)$ -time algorithm of Thorup [J.Alg. 1999], which runs in linear time only for graphs with $\Omega(n^2)$ edges.

Our constant amortized cost for edge deletions in decremental connectivity in non-sparse graphs should be contrasted with an $\Omega(\log n / \log \log n)$ worst-case time lower bound in the decremental setting [Alstrup et al. FOCS'98] as well as an $\Omega(\log n)$ amortized time lower-bound in the fully-dynamic setting [Patrascu and Demaine STOC'04].

1 Introduction

In this paper, we present Monte Carlo randomized decremental dynamic algorithms for maintaining the connected and 2-edge-connected components in an undirected graph subject to edge deletions. Starting from a graph with n nodes and m edges, the algorithm can process any sequence of edge deletions in $O(m + n \text{polylog } n)$ total time while answering queries whether a pair of vertices is currently in the same (2-edge-)connected component. Each query is answered in constant time. The algorithm for decremental 2-edge-connectivity additionally reports all bridges as they appear.

For some concrete constant $\alpha \leq 7$, our decremental algorithms thus use $O(m)$ total time on the edge deletions from a graph with $m \geq n \log^\alpha n$ edges, and we will refer to such graphs as *non-sparse*. If we delete all the edges, we support edge deletions in constant amortized time. As we shall discuss in Section 1.1, it is not possible to obtain a constant worst-case time bound for individual edge deletions in this decremental setting, nor is it possible to obtain a constant amortized time bound for edge updates in the fully-dynamic version of the connectivity problem.

Our algorithms are Monte Carlo randomized and answer all queries correctly with high probability¹. We note that since the correct answer to each query is uniquely determined from the input, the algorithms work against adaptive adversaries, that is, each deleted edge may depend on previous answers to queries.

Furthermore, our algorithms offer a *self-check* capability. At the end, after all updates and queries have been processed online, each algorithm can deterministically check if it might have made a mistake. If the self-check passes, it is *guaranteed* that no incorrect answer was given. Otherwise, the algorithm *may have* made a mistake. However, as we show in the following, the self-check passes with high probability. This feature implies that we can obtain Las Vegas algorithms for certain *non-dynamic* problems whose solutions employ decremental (2-edge-)connectivity algorithms as subroutines: we simply repeat trying to solve the static problem from scratch, each time with new random bits, until the final self-check is passed. With high probability, we are done already after the first round. A nice concrete example is the algorithm of Gabow et al. [15] for the static problem of deciding if a graph has a unique perfect matching. The algorithm uses a decremental 2-edge-connectivity algorithm as a subroutine. With our decremental 2-edge-connectivity algorithm, repeating until the self-check is passed, we obtain a Las Vegas algorithm for the unique perfect matching problem that is always correct, and which terminates in $O(m + n \text{polylog } n)$ time with high probability.

The tradition of looking for linear time algorithms for non-sparse graphs goes back at least to Fibonacci heaps, which can be used for solving single source shortest paths in $O(m + n \log n)$ time [14]. Our results show that another fundamental graph problem can be solved in linear time in the non-sparse case.

The previous best time bounds for the decremental connectivity and 2-edge-connectivity problems were provided by Thorup [36]. His algorithms run in $O(m \log(n^2/m) + n \text{polylog } n)$ total time. This is $O(m)$ only for dense graphs with $\Omega(n^2)$ edges. It should be noted that [36] used Las Vegas randomization, that is, correctness was guaranteed, but the running time bound only held with high probability. Our algorithms are Monte Carlo randomized, but offer the final self-check. Another difference is that our new algorithms need only a polylogarithmic number of random bits, whereas the ones from [36] used $\Theta(m)$ random bits.

Both our algorithm and the previous one by Thorup are based on a general reduction from decremental c -edge-connectivity to fully-dynamic c -edge-connectivity on a sparse graph with $\tilde{O}(cn)$ updates. The reductions have a polylogarithmic cost per node as well as a cost per original edge.

¹We define high probability as probability $1 - O(n^{-\gamma})$ for any given γ .

Our contribution is to reduce the edge cost from $O(\log(n^2/m))$ to the optimal $O(1)$. The general reduction and its consequences will be discussed in Section 1.2.

We will now give a more detailed discussion of our results in the context of related work.

1.1 Connectivity

Dynamic connectivity is the most fundamental dynamic graph problem. The fully dynamic version has been extensively studied [8, 9, 12, 20, 21, 24, 26, 29, 32, 33, 34, 37, 40, 41] from both the lower and upper bound perspective, even though close to optimal amortized update bounds have been known since the 90s [20, 21, 37]. Currently, the best known amortized update time bounds are (expected) $O(\log n \cdot (\log \log n)^2)$ using randomization [24] and $O(\log^2 n / \log \log n)$ deterministically [41]. Both these results have optimal (wrt. to the lower bounds) query time.

Connectivity Lower Bounds Our result implies that decremental connectivity is provably easier than fully-dynamic connectivity for a wide range of graph densities. Specifically, let t_u be the update time of a fully dynamic connectivity algorithm and let t_q be its query time. Pătraşcu and Demaine [33] showed a lower bound of $\Omega(\log n)$ on $\max(t_u, t_q)$ in the cell-probe model. Pătraşcu and Thorup [34] also showed that $t_u = o(\log n)$ implies $t_q = \Omega(n^{1-o(1)})$. These lower bounds hold for all graph densities and allow for both amortization and randomization. As a result, no fully-dynamic connectivity algorithm can answer connectivity queries in constant time and have an amortized update time of $o(\log n)$.

In sharp contrast, assuming that $m = \Omega(n \text{ poly } \log n)$ edges are deleted, our algorithm shows that one can solve decremental connectivity handling both queries and updates in constant amortized time.

We note that such a result is possible only because we allow for amortization, as any decremental connectivity algorithm with *worst-case* update time $O(\text{poly } \log n)$ must have worst-case query time $\Omega\left(\frac{\log n}{\log \log n}\right)$ [3]. This lower bound holds even for trees supporting restricted connectivity queries of the form "are u and v connected?" for a fixed "root" u . This lower bound also holds for dense graphs, as we can always add a large static clique to the problem.

An optimal incremental connectivity algorithm has been known for over 40 years. Namely, to handle $m \geq n$ edge insertion and q connectivity queries, one can use the union-find data structure [35] with $n - 1$ unions and $2(m + q)$ finds. The total running time is $\Theta((m + q)\alpha((m + q), n))$, which is linear for all but very sparse graphs (since $\alpha(\Omega(n \log n), n) = O(1)$). It was later shown that this running time is optimal for incremental connectivity [13].

Similarly to the decremental case, one cannot hope to obtain an analogous result with a worst-case update time in the incremental setting: Pătraşcu and Thorup [34] showed that any incremental connectivity data structure with $o\left(\frac{\log n}{\log \log n}\right)$ worst-case update time must have worst-case $\Omega(n^{1-o(1)})$ query time in the cell-probe model.

Other cases of optimal decremental connectivity There is much previous work on cases where decremental connectivity can be supported in $O(m)$ total time. Alstrup et al. [4] showed that decremental connectivity can be solved in optimal $O(m)$ total time on forests, answering queries in $O(1)$ time. This was later extended to other classes of sparse graphs: planar graphs [30], and minor-free graphs [22]. All these special graph classes are sparse with $m = O(n)$ edges.

For general graphs, we only have the previously mentioned work by Thorup [36], yielding a total running time of $O(m)$ for very dense graphs with $m = \Omega(n^2)$ edges. We now obtain the same linear time bound for all non-sparse graphs with $m = \Omega(n \text{ poly } \log n)$ edges.

1.2 General reduction for c -edge-connectivity

Our algorithm for decremental connectivity is based on a general randomized reduction from decremental c -edge-connectivity (assuming all m edges are deleted) to fully-dynamic c -edge-connectivity on a sparse graph with $\tilde{O}(cn)$ updates. The reduction has a polylogarithmic cost per node as well as a constant cost per edge. The previous decremental connectivity algorithm of Thorup [36] was also based on such a general reduction, but the cost per edge was $O(\log(n^2/m))$ which is $O(1)$ only for very dense graphs with $m = \Omega(n^2)$. Below we will describe the format of the reductions in more detail.

Because there are different notions of c -edge-connectivity, we first need to clarify our definitions. We say that two vertices u, v are c -edge-connected iff there exist c edge-disjoint paths between u and v in G . It is known that c -edge-connectivity is an equivalence relation; we call its classes the *c -edge-connected classes*. However, for $c \geq 3$, a c -edge-connected class may induce a subgraph of G which is not connected, so it also makes sense to consider *c -edge-connected components*, i.e., the maximal c -edge-connected induced subgraphs of G .² It is important to note that the c -edge-connected components and the c -edge-connected classes are *uniquely defined* and both induce a natural partition of the vertices of the underlying graph. Moreover, each c -edge-connected component of G is a subset of some c -edge-connected class of G . For $c = 1, 2$, the c -edge-connected classes are c -edge-connected, so the two notions coincide. To illustrate the difference, let us fix $c \geq 3$ and consider a graph with $c + 2$ vertices $v_s, v_t, v_1, \dots, v_c$ and edges $\{v_s, v_t\} \times \{v_1, \dots, v_c\}$; while all c -edge-connected components in this graph are singletons, there is one c -edge-connected class, which is not a singleton, namely $\{v_s, v_t\}$.

We define a *c -certificate* of G to be a subgraph H of G that contains all edges not in c -edge-connected components, and a c -edge-connected subgraph of each c -edge-connected component. Both Thorup's and our reduction maintains a c -certificate H of G . Then, for any $c' \leq c$, we have that the c' -edge-connected equivalence classes and the c' -edge-connected components are the same in G and H . As the edges from G are deleted, we maintain a c -certificate with $\tilde{O}(cn)$ edges undergoing only $\tilde{O}(cn)$ edge insertions and deletions in total.

The uniquely defined c -edge-connected components of a graph may be found by repeatedly removing cuts of size at most $c - 1$. For the reductions, we need algorithms that can help us in this process. We therefore define the *fully dynamic c -edge-cut problem* as follows. Suppose a graph G is subject to edge insertions and/or deletions. Then, a fully dynamic c -edge-cut data structure should maintain any edge e that belongs to some cut of size less than c . A typical application of such a data structure is to repeatedly remove such edges e belonging to cuts of size less than c , which splits G into its c -edge-connected components. For each $c \geq 1$, denote by $T_c(n)$ the amortized time needed by the data structure to find an edge belonging to a cut of size less than c . For example, for $c = 1$ we have $T_1(n) = O(1)$ since we do not have to maintain anything. For $c = 2$, the data structure is required to maintain some *bridge* of G and it is known that $T_2(n) = O((\log n \cdot \log \log n)^2)$ [23]. For $c \geq 3$, in turn, we have $T_c(n) = O(n^{1/2} \text{poly}(c))$ [38].

Given a fully dynamic c -edge-cut data structure, whose update time for a graph on n nodes is $T_c(n)$, Thorup's [36] reduction maintains, in $O(m \log(n^2/m)) + \tilde{O}(c \cdot n \cdot T_c(n))$ total time, a c -certificate H of the decremental graph G starting with n nodes and m edges. The certificate undergoes only $\tilde{O}(cn)$ edge insertions and deletions throughout any sequence of deletions issued to G . We reduce here the total time to $O(m) + \tilde{O}(c \cdot n \cdot T_c(n))$.

Combining our reduction with the polylogarithmic fully-dynamic connectivity and

²There is no consensus in the literature on the terminology relating to c -edge-connected components and classes. Some authors (e.g., [16, 17]) reserve the term c -edge-connected components for what we in this paper call c -edge-connected classes.

2-edge-connectivity algorithm of Holm et al. [21], we can now solve decremental connectivity and 2-edge-connectivity in $O(m) + \tilde{O}(n)$ time.

We can also apply the fully dynamic min-cut algorithm of Thorup [38] which identifies cuts of size $n^{o(1)}$ in $n^{1/2+o(1)}$ worst-case time. For $c = n^{o(1)}$, we then maintain a c -certificate H in $O(m + n^{3/2+o(1)})$ total time. This includes telling which vertices are in the same c -edge-connected component. If we further want to answer queries about c -edge-connectivity between pairs of nodes, we can apply the fully-dynamic data structure of Jin and Sun [25] to the c -certificate H . By definition, the answers to these queries are the same in H and G , and the algorithm takes $n^{o(1)}$ time per update or query. Hence the total time for the updates remains $O(m + n^{3/2+o(1)})$, and we can tell if two vertices are c -edge-connected in $n^{o(1)}$ time.

1.2.1 Results

We will now give a more precise description of our reduction, including the log-factors hidden in the $\tilde{O}(cn)$ bound. Let the *decremental c -certificate* problem be that of maintaining a c -certificate of G when G is subject to edge deletions. Recall that $T_c(n)$ denotes the amortized update time of a fully-dynamic c -edge-cut data structure. Thorup [36] showed the following.

Theorem 1.1 (Thorup [36]). *There exists a Las Vegas randomized algorithm for the decremental c -certificate problem with expected total update time $O(m \log(n^2/m) + n(c + \log n) \cdot T_c(n) \log^2 n)$. The maintained certificate undergoes $O(n \cdot (c + \log n))$ expected edge insertions and deletions throughout, assuming $\Theta(m)$ random bits are provided. These bounds similarly hold with high probability.*

In particular the total update time is $O(m)$ for very dense graphs with $\Omega(n^2)$ vertices. Our main result, which we state below, shows that an amortized constant update time can be obtained as long as the initial graph has $\Omega(n \text{ polylog}(n))$ edges.

Theorem 1.2. *There exists a Monte Carlo randomized algorithm for the decremental c -certificate problem with total update time $O(m + n(c + \log n) \cdot T_c(n) \log^3 n + nc \log^7 n)$. The maintained certificate undergoes $O(nc \log^4 n)$ edge insertions and deletions throughout. The algorithm is correct with high probability. Within this time bound, the algorithm offers a final self-check after processing all updates.*

In fact, our algorithm is itself a reduction to $O(\log n)$ instances of the decremental c -certificate problem on a subgraph of G with $O(m/\log^2 n)$ edges. To handle these instances, we use the state-of-the-art data structure (Theorem 1.1) which costs only $O(m)$ in terms of m . As a result, our improved reduction (Theorem 1.2) requires $\Theta(m)$ random bits to hold.

However, our new randomized c -certificate that is the key to obtaining the new reduction requires only pairwise independent sampling to work. This is in sharp contrast with the certificate of Karger [28], used in the construction of Thorup's data structure (Theorem 1.1), which requires full independence, i.e., $\Theta(m)$ random bits. We show that we may instead plug our new certificate into Thorup's data structure at the cost of a single additional logarithmic factor in the running time. Since Karger's certificate constitutes the only use of randomness in Thorup's data structure, and full independence in our construction is required only for invoking Theorem 1.1, we obtain the below low-randomness version of our main result.

Theorem 1.3. *There exists a Monte Carlo randomized algorithm for the decremental c -certificate problem with total update time $O(m + nc \cdot T_c(n) \log^4 n + nc \log^7 n)$. The maintained certificate undergoes $O(nc \log^4 n)$ edge insertions and deletions throughout. The algorithm is correct with high probability if $O(\text{polylog } n)$ random bits are provided. Within this time bound, the algorithm offers the final self-check after processing all updates.*

By using Theorem 1.3 with best known fully dynamic algorithms for different values of c [21, 25, 38], we obtain:

Theorem 1.4. *There exists Monte Carlo randomized decremental connectivity and decremental 2-edge-connectivity algorithms with $O(m + n \log^7 n)$ total update time and $O(1)$ query time.*

Theorem 1.5. *Let $c = (\log n)^{o(1)}$. There exists a Monte Carlo randomized decremental c -edge-connectivity data structure which can answer queries to whether two vertices are in the same c -edge connected class in $O(n^{o(1)})$ time, and which has $O(m) + \tilde{O}(n^{3/2})$ total update time.*

Theorem 1.6. *Let $c = O(n^{o(1)})$. There exists a Monte Carlo randomized decremental c -edge-connected components data structure with $O(m + n^{3/2+o(1)})$ total update time and $O(1)$ query time.*

All the above applications of our main result work using only $O(\text{polylog } n)$ random bits. They moreover each have the self-check property as well. As discussed before, our new 2-edge-connectivity data structure implies an optimal $O(m)$ -time unique perfect matching algorithm for $m = \Omega(n \text{ polylog } n)$.

1.3 Adaptive updates and unique perfect matching

All our time bounds are amortized. Amortized time bounds are particularly relevant for dynamic data structures used inside algorithms solving problems for static graphs. In such contexts, future updates often depend on answers to previous queries, and therefore we need algorithms that work with adaptive updates.

Our reduction works against adaptive updates as long as queries do not reveal any details about the c -certificate H . The reduction will safely maintain the following public information about the c -edge-connected components of G : between deletions, each such component will have an ID stored with all its vertices, so two vertices are in the same c -edge-connected component if and only if they have the same component ID. With the component ID, we store its size and a list with its vertices in order of increasing vertex ID. Finally, we can have a list of all edges that are not in c -edge-connected components. After each update, we can also tell what are the IDs of the new c -edge-connected components, and what are the edges between them.

For the case of 2-edge-connectivity, the above means that we can maintain the bridges of a decremental graph and we can also maintain the connected components and their sizes without revealing what the current randomized certificate looks like. All this is needed for the unique perfect matching algorithm of Gabow et al. [15]. The algorithm is an extremely simple recursion based on the fact that a graph with a unique perfect matching has a bridge and all components have even sizes. The algorithm first asks for a bridge (u, v) of some component. If there is none, there is no unique matching. Otherwise we remove (u, v) and check the sizes of the components of u and v . If they are odd, (u, v) is in the unique matching, and we remove all other incident edges. Otherwise (u, v) is not in the unique matching. The important thing here is that the bridges do not tell us anything about our 2-certificate of the 2-edge-connected components.

Thus we solve the static problem of deciding if a graph has a unique perfect matching in $O(m) + \tilde{O}(n)$ time. If the self-verification reports a possible mistake, we simply rerun. Thus we get a Las Vegas algorithm that terminates in $O(m) + \tilde{O}(n)$ time with high probability.

1.4 Techniques

Our main technical contribution is a new construction of a sparse randomized c -certificate that witnesses the c -edge-connected components of G and can be maintained under edge deletions in G .

In the static case, deterministic certificates of this kind have been known for decades [31]. However, they are not very robust in the decremental setting, where an adversary can constantly remove its edges forcing it to update frequently. Consequently, Thorup [36] used a randomized sample-based certificate to obtain his reduction. The general idea behind this approach is to ensure that the certificate is sparse and undergoes few updates. Ideally, the sparse certificate will only have to be updated whenever an edge from the certificate is deleted. Using a fully dynamic data structure on the certificate, we may obtain efficient algorithms provided that we don't spend too much time on maintaining the certificate. Thorup's reduction had an additive overhead $O(m \log(n^2/m))$ for maintaining the certificate, which we will reduce to $O(m)$. We shall, in fact, use Thorup's reduction as a subroutine, called on $O(\log n)$ decremental subproblems each starting with $O(m/\log^2 n)$ edges.

1.4.1 Thorup's construction [36]

Let us first briefly describe how Thorup's algorithm operates on certificates and highlight difficulties in improving his reduction to linear time. First of all, the c -certificate is constructed as follows: initially, sample edges of G uniformly with probability $P \leq 1/2$, thus obtaining a subgraph S . Then, compute the c -edge-connected components of S and form a certificate H out of two parts: (1) A *recursive* certificate of S , and (2) the subgraph D consisting of edges of G connecting distinct c -edge-connected components of S .

As proved by Karger [28], D has size $\tilde{O}(cn/P)$ with high probability. Thorup [36] generalizes this by proving that D undergoes only $\tilde{O}(cn/P)$ insertions throughout any sequence of edge deletions to S . Since D depends only on the c -edge-connected components of S , it is enough to have a c -certificate of S in order to define D . Hence, a c -certificate of S (which is a graph a size $O(mP)$, i.e., a constant factor smaller) is maintained under edge deletions recursively. The recursion stops when the size of the input graph is $O(cn)$. To maintain D at each recursive level, we first need to maintain the c -edge-connected components of the (recursive) certificate of S under edge deletions. The certificate of S can be (inductively) seen to have $\tilde{O}(cn/P)$ edges and undergo $\tilde{O}(cn/P)$ updates. As a result, for $P = 1/2$ maintaining its c -edge-connected components costs $\tilde{O}(cn \cdot T_c(n))$ total time using the fully-dynamic c -edge-cut data structure. Since at each recursion level the certificate size decreases geometrically, the expected cost of all the dynamic c -edge-cut data structures is $\tilde{O}(cn \cdot T_c(n))$.

The additional cost of $O(m \log(n^2/m))$ comes from the fact that, at each level of the recursion, when a c -edge-connected component in S splits into two components as a result of an edge deletion, we need to find edges of G between these two components in order to update D . This takes $O(m \log(n^2/m))$ total time throughout using a standard technique of iterating through the edges incident to the nodes in the smaller component every time a split happens [11]. The $O(\log(n^2/m))$ (instead of $O(\log(n))$) cost comes by noticing that a vertex can at most have q neighbors in a component of order q , and that after we go through the edges of a vertex i times it is in a component of order $\leq n/2^i$; hence it is only the first $O(\log(n/\deg(v)))$ times that all neighbors of v have to be considered, so the total time spent on this becomes $O(\sum_{v \in V} \deg(v) \log(n/\deg(v))) = O(m \log(n^2/m))$.

It turns out very challenging to get rid of the $O(m \log(n^2/m))$ term associated with finding cuts when components split in Thorup's reduction. If we knew that all of these cuts were *small*, say of size at most δ , then we could apply a whole bag of tricks for efficiently finding them in a total time of $\tilde{O}(n\delta)$, e.g., using invertible Bloom lookup tables [19], or the XOR-trick [1, 2, 27]. Unfortunately, the bound of $\tilde{O}(cn/P)$ only gives an average bound on the number of edges between pairs of components, and in fact there can be pairs of components having as many as $\Omega(n^{1/3})$ edges between them, as we will later show. In order to resolve this, we have to introduce a new type of sample based c -edge certificate obtained by only removing cuts of size at most $\delta = O(c \text{polylog } n)$ from G . In the following three subsections, we describe the ideas behind this new certificate, the

technical challenges encountered in efficiently maintaining it, and why such a certificate is relevant for decremental connectivity algorithms.

1.4.2 A small cut sample certificate

On the highest level, our c -edge certificate of a graph $G = (V, E)$ is constructed starting with a sample $S \subset G$. For now we assume that the edges of S are sampled independently with some probability $P \leq 1/2$, but we will later see how to reduce the number of random bits needed for the sampling to $O(\text{polylog } n)$. In our algorithms, G will be the current decremental graph and the sample S will be made from the original graph. Thus, when G has undergone a sequence of deletions, the current sample will be $S \cap G$. At any point in time, the maintained c -edge certificate only depends on the sample S and the current graph G , not on the sequence of edge updates made to G so far. We may therefore describe the c -edge certificate statically.

The critical idea behind our certificate is to introduce a small-cut-parameter δ . Our certificate is obtained by iteratively removing certain cuts from G where each cut is allowed to be of size at most δ . We denote by $D \subset G$ the graph whose edge set consists of the edges removed in this process. The overall goal is to define this cut removal process in a way so that (1) each connected component of $G \setminus D$ is c -edge-connected in S , and (2) it is easy to detect new small cuts under edge deletions issued to G . We then use $S \cup D$ as our c -edge connectivity certificate of G . Importantly, we want δ to be *as small as possible*, ideally $\delta = O(c \text{ polylog}(n))$. This is because $\tilde{O}(\delta n)$ will show up as an additive cost in our algorithm for maintaining the certificate. We will describe shortly how this type of certificate can be used in the design of efficient decremental c -edge-connectivity algorithms, but let us first demonstrate that the existence of such a cut removal process for a small δ is non-trivial.

First of all, we could simply remove *all* cuts from G of size at most δ leaving us with the $(\delta + 1)$ -connected components. Karger's result [28] implies that with $\delta = O((c + \log n)/P)$ sufficiently large, these components will remain c -edge connected in S . However, in order to maintain the small cuts, we would need a decremental δ -edge connectivity algorithm. As $\delta > c$, this approach simply reduces our problem to a much harder one.

Suppose on the other hand that we attempted to use Thorup's sampling certificate [36] described above. To simplify the exhibition, let's assume that $P = 1/2$ and $c = 1$. If D is the set of edges between connected components of S , $D \cup S$ is a certificate. Thorup's algorithm recurses on S to find a final certificate of G . At first sight it may seem like D can be constructed by iteratively removing cuts of size at most $\delta = O(\log n)$ between the connected components of S . After all, isn't it unlikely that a connected component of S has more than, say, $100 \log n$ unsampled outgoing edges when the sampling probability is $P = 1/2$? As tempting as this logic may be, it is flawed. To illustrate this, let $G = G(n, 2/n)$ be the Erdős–Rényi graph obtained by sampling each edge of the complete graph on n vertices independently with probability $2/n$. Let further S be the subgraph of G obtained by sampling each edge with probability $1/2$. Then S is distributed as the Erdős–Rényi graph $G(n, 1/n)$ and basic phase transition results [10] show that the two largest components of S , C_1 and C_2 , almost surely have size $\Theta(n^{2/3})$. Now, we can conversely construct G from S by including each non-sampled edge of the complete graph with probability $\frac{1}{n-1}$, and then we expect G to contain as many as $\Theta(n^{1/3})$ edges between C_1 and C_2 . At some point in the iterative process, we are thus forced to remove a cut of size $\Omega(n^{1/3})$ splitting C_1 and C_2 , and we would have to choose δ of at least this size (but it is possible that other examples could show that δ would have to be even larger). Our algorithms spend total time $\tilde{O}(n\delta)$ on finding these cuts, and if $\delta = \Omega(n^{1/3})$, this is not good enough for a linear time algorithm for non-sparse graphs. We remark that in this example, each vertex of G has degree $O(\log n)$ with high probability. Therefore, an alternative approach yielding cuts of size $O(\log n)$ would be to cut out one vertex at a time moving all incident edges to

D . In particular this would cut the large sampled components C_1 and C_2 into singletons, one vertex at a time. Clearly, we cannot proceed like this for general graphs which may have many vertices of large degree. Nevertheless, this simple idea will be critically used in our construction.

Our actual certificate uses $\delta = O(\frac{c \log n}{P})$. The certificate can be constructed for any $P \leq 1/2$, but in our applications, $P = 1/\text{polylog } n$. We proceed to sketch the construction now and refer the reader to Section 4 for the precise details. On a high level, the sample S is partitioned into $\ell = O(\log n)$ samples S_1, \dots, S_ℓ , each having size approximately Pm/ℓ . To construct our certificate, we start by iteratively pruning G of the edges incident to vertices of degree less than δ , moving these edges to D . The graph left after the pruning $G_1 = G \setminus D$ satisfies that each vertex of positive degree has degree at least δ . Next, S_1 defines a sample of G_1 , $H_1 = S_1 \cap G_1$ and the bound on the minimum positive degree in G_1 guarantees that with constant probability a fraction of $3/4$ of the vertices with positive degree in the sampled subgraph H_1 has degree $\geq 4c$. We prove a combinatorial lemma stating that such a graph can have at most $5n/6$ c -edge-connected components. As a result, if we contract the c -edge-connected components of H_1 in the pruned graph G_1 , the resulting graph G'_1 would have at most $5n/6$ vertices. Finally, we construct a c -certificate for G'_1 recursively using the samples S_2, S_3, \dots , stopping when the contracted graph has no edges between the contracted vertices (here G played the role of G'_0). The constant factor decay in the number of components ensures that we are done after $\ell = O(\log n)$ steps with high probability. All edges of D are obtained as the removed edges of cuts of G of size less than δ , so D will have size $O(n\delta)$. Our certificate will simply be $S \cup D$ which we prove is in fact a c -certificate.

With this, we have thus completed the goal of obtaining a small cut sample certificate with δ as small as $O(\frac{c \log n}{P})$. Abstractly, our certificate has a quite simple description: we alternate between sampling, removing small cuts around c -edge-connected components in the sample, and finally contracting these components. However, in our implementation, we cannot afford to perform the contractions as described above explicitly. This makes it difficult to efficiently find the edges leaving the c -edge connected components of H (at a given recursive level), and we must be able to do this since these c -edge-connected components may split as edges are deleted from G . It turns out that since we are only concerned with cuts of size at most δ , we can in fact identify these cuts in total time $O(m) + \tilde{O}(\delta n)$. We will describe this in the following section.

A final property of our new randomized decremental c -certificate algorithm is that it requires only $O(\log^2 n)$ random bits to yield high-probability correctness bounds. This is in sharp contrast with Thorup's algorithm [36] which requires $\Omega(m)$ random bits. On a high level, the reason we can do with few random bits is that in each step of the construction of our certificate, we only need the bounds on the number of contracted components to hold with constant probability. Indeed, we will still only have $O(\log n)$ recursive levels with high probability. This means that for the probability bounds within a single recursive level, it suffices to apply Chebyshev's inequality. While the reduction of the number of required random bits is nice, the main point, however, is that with our new certificate we can get down to constant amortized update time per edge-deletion for decremental (2-edge)-connectivity for all but the sparsest graphs.

1.4.3 Maintaining our certificate

As edges are deleted from G , the recursive structure of the c -certificate H changes. Indeed, the deletion of an edge may cause the following changes in one of the recursive layers of H : (1) introduce a cut of size less than δ surrounding a c -edge-connected component or (2) break a c -edge-connected component in two. In the first case, the edges of the cut have to be moved to D , and deleted from the current and later layers of H , causing further cascading. When a c -edge-connected component (in a recursive layer) of H breaks in two, we need to determine whether either of the new components

has less than δ outgoing edges in G . If we use the standard technique of iterating over all the edges incident to nodes of the smaller component, this again incurs an $O(\log(n^2/m))$ cost per edge which is insufficient. However, as we only care about components with at most δ outgoing edges, it turns out that we can do better. We define the *boundary* of a component C of some graph $H \subset G$ to be the set of edges of G with one endpoint in C , and another in $V \setminus C$. To overcome the $O(\log(n^2/m))$ cost per edge, we prove that we can maintain boundaries of size at most δ under splits using a Monte Carlo randomized algorithm in $O(m + n\delta \text{ polylog } n)$ total time. We realize this result by deploying the XOR-trick [1, 2, 27] in a somehow unusual manner. In particular, we randomly partition the set of edges of G and use the XOR-trick to detect the parts that are relevant for scanning, as opposed to standard applications of the XOR-trick, which are used to detect a single replacement edge over a polylogarithmic number of independent samples which unavoidably introduces a polylogarithmic dependence in the cost per edge. Since we are not interested in discovering a single edge incident to some set, we only need to apply the XOR-trick once, which allows us to keep the running time linear in m .

1.4.4 Combining our certificate with Thorup's algorithm

With the certificate as above, the overall idea for a decremental connectivity algorithm is to maintain a c -certificate of (each recursive layer of) the *decremental* graph $H = S \setminus D$ using the algorithm by Thorup [36]. By choosing $P = 1/\log^2 n$, S has $m' = O(m/\log^2 n)$ edges with high probability, so employing the algorithm of Theorem 1.1 on each recursive layer takes total time $O(m' \log^2 n + ncT_c(n) \text{ polylog } n) = O(m + ncT_c(n) \text{ polylog } n)$ with high probability. Let H^* be the c -certificate thus obtained for H . Using a fully dynamic c -edge-connectivity algorithm on $H^* \cup D$ (which undergoes $O(cn \text{ polylog } n)$ updates), we maintain a c -edge certificate of G . As $H^* \cup D$ undergoes $O(cn \text{ polylog } n)$ updates, running the fully dynamic algorithm takes total time $O(cnT_c(n) \text{ polylog } n)$.

We remark that for $c = 1, 2$ we could instead use a fully dynamic c -edge connectivity algorithm on H with polylogarithmic update and query time at the price of a smaller P (which would incur more log-factors in our final time bound). For $c > 2$, however, we only know that $T_c(n) = O(n^{1/2} \text{ poly}(c))$. Since, running a fully dynamic algorithm on H takes total time $\Omega(mT_c(n)/\text{polylog } n)$, this is insufficient to obtain linear time algorithms for dense graphs.

1.4.5 Final self-check

Let us finally describe the ideas behind the final self-checks claimed in Theorem 1.2 and 1.3 in a more general context. In particular, we show that if a randomized Monte Carlo dynamic algorithm satisfies some generic conditions then it can be augmented to detect, at the end of its execution, whether there is any chance that it answered any query incorrectly. That is, if the self-check passes then it is guaranteed that all queries were answered correctly throughout the execution of the algorithm. Otherwise, it indicates that some queries might have been answered incorrectly. The self-check property is particularly useful in applications of dynamic algorithms as subroutines in algorithms solving static problems, that is, it enables static algorithms to exhibit Las Vegas guarantees instead of the Monte Carlo guarantees provided by the dynamic algorithm, as they can simply re-run the static algorithm with fresh randomness until the self-check passes.

The properties of a dynamic algorithm amenable to a self-check behavior are as follows:

- Once a mistake is made by the dynamic algorithm it should be detectable and any subsequent updates of the algorithm do not correct the mistake before it is detected.
- If the dynamic algorithm is stopped at any point in time, it should be able to still perform the self-check within the guaranteed running time of the algorithm.

In our algorithm, as long as the c -certificate maintained by our algorithm is correct, the c -edge-connectivity queries answered by our algorithm exhibit the same guarantees as the fully dynamic c -edge-connectivity algorithm running on the c -certificate H . Hence, we only need to detect potential mistakes in the process of maintaining the c -certificate H . Such mistakes only happen with probability $n^{-\Omega(1)}$.

The c -certificate $H \subseteq G$ of G that we maintain is such that it includes every edge from G that is not in a c -edge-connected component of G (more specifically, not in a c -edge-connected component of the maintained sample $S \subset G$, which includes all edges not in c -edge-connected components of G) and such that every c -edge-connected component of G is also a c -edge-connected component in H . A different equivalent formulation is that for every edge (u, v) of G , if (u, v) is not in H , then u and v are in the same c -edge-connected component of H .

As we later show, the only possible error in the maintenance of the c -certificate is that an edge is missing from H while its endpoints are not c -edge-connected in S . Therefore, to satisfy the properties above, any incorrectly missing edge in the c -certificate should remain missing until the error is detected. We show (in the proof of Theorem 1.2) that before each update to H we can check whether the edge should have been part of the certificate but was omitted due to an error. It is trivial to check for such mistakes during the execution of our algorithm, as follows. For any edge that is deleted from the graph we simply check whether the two endpoints of the deleted edge belong to distinct c -edge-connected components of H and, if so, declare a potential error. For every edge that is inserted to the certificate, due to the updates following an edge deletion, we check whether the edge was supposed to be part of the certificate before the last edge deletion but was omitted due to an error. In either case, our self-check flags an execution invalid only if there has been a mistake (which might or might not have affected c -edge-connectivity queries on the certificate), which happens with low probability. As long as each vertex knows the ID of its c -edge-connected component in the c -certificate, the aforementioned check takes constant time to perform per edge deletion, and thus does not affect the overall running time of our algorithm. Since our algorithm is Monte Carlo randomized and we only flag an execution invalid if a mistake in the maintenance of the certificate was detected, any single execution of our algorithm is flagged invalid with probability $n^{-\Omega(1)}$.

Notice that if the algorithm is terminated before all edges are deleted, we can simply iterate over the remaining edges and apply the aforementioned check for each remaining edge.

2 Preliminaries

The problem of dynamic connectivity consists in designing a data structure that maintains a dynamic undirected graph and supports two operations: an *update* operation which modifies the maintained graph, and a *query* operation, which asks if two given vertices belong to the same connected component of the current graph.

Dynamic connectivity comes in three variants, which differ in the allowed types of update operations. The most general is the *fully dynamic* connectivity problem, in which each update may either add or remove a single edge. The two more restricted variants are *incremental* connectivity – each update adds a single edge, and *decremental* connectivity – each update removes a single edge.

Graphs. Throughout the paper we consider undirected graphs which may have parallel edges, but not self-loops. Generally, for $G = (V, E)$, we denote by n and m the number of vertices and edges of G , respectively. When referring to other graphs $H = (V, E')$, we write $|H|$ to denote $|E'|$.

If $G' = (V', E')$ is a graph with $V' = V$ and $E' \subset E$ then G' is a *subgraph* of G , denoted

$G' \subseteq G$. If W is a set of edges, then we denote by $G \setminus W$ the graph with vertex set V and edge set $E' = E \setminus W$. We often use $G \setminus H$ to denote $G \setminus E(H)$. Finally, if $G = (V, E_G)$ and $H = (V, E_H)$ are graphs on the same set of vertices, then $G \cup H$ is the graph with vertex set V and edge set $E_G \cup E_H$.

Let $A, B \subset V$ be subsets of vertices of $G = (V, E)$. The set of edges from A to B in G is denoted $E_G(A, B)$. For $A \subset V$, we denote by $\partial_G(A) = E_G(A, V \setminus A)$ the *boundary* of A in G .

Cuts. A *cut* of $G = (V, E)$ is a partition of the vertices of G into two non-empty sets V_1, V_2 . We shall mostly identify the cut with the set of edges crossing the cut, $E(V_1, V_2)$. The number of such edges is the *size* of the cut. A cut of G of size $c \in \mathbb{N}$ is called a *c-cut* of G . A *simple cut* of G is a set of edges $S \subset E$ such that $G \setminus S$ has exactly one connected component more than G and such that for every proper subset $S' \subsetneq S$, $G \setminus S'$ has the same connected components as G .

Edge connectivity. Let c be a positive integer. Two distinct vertices u, v of G are *c-edge-connected* if there exist c pairwise edge-disjoint paths from u to v . Being *c-edge-connected* is an equivalence relation on the vertices of G and we call the corresponding equivalence classes the *c-edge-connected classes*. The graph G is *c-edge-connected* if it contains no cut of size $< c$. Equivalently, a graph G is *c-edge-connected* if every pair of distinct vertices of G are *c-edge-connected*. It is worth noting that the graph consisting of a single vertex is *c-edge-connected* since it contains no cut. The *c-edge-connected components* of G are the maximal induced *c-edge-connected* subgraphs of G , i.e., an induced subgraph C of G is a *c-edge-connected component* if it is *c-edge-connected* and there exists no intermediate subgraph $C \subsetneq C' \subset G$ such that C' is *c-edge-connected*.

For $c = 1$, we have simpler terminology. We say that 1-edge-connected vertices are *connected* and call the 1-edge-connected components of G the *connected components* or simply *components* of G . We also use $\mathcal{C}(G)$ to denote the set of connected components of G .

Fully dynamic c-edge-cut. The crucial ingredient in obtaining our general decremental algorithm for arbitrary $c \geq 1$ is the *fully-dynamic c-edge-cut* data structure, defined as follows. Let G be a graph. Then, the data structure maintains any edge e (if one exists) satisfying the following: e belongs to some cut of size $< c$ of the connected component of G containing e . For each $c \geq 1$, we denote by $T_c(n)$ the amortized update time bound of such a data structure that holds whp.

For example, for $c = 1$ we have $T_1(n) = O(1)$ since we do not have to maintain anything. For $c = 2$, the data structure is required to maintain some *bridge* of G and it is known that $T_2(n) = O((\log n \cdot \log \log n)^2)$ [23]. For $c \geq 3$, in turn, we have $T_c(n) = O(n^{1/2} \text{poly}(c))$ [38].

Chernoff Bound. In our analysis we will occasionally need the classic Chernoff concentration bounds. We state a version here for convenience.

Theorem 2.1 (Chernoff Bound). *Let X_1, \dots, X_n be independent random variables supported on $[0, 1]$ and denote by $\mu = \mathbb{E}[\sum_{i=1}^n X_i]$ the mean of their sum. For every $\delta > 0$,*

$$\forall \mu' \geq \mu: \Pr \left[\sum_{i=1}^n X_i > (1 + \delta)\mu' \right] \leq e^{-\frac{\mu'\delta^2}{2+\delta}}, \quad \forall \mu' \leq \mu: \Pr \left[\sum_{i=1}^n X_i < (1 - \delta)\mu' \right] \leq e^{-\frac{\mu'\delta^2}{2}}.$$

Uniform edge sampling. Finally, for a graph $G = (V, E)$ and $p \in [0, 1]$ a real number, we define the *uniform edge sampling* $G(p)$ as follows. Let $\{X_e\}_{e \in E}$ be independent Bernoulli random variables with parameter p and $E' = \{e \in E \mid X_e = 1\}$. Then $G(p) = (V, E')$.

3 Some Useful Properties of c -Edge-Connected Components

In this section, we present structural lemmas regarding the c -edge-connected components, in particular in graphs of high minimum degree. We note that some proofs from Sections 3, 5.1, and 5 can be found in Section 8.

Lemma 3.1 (Benczúr and Karger [5]). *Let c and n be positive integers. Every graph on n vertices with strictly more than $(c - 1)(n - 1)$ edges contains a non-trivial c -edge-connected component.*

Corollary 3.2. *Let c be a positive integer and G be a graph on n vertices. Denote by q_c the number of c -edge-connected components of G . Then the number of edges connecting distinct c -edge-connected components of G is at most $(c - 1)(q_c - 1)$.*

Another central lemma is the following, stating that if a graph on n vertices has at least $3/4n$ vertices with sufficiently high degree, the number of c -edge-connected components is at most $5/6n$.

Lemma 3.3. *Let $G = (V, E)$ be a graph on n vertices such that at least $3n/4$ of its vertices have degree at least $4c$. The number of c -edge-connected components of G is at most $5n/6$.*

Proof. Denote by q_c the number of c -edge-connected components of G . Let $V_0 \subset V$ be the set of vertices of G that are trivial c -edge-connected components. Then $q_c \leq (n - |V_0|)/2 + |V_0| = (n + |V_0|)/2$. Furthermore, every edge incident to a vertex of V_0 connects distinct c -edge-connected components of G . Since there are at most $n/4$ vertices with degrees less than $4c$, at least $|V_0| - n/4$ vertices in V_0 have degree at least $4c$. Hence, there are at least $2c \cdot (|V_0| - n/4)$ edges incident to vertices in V_0 . By Corollary 3.2, we have $c \cdot (n + |V_0|)/2 \geq 2c(|V_0| - n/4)$, which implies $|V_0| \leq 2n/3$. The conclusion follows since $q_c \leq (n + |V_0|)/2 \leq 5n/6$. \square

4 The new c -certificate

In this section we describe our new c -certificate that is instrumental in obtaining the near-optimal decremental connectivity algorithm. A c -certificate H of a graph G allows us to answer queries about c -edge-connected components and c -edge-connected classes of G .

Definition 4.1 (c -certificate). *Let G be a graph and $c \in \mathbb{N}$. A c -certificate for G is a subgraph $H \subseteq G$ such that the c -edge-connected components and classes of G are preserved in H .*

Let $\delta > c$ and $\ell \geq 1$ be integers to be set later. The certificate is defined based on $\ell + 1$ levels of graphs $H_i, G_i \subseteq G$ for $i = 0, \dots, \ell$.

The first step is to sample graphs $H_0^0, H_1^0, \dots, H_\ell^0$ that constitute the basis for graphs H_0, \dots, H_ℓ . Let $p \in (0, 1)$ be a real number to be fixed later. The sampled subgraphs satisfy $(V, \emptyset) = H_0^0 \subseteq H_1^0 \subseteq \dots \subseteq H_\ell^0$. Specifically, for each $i = 1, \dots, \ell$, H_i^0 is constructed as follows. Let $s = \lceil pm \rceil$ and suppose $E = \{e_1, \dots, e_m\}$. Let $r_i : \{1, \dots, s\} \rightarrow \{1, \dots, m\}$ be a *pairwise independent random number generator*, or, in other words, a pairwise independent hash function. Then, we set:

$$H_i^0 := H_{i-1}^0 \cup (V, \{e_{r_i(1)}, e_{r_i(2)}, \dots, e_{r_i(s)}\}).$$

However, we stress that for each level i , we require the random generator to be fully independent from the random generators at previous levels $1, \dots, i - 1$. It is well known [6] that a pairwise independent random number generator can be implemented using $\Theta(\log n)$ random bits so that it generates numbers in constant time in the word RAM model. As a result, if $\Theta(\ell \log n)$ random bits are provided, each H_i^0 can be constructed in $O(si) = O(mpi)$ time and has $O(mpi)$ edges.

Now, the graphs $H_0, G_0, H_1, G_1, \dots, H_\ell, G_\ell$ are defined inductively. Set $G_{-1} = G$. Then for $i = 0, \dots, \ell$ the graphs H_i, G_i are obtained as follows. First, the graph H_i is obtained from $H_i^0 \cap G_{i-1}$ by repeatedly removing all the cuts of size less than c . In other words, H_i equals the c -edge-connected components of $H_i^0 \cap G_{i-1}$. Afterwards, the graph G_i is in turn obtained from G_{i-1} as follows. While for some c -edge-connected component C of H_i , we have $|\partial_{G_i}(C)| < \delta$, the edges of the boundary $\partial_{G_i}(C)$ are removed from both H_i and G_i . Equivalently, one could obtain G_i by first contracting all the c -edge-connected components of H_i in the initial G_i , then repeatedly removing edges incident to vertices of degree $< \delta$ in the contracted graph, and finally undoing all the contractions.

By the construction, the graphs H_1, \dots, H_ℓ and G_1, \dots, G_ℓ satisfy the following properties:

- (1) Every connected component of H_i is c -edge-connected.
- (2) $H_i \subseteq G_i$ and $G_{i+1} \subseteq G_i$ for all $i \geq 0$.
- (3) Each connected component C of H_i satisfies either $\partial_{G_i}(C) = \emptyset$ or $|\partial_{G_i}(C)| \geq \delta$.

Moreover, we have the following property.

Lemma 4.2. *For any $i = 0, \dots, \ell - 1$, $H_i \subseteq H_{i+1}$.*

Proof. First of all, note that $H_i \subseteq H_{i+1}^0 \cap G_i$ since $H_i \subseteq G_i$ and $H_i \subseteq H_i^0 \subseteq H_{i+1}^0$. Moreover, each component of H_i is c -edge-connected so it is contained in some c -edge-connected component of any supergraph of H_i , in particular $H_{i+1}^0 \cap G_i$. As a result, when obtaining H_{i+1} from $H_{i+1}^0 \cap G_i$ by taking the c -edge-connected components, we never remove edges of H_i . \square

Figure 1 shows the inclusion relations between the graphs G_i, H_i (property (2) and Lemma 4.2).

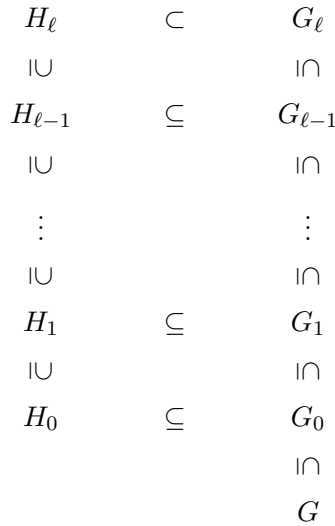


Figure 1: Illustration for Algorithm 1

Lemma 4.3. *There exists $\ell = O(\log n)$ such that if $p\ell < 1$ and $p\delta \geq 32c$, then, with high probability, the connected components of G_ℓ are c -edge-connected and equal to the connected components of H_ℓ .*

Proof. Denote by G'_i the graph G_i with the components of H_i contracted. By property (3), every vertex in G'_i has degree either 0 or at least δ . Let k_i be the number of positive (in fact, at least δ) degree vertices of G'_i .

Recall that H_{i+1} contains precisely the edges inside the c -edge-connected components of $H_{i+1}^0 \cap G_i$. Moreover, let $H'_{i+1} \subset G'_i$ be the graph $H_{i+1}^0 \cap G_i$ with the (c -edge-connected) components of H_i contracted. Consider some vertex v' of G'_i . If v' has degree 0 in G'_i it does so as well in H'_{i+1} . Otherwise, by property (3), v' has degree at least δ in G'_i . Recall that the edges of $H_{i+1}^0 \setminus H_i^0$ are chosen independently of the graphs G'_i and H_i (which only depend on the randomness from levels $0, \dots, i$) via sampling with replacement $s = \lceil pm \rceil$ edges using a pairwise independent random number generator r_{i+1} . Consider a random variable $X_{v'}$ equal to the degree of v' in H'_{i+1} . We now prove that v' has degree less than $4c$, i.e., $X_{v'} < 4c$ with probability no more than $\frac{3}{16}$.

To this end, we introduce two more random variables $Y_{v'}, Z_{v'}$:

- $Y_{v'}$ equals the number of times an edge incident to v' is sampled when sampling $H_{i+1}^0 \setminus H_i^0$:

$$Y_{v'} = \sum_{1 \leq j \leq s} [e_{r_{i+1}(j)} \text{ is incident to } v' \text{ in } G'_i]$$

- $Z_{v'}$ equals the number of collisions incident to v' during sampling $H_{i+1}^0 \setminus H_i^0$, i.e.,

$$Z_{v'} = \sum_{1 \leq j < k \leq s} [r_{i+1}(j) = r_{i+1}(k) \text{ and } e_{r_{i+1}(j)} \text{ is incident to } v' \text{ in } G'_i]$$

Let $d = \deg_{G'_i}(v') \geq \delta$. Since $Y_{v'}$ is a sum of s pairwise independent indicator variables with mean d/m , we have:

$$\begin{aligned} \mathbb{E}[Y_{v'}] &= s \cdot d/m = \lceil pm \rceil \cdot d/m \geq pd. \\ \text{Var}[Y_{v'}] &= s \cdot (d/m) \cdot (1 - d/m) \leq \lceil pm \rceil \cdot (d/m) \leq 2pm \cdot (d/m) = 2pd. \end{aligned}$$

By $pd \geq p\delta \geq 32c \geq 32$ and Chebyshev's inequality $\Pr[Y_{v'} \leq (1 - \varepsilon)\mu] \leq \frac{\text{Var}[Y_{v'}]}{\varepsilon^2(\mathbb{E}[Y_{v'}])^2}$ we have:

$$\Pr[Y_{v'} \leq pd/4] \leq \Pr[Y_{v'} \leq \mathbb{E}[Y_{v'}]/4] \leq \frac{2pd}{\frac{9}{16}(dp)^2} \leq \frac{32}{9pd} \leq \frac{1}{9} < \frac{1}{8}. \quad (1)$$

By pairwise independence we also have:

$$\mathbb{E}[Z_{v'}] = \sum_{1 < j < k \leq s} \frac{1}{m} \cdot \frac{d}{m} \leq \frac{s^2}{2} \cdot \frac{d}{m^2} \leq \frac{4p^2m^2}{2} \cdot \frac{d}{m^2} = 2p^2d.$$

Since we are aiming at proving the lemma for $\ell = \gamma \cdot \log n$ for a constant γ of our choice, we can without loss of generality require that $p\ell < 1$ implies $p \leq 1/256$. Hence, using Markov's inequality we obtain:

$$\Pr[Z_{v'} \geq pd/8] \leq \frac{\mathbb{E}[Z_{v'}]}{pd/8} \leq 16p \leq \frac{1}{16}.$$

Note that we have $X_{v'} \geq Y_{v'} - Z_{v'}$. So, $X_{v'} \leq pd/8$ implies $Y_{v'} - Z_{v'} \leq pd/8$. This in turn implies that either $Y_{v'} \leq pd/4$ or $Z_{v'} \geq pd/8$. As a result, via the union bound we get:

$$\Pr[X_{v'} \leq pd/8] \leq \Pr[Y_{v'} - Z_{v'} \leq pd/8] \leq \Pr[Y_{v'} \leq pd/4] + \Pr[Z_{v'} \geq pd/8] \leq \frac{1}{8} + \frac{1}{16} = \frac{3}{16}.$$

By $pd \geq 32c$ we have that $X_{v'} < 4c$ implies $X_{v'} \leq pd/8$, so we obtain $\Pr[X_{v'} < 4c] \leq \frac{3}{16}$ as desired.

Now let us consider the probability q that more than a fraction of $1/4$ of such n' vertices v' (with degree at least δ in G'_i) have degree less than $4c$ in H'_{i+1} . By (1), the expected number of such vertices is clearly no more than $\frac{3n'}{16}$. As a result, by Markov's inequality, $q \leq \frac{3n'}{16} \cdot \frac{4}{n'} = \frac{3}{4}$. In other words, with probability at least $1 - q \geq 1/4$, at least a fraction of $3/4$ of positive-degree vertices v' of G'_i will have degree at least $4c$ in H'_{i+1} .

Observe that since $G_i \supseteq G_{i+1} \supseteq \dots \supseteq G_\ell \supseteq H_\ell \supseteq \dots \supseteq H_i$, the isolated vertices of G'_i (which are obviously also isolated in H'_{i+1}) correspond to c -edge-connected components of H_i that are also c -edge-connected components of $H_{i+1}, H_{i+2}, \dots, H_\ell, G_i, \dots, G_\ell$. Since $H_i \subseteq H_{i+1}$, by Lemma 3.3, with probability at least $1/4$, the k_i non-isolated vertices of G'_i are “merged” into at most $5k_i/6$ c -edge-connected components of H_{i+1} . Observe that those are the only c -edge-connected components of H_{i+1} that can give rise to positive-degree vertices of G'_{i+1} . Consequently, with probability $\geq 1/4$ we have $k_{i+1} \leq 5k_i/6$. This proves that k_i is very likely to decrease geometrically with i . More concretely, the quantity k_i is 0 for $i = \ell = z \cdot \log n$ (where z is a sufficiently large constant) with high probability via the Chernoff bound.

Note that the lemma follows by $k_\ell = 0$, the fact that $H_\ell \subseteq G_\ell$, and property (1) for $i = \ell$. \square

Finally, we obtain a c -certificate by taking a union of H_ℓ and $G \setminus G_\ell$. Roughly speaking, since H_ℓ sparsifies the c -edge-connected components of G_ℓ , replacing the subgraph G_ℓ with H_ℓ preserves both the c -edge-components and c -edge-classes. The formal proof can be found in the Appendix.

Lemma 4.4. *Let $D := G \setminus G_\ell$. $H_\ell \cup D$ constitutes a c -certificate for G .*

We now show that the basis of our construction, i.e., pairwise independent sampling at $O(\log n)$ levels, yields an interesting low-randomness alternative to Karger's result [28] saying that if a graph G is c' -edge-connected graph, where $c' = \Omega((c + \log n)/p)$, then $G(p)$ is c -edge-connected with high probability (depending on the constant hidden in the Ω notation). Roughly speaking, Karger's proof applies a Chernoff bound to an *exponential* number of cuts in G and therefore requires sampling with full independence, i.e., $\Theta(m)$ random bits. We show that the graph H_0^ℓ has a similar property, but requires only *polylogarithmic* number of random bits: pairwise independence requires $O(\log n)$ bits, and there are $O(\log n)$ sampling levels.

Lemma 4.5. *Let $\ell = \Theta(\log n)$ be as in Lemma 4.3. Let $p' \in (0, 1)$. Suppose G is c' -edge-connected, where $c' = \Omega(c \log n/p')$ and the constant hidden in the Ω notation is sufficiently large. Then the sampled graph H_ℓ^0 , defined as before, has $O(mp')$ edges and is c -edge-connected with high probability.*

Proof. By Lemma 4.3, if $p\ell < 1$ and $p\delta \geq 12c$, then H_ℓ has the same c -edge-connected components as G_ℓ . In particular, for any $c' \geq c$, if G_ℓ is c' -edge-connected, then H_ℓ is c -edge-connected. Observe that G_ℓ can be obtained from G by repeatedly removing from G some cuts of size less than δ . However, if G is c' -edge-connected for $c' \geq \delta$, no such cuts exist in G so in fact we have $G = G_\ell$. Let $p = p'/\ell$, $\delta = 12c/p = \Theta(c \log n/p')$. It follows that if G is $\geq \delta$ -edge-connected, i.e., $\Omega(c \log n/p')$ -edge-connected, then H_ℓ is c -edge-connected with high probability. Since $H_\ell \subseteq H_\ell^0$, so is H_ℓ^0 . \square

5 Decremental Maintenance of a c -certificate

In this section we give an algorithm for maintaining a c -certificate of Section 4 for a graph G that is subject to edge deletions. Even though the graph G is decremental, our maintained certificate will undergo both edge insertions and deletions. However, we will show that, for non-sparse graphs, it is possible that the certificate undergoes only a sublinear-in- m number of updates throughout.

Moreover, we will show that it is possible to maintain the certificate in roughly $O(m) + \tilde{O}(c \cdot n \cdot T_c(n))$ time which is $O(m)$ for non-sparse graphs, depending on the known upper bounds on $T_c(n)$.

In this section we disregard the total number of random bits needed to achieve the claimed bounds. We discuss how the data structure can be implemented using only $O(\text{polylog } n)$ random bits later in Section 7.

During initialization the algorithm samples H_0^0, \dots, H_ℓ^0 as described in Section 4 and initially sets $H_i := H_i^0$ for all i . Furthermore, at the start of the algorithm, each G_i is (conceptually) initialized to G . We stress at this point that the ℓ graphs G_i are stored explicitly only in the basic version of the algorithm. The refined version avoids that, as will be discussed later on. The initialization of graphs H_i and G_i – so that they match their definition from Section 4 – is completed using the update procedure as described below.

The update procedure simply rebuilds the subsequent levels $i = 0, \dots, \ell$ of the certificate according to their definition from Section 4. Each of these maintained graphs H_i, G_i is decremental in time. A deletion of a single edge e of G (or the final step of the initialization) may in general cause a deletion of a larger set A of edges from the level- j graphs H_j, G_j . More precisely, e is first deleted from G_0 . This in turn may give rise to new vertices of degree less than δ in G_0 . Recall that edges incident to such vertices should be repeatedly removed from G_0 until there are none; denote by A the set of edges removed in this process plus the edge e . Observe that all graphs at levels $1, \dots, \ell$ are subgraphs of G_0 , so all the edges from A should also be removed from these graphs. More generally, if the level j is passed a set A of edges to be removed, these edges are first removed from both H_j and G_j . As a result of this change, some new cuts of size less than c may appear in H_j , and consequently some c -edge-connected components of H_j may split. The splits (as well as the deletions of edges from A) may give rise to new boundaries $\partial_{G_j}(C)$ of size less than δ that have to be detected and pruned. The removed boundaries are added to the set A to be passed to subsequent levels.

Algorithm 1 summarizes this conceptual implementation of the above procedure for rebuilding the certificate. In the algorithm, as well as in the following we set $D := G \setminus G_\ell$.

The correctness of this approach follows by Lemmas 4.3 and 4.4 applied to each subsequent version of the graph G . If the certificate is not revealed to the user, and is only used to answer c -edge-connectivity queries or track c -edge-connected components, randomness is not leaked as long as the algorithm gives correct answers (which happens with high probability). By suitably increasing the constants hidden in Lemma 4.3 we obtain high probability correctness for *all* the $O(m)$ versions of the graph.

In the following we assume that $\ell = \Theta(\log n)$, $p = O(1/\log n)$ and $p\delta = \Omega(c)$ so that Lemma 4.3 implies that $H_\ell \cup D$ remains a c -certificate for G .

Lemma 5.1. *The graph $H_\ell \cup D$ has initially $O(mp \log n + n\delta \log n)$ edges and undergoes $O(n\delta \log n)$ edge insertions throughout.*

Proof. The bound on the initial size of H_ℓ follows easily by the used sampling scheme. Moreover, H_ℓ is decremental, whereas the set D can undergo both insertions (when an edge is removed from G_ℓ) and deletions (when an edge deletion is issued to G). Therefore, we only need to prove that D undergoes $O(n\ell\delta)$ insertions throughout. To this end, we show that each G_i undergoes $O(n\delta)$ edge removals following a detection of a component C of H_i with $0 < |\partial_{G_i}(C)| < \delta$. Recall that H_i and G_i are both decremental, so at most $2n - 1$ different components can ever arise in H_i . Each such component causes at most δ insertions to D if its boundary size ever drops below δ . \square

Algorithm 1: Abstract algorithm for maintaining a c -certificate decrementally.

Input : A graph $G = (V, E)$, where $E = \{e_1, \dots, e_m\}$
Parameters: A real $p \in (0, 1)$ and integers $\ell, \delta \in \mathbb{N}$
Maintains : A c -certificate of G given by the graph $D \cup H_\ell$ as defined below

```

1 Procedure Initialize():
2   Initialize graphs  $G_0, \dots, G_\ell$  all equal to  $G$ ;
3   Initialize the empty graph  $D$ ;
4    $H_0 \leftarrow (V, \emptyset)$ ;
5    $s \leftarrow \lceil pm \rceil$ ;
6   for  $i = 1$  to  $\ell$  do
7      $r_i \leftarrow$  a 2-independent random number generator  $\{1, \dots, s\} \rightarrow \{1, \dots, m\}$ ;
8      $H_i \leftarrow H_{i-1} \cup (V, \{e_{r_i(1)}, e_{r_i(2)}, \dots, e_{r_i(s)}\})$ ;
9   CleanUp( $\emptyset$ );
10 Procedure Delete( $e$ ):
11   Delete  $e$  from  $G$  and  $D$ ;
12   CleanUp( $\{e\}$ );
13 /* Internal deletion of set of edges  $A$ , maintains  $D$ ,  $\{H_j\}_{j=0}^\ell$  and  $\{G_j\}_{j=0}^\ell$  */
14 Procedure CleanUp( $A$ ):
15   for  $j = 0$  to  $\ell$  do
16     Delete the edges of  $A$  from  $G_j$  and  $H_j$ ;
17     while there exists an edge  $g \in H_j$  contained in a cut of  $H_j$  of size  $< c$  do
18       Delete  $g$  from  $H_j$ ;
19     while there exists a component  $C$  of  $H_j$  with  $S := \partial_{G_j}(C)$  satisfying  $0 < |S| < \delta$  do
20       Add  $S$  to  $A$  and to  $D$ ;
21       Delete  $S$  from  $G_j$ ;

```

5.1 Supporting data structures

Now we define a few data structures that we use as subroutines when maintaining the certificate. These results are either known or should be considered folklore. For completeness, we provide the proofs of the lemmas in this section in the Appendix.

Restricted fully-dynamic connectivity. Suppose G is a graph subject to edge insertions and deletions. However, assume insertion of an edge $\{u, v\}$ is allowed only if u and v are currently connected. As a result, the connected components of G are decremental in time in the sense that they can only split, but never merge. In this restricted setting we can explicitly maintain the connected components of each vertex and thus support constant-time connectivity queries.

Lemma 5.2. *Let G be a graph subject to edge insertions and deletions. Suppose the endpoints of each edge inserted are connected in G immediately prior to the insertion. Let m be the number of initial edges in G plus the number of insertions issued. There is a data structure that maintains the connected components $\mathcal{C} = \{C_1, \dots, C_k\}$ of G , and an explicit mapping $q : V \rightarrow \{1, \dots, |\mathcal{C}|\}$ such that $v \in C_{q(v)}$. Moreover, after each edge deletion that increases the number of components of G , the data structure outputs a pair (j, A) describing how \mathcal{C} evolves: the component C_j is split into $C_j \setminus A$ and A , where $|A| \leq |C_j \setminus A|$, and we set $C_j := C_j \setminus A$ and $C_{k+1} := A$, and update $k \leftarrow k + 1$. The total update time is $O(m \log^2 n)$, whereas the sum of sizes of sets A output is $O(n \log n)$.*

Maintaining boundaries of splitting sets in a fully dynamic graph. We will often need to solve the following abstract dynamic problem on graphs. Suppose we have two possibly unrelated graphs: a *fully dynamic* graph G and a *decremental* graph H , both on V . We would like to maintain boundaries $\partial_G(C)$ of all the connected components C of H under the allowed updates to G and H .

Lemma 5.3. *Let $G = (V, E)$ be a fully dynamic graph. Let \mathcal{C} be the set of connected components of some (possibly unrelated) decremental graph on V . Suppose the updates to \mathcal{C} are given in the same form as in the output of the data structure of Lemma 5.2. Then, the boundaries $\partial_G(C)$ for $C \in \mathcal{C}$ can be maintained explicitly subject to edge insertions/deletions issued to G , and updates to \mathcal{C} in $O((n + m) \log n)$ total time, where m is the number of initial edges of G plus the number of edge insertions issued to G .*

5.2 Basic data structure

We now discuss how the algorithm maintaining the c -certificate can be efficiently implemented. We start with a basic version of the data structure that does not yet achieve linear dependence on m .

First consider maintaining the graphs H_i . Recall that we need to efficiently detect cuts of size $< c$ in H_i under deletions, prune H_i of these cuts, and keep track of how the c -edge-connected components (or, equivalently, the connected components) of H_i evolve. To this end, we will need the following auxiliary dynamic graph data structures. First of all, we maintain a c -certificate of H_i using the data structure of Theorem 1.1.³ On top of the c -certificate of H_i , we set up a fully-dynamic c -edge-cut data structure, and the data structure of Lemma 5.2. Since the c -edge-connected components of H_i are precisely the components of the graph obtained by repeatedly removing $< c$ -edge cuts from the c -certificate of H_i , these components combined can maintain the c -edge-connected components of H_i and provide an efficient description of the splits these components undergo.

In the basic version of our algorithm, for each G_i in turn, we use a separate decremental boundary maintenance data structure of Lemma 5.3, where the connected components whose boundaries we care about come from H_i . This data structure is passed all the updates to the components of H_i as described in Lemmas 5.2 and 5.3. Recall how the boundary maintenance structures are used: while for some (c -edge-) connected component C of H_i , the boundary of C in G_i has positive size $< \delta$, we remove that boundary from G_i (and propagate that change to subsequent layers $j > i$). In particular, for $i = 0$, since H_0 is empty and has only trivial components, this corresponds to removing from G_0 all edges incident to vertices of degree $< \delta$ until no such vertices exist. The boundaries of size less than δ can be accessed easily using the data structure of Lemma 5.3 associated with G_i .

Assuming a fully-dynamic c -edge-cut data structure with amortized update time $T_c(n)$ and a proper choice of parameters p, δ , the above algorithm runs in $\tilde{O}(m)$ time and, most importantly, updates the maintained c -certificate a sublinear (in m) number of times.

Lemma 5.4. *There exists a decremental algorithm maintaining a c -certificate for G such that the certificate undergoes $O(nc \log^4 n)$ edge updates throughout. The total update time of the algorithm is $\tilde{O}(m) + O(n(c + \log n) \cdot T_c(n) \log^3 n)$ with high probability.*

Proof. We set $p = \frac{1}{\log^3 n}$. This forces us to set $\delta = 12c \log^3 n$ for $p\delta$ to be sufficiently large which is required by Lemma 4.3. Recall that each H_i has $O(mpl)$ edges initially and we maintain its c -certificate under edge deletions using Theorem 1.1. As a result, this incurs a cost of $O(mpl \log n + n \cdot (c + \log n) \cdot T_c(n) \log^2 n)$ time. Since the c -certificate of H_i undergoes $O(n(c + \log n))$ updates, using a fully-dynamic c -edge-cut data structure upon the maintained c -certificate of H_i

³We could in principle use a *decremental* c -edge-cut data structure on the graph H_i itself as opposed to a *fully dynamic* data structure on its c -certificate, but that would prove less efficient.

costs $O(n(c + \log n) \cdot T_c(n))$ time. Similarly, using the data structure of Lemma 5.2 on the c -certificate of H_i costs $O(n(c \log n) \log^2 n)$ time. Summing over all $\ell = O(\log n)$ graphs H_i , we get $O(mp \log^3 n + n(c + \log n) \cdot T_c(n) \log^3 n)$ time. By our choice of p , the first term is $O(m)$.

We also use a simple-minded boundary maintenance data structure of Lemma 5.3 on each of $O(\log n)$ graphs G_i with components from H_i . The total update time of these data structures is $O(m \log^2 n)$. The bound on the number of updates to the certificate follows by Lemma 5.1. \square

Note that the only obstacle preventing us from getting an $O(m)$ bound in Lemma 5.4 is the maintenance of component boundaries for each pair (H_i, G_i) independently. The simple-minded solution yields an $O(m \log^2 n)$ overhead for this task and in fact solves an overly general problem of maintaining the boundaries regardless of their size: recall that we only care about the precise elements of the set $\partial_{G_i}(C)$ for a component C of H_i if $|\partial_{G_i}(C)| \leq \delta$. Otherwise, a (high-probability) guarantee that $|\partial_{G_i}(C)| > \delta$ is sufficient for our needs.

5.3 Maintaining small boundaries

We now describe how to obtain an $O(m) + \tilde{O}(n\delta)$ bound for maintaining all the required small boundaries. Note that this will imply the desired $O(m)$ running time for sufficiently dense graphs, assuming $T_c(n)$ is low enough. First of all, let $\Delta_i := G_i \setminus G_\ell$. We can split the task of maintaining $\partial_{G_i}(C)$ into maintaining $\partial_{G_\ell}(C)$ and $\partial_{\Delta_i}(C)$ separately. Clearly, we have $\Delta_i \subset D$. Recall that D (and thus also Δ_i) is initially empty and undergoes only $O(n\delta \log n)$ insertions throughout. As a result, we can afford maintaining the boundaries of the form $\partial_{\Delta_i}(C)$ even exactly (i.e., regardless of their sizes) using the data structure of Lemma 5.3 in $O(\ell \cdot n\delta \log n \cdot \log n) = O(n\delta \log^3 n)$ time.

It remains to show how to efficiently maintaining the boundaries $\partial_{G_\ell}(C)$ for components C of the graphs H_1, \dots, H_ℓ , provided that $|\partial_{G_\ell}(C)| \leq \delta$. We accomplish this goal using three components.

The sampled graph R . The first component is responsible solely for estimating the sizes $|\partial_{G_\ell}(C)|$. Let R be a graph obtained from G_ℓ via uniform sampling with probability $q = 1/\log^2 n$. Clearly, the graph R has size $\Theta(mq)$ with high probability by the Chernoff bound. Recall that G_ℓ is decremental; whenever an edge e of G_ℓ is deleted, it is removed from R as well if it was sampled. So R can be initialized and maintained in $O(m)$ total time. Assume $\delta = \Omega(\log^2 n \cdot \log m)$, where the constant hidden is sufficiently large. Then, for each version of G_ℓ in time, and each of some $O(\text{poly}\{n, m\})$ sets $C \subseteq V$ chosen independently of R , $|\partial_{G_\ell}(C)| \leq \delta$ implies $|\partial_R(C)| \leq 2q\delta$, and $|\partial_R(C)| \leq 2q\delta$ implies $|\partial_{G_\ell}(C)| \leq 4\delta$, both with high probability via the Chernoff bound.

The small-boundary oracle. Consider the following abstract problem. Suppose $G = (V, E)$ is a fully-dynamic graph. We would like to have a data structure that supports the following query: given some $S \subseteq V$, compute $\partial_G(S)$. The obvious query procedure would be to go through all edges incident to the vertices of S ; this would give a $O(|E(S, V)|)$ query bound. However, if $|S| \cdot |\partial_G(S)|$ is significantly smaller than $|E(S, V)|$, a more efficient solution is possible. Formally, we prove the following theorem which we believe might be of independent interest.

Theorem 5.5. *Let $G = (V, E)$ be an initially empty graph subject to edge insertions and deletions and let $s, 1 \leq s \leq n$, be an integral parameter. There exists a data structure that can process up to $O(\text{poly } n)$ queries about the current set $\partial_G(S)$, where $S \subseteq V$ is the query parameter, so that with high probability, each query is answered correctly in $O\left(|S|s + |E(S, V)| \cdot \frac{|\partial_G(S)|}{s} + \log n\right)$ time. The data structure is initialized in $O(ns)$ time and can be updated in constant time.*

We use the above data structure for G_ℓ and only ask queries when $|\partial_{G_\ell}(S)| = O(\delta)$. By setting $s = \delta \cdot \log^2 n$ we will achieve $O(|S|\delta \log^2 n + |E(S, V)|/\log^2 n)$ query time. Since G_ℓ undergoes m updates, the total update time of this data structure is $O(m + n\delta \log^2 n)$.

Maintaining small boundaries of G_ℓ under splits. Finally, we show how to combine the above two components with a neat variant of the data structure of Lemma 5.3 in order to maintain, for each i , the boundaries of components C of H_i with $|\partial_{G_\ell}(C)| \leq \delta$ in $\tilde{O}(n\delta) + O(m/\log n)$ time with high probability. Through all i , this will imply the desired $\tilde{O}(n\delta) + O(m)$ total time bound.

Let $\mathcal{C} = \{C_1, \dots, C_k\}$ be the components of H_i . Recall that the data structure of Lemma 5.2 for the c -certificate of H_i yields decremental updates of the form (j, C') (where $\emptyset \neq C' \subseteq C_j$ and $|C'| \leq |C_j \setminus C'|$) to \mathcal{C} that set $C_j := C_j \setminus C'$ and $C_{k+1} := C'$. As argued in Lemma 5.2, the total number of updates is at most $n - 1$ and the sum of $|C'|$ over all updates is $O(n \log n)$. We will show how to process these updates so that with high probability, at all times for each C_j , $|\partial_{G_\ell}(C_j)| \leq \delta$ implies that we store the set $\partial_{G_\ell}(C_j)$ explicitly. Wlog. assume that $\mathcal{C} = \{V\}$ initially.

We will say that a set $S \subseteq V$ has *small boundary* if $|\partial_R(S)| \leq 2q\delta$. Otherwise, we say that S has *large boundary*. As argued before, with high probability, the subset of components with small boundary includes all components C of our interest, i.e., with $|\partial_{G_\ell}(C)| \leq \delta$, and does not include any components with $|\partial_{G_\ell}(C)| = \omega(\delta)$. We keep track of which components have large/small boundaries by running a simple-minded boundary maintenance data structure of Lemma 5.3 on R . The total update time of this data structure is $O(n \log n + |R| \log n) = O(n \log n + m/\log n)$ whp.

A naive approach to solve our problem would be to maintain $\partial_{G_\ell}(C)$ for small boundary components $C \in \mathcal{C}$. However, this approach fails for the following reason. Suppose a component C is split into C', C'' , where $|C'| \leq |C''|$. Assume that C'' does have a small boundary, whereas C and C' do not. It is then unclear how to compute the set $\partial_{G_\ell}(C'')$ (of size $\leq \delta$) using time less than linear in either $|C''|$ or $|E(C', V)|$. Had $\Omega(n)$ splits like this happened, we could spend time as much as either $\Theta(n^2)$ or $\Theta(m \log n)$ which is obviously too much. We need a smarter approach.

First of all, denote by \mathcal{S} be the family of sets C that *ever* appeared in \mathcal{C} and had small boundary when still in \mathcal{C} . We will maintain $\partial_{G_\ell}(S)$ for *all* $S \in \mathcal{S}$ – as opposed to exclusively for $S \in \mathcal{S} \cap \mathcal{C}$ as in the naive approach. Moreover, for each $C \in \mathcal{C}$, $C \neq V$, we store (a pointer to) $s(C)$: the unique smallest set in \mathcal{S} such that $C \subsetneq s(C)$. Initially we have $\mathcal{C} = \mathcal{S} = \{V\}$, and $\partial_{G_\ell}(V) = \emptyset$.

We now show how to update the stored information when an update (j, C') comes. Let $A = C'$ and $B = C_j \setminus C'$. Recall that $|A| \leq |B|$. Then, if $C_j \in \mathcal{S}$, we have $s(A) = s(B) = C_j$. Otherwise, we have $s(A) = s(B) = s(C_j)$. Now, if some $C \in \mathcal{C}$ becomes small-boundary⁴ (either as a result of edge deletion issued to R or immediately when it appears), we compute $\partial_{G_\ell}(C)$ as follows. If $|C| \leq |s(C)|/2$, then we compute $\partial_{G_\ell}(C)$ using the small boundary oracle query on C . Otherwise, we compute it by issuing a query about the set $s(C) \setminus C$ to the small boundary oracle and then taking the symmetric difference $\partial_{G_\ell}(s(C)) \Delta \partial_{G_\ell}(s(C) \setminus C)$ which equals $\partial_{G_\ell}(C)$. It is important to note that we do not require that $s(C) \setminus C$ is an element of \mathcal{S} here; it is sufficient to have that $|\partial_{G_\ell}(s(C) \setminus C)| = O(\delta)$, which follows by $\partial_{G_\ell}(s(C) \setminus C) \subseteq \partial_{G_\ell}(C) \cup \partial_{G_\ell}(s(C))$ and $|\partial_{G_\ell}(C)|, |\partial_{G_\ell}(s(C))| \leq \delta$ with high probability. Clearly, taking the symmetric difference takes $O(\delta)$ time.

Lemma 5.6. *The total time spent on computing all the required boundaries $\partial_{G_\ell}(C)$ for $C \in \mathcal{S}$ is $O(n\delta \log^3 n + m/\log n)$ with high probability.*

Proof. Consider a natural tree that the sets of \mathcal{S} form, where each $C \neq V$ is a child of $s(C)$. Let $\mathcal{S}^* = \{C \in \mathcal{S} : |C| \leq |s(C)|/2\}$. Computing boundaries of sets $C \in \mathcal{S}$ such that $|C| \leq |s(C)|/2$, i.e.,

⁴Recall that the boundaries $\partial_R(C)$ are maintained explicitly using the simple-minded data structure of Lemma 5.3. Consequently, it is easy to detect this event “on the fly”.

of sets $C \in \mathcal{S}^*$, requires a single oracle query for $\partial_{G_\ell}(C)$ per each $C \in \mathcal{S}^*$. By Theorem 5.5, such a query costs

$$O\left(|C|\delta \log^2 n + \sum_{v \in C} \frac{\deg(v)}{\log^2 n} + \log n\right)$$

time. When $|C| > |s(C)|/2$, $s(C) \setminus C$ equals the union of siblings of C in the tree that the sets of \mathcal{S} form. So the cost of a query for $\partial_{G_\ell}(s(C) \setminus C)$ is:

$$O\left(\sum_{C' \text{ a sibling of } C} \left(|C'|\delta \log^2 n + \sum_{v \in C'} \frac{\deg(v)}{\log^2 n} + \log n\right)\right)$$

Observe that each $C' \in \mathcal{S}$ has at most one sibling whose size is at least $|s(C')|/2$. As a result, each C' with $|C'| \leq |s(C')|/2$ contributes to a sum above for at most one $C \in \mathcal{S}$ with $|C| > |s(C)|/2$. As a result, the total time spent on small boundary oracle queries (through all $C \in \mathcal{S}$) is:

$$O\left(\sum_{C \in \mathcal{S}^*} \left(|C|\delta \log^2 n + \sum_{v \in C} \frac{\deg(v)}{\log^2 n}\right)\right) = O\left(\sum_{v \in V} \left(\delta \log^2 n + \frac{\deg(v)}{\log^2 n}\right) \cdot |\{C \in \mathcal{S}^* : v \in C\}|\right).$$

Observe that each $v \in V$ can be an element of at most $O(\log n)$ sets of \mathcal{S}^* : all these sets are ancestors of $\{v\}$ in the tree corresponding to \mathcal{S} and have size smaller than their respective parent by a factor of at least 2. As a result, the total time spent on this step is $O(n\delta \log^3 n + \sum_{v \in V} \deg(v)/\log n) = O(n\delta \log^3 n + m/\log n)$. This dominates the $O(n\delta \log n)$ cost of taking symmetric differences. \square

We also need to maintain the stored boundaries $\partial_{G_\ell}(C)$ for $C \in \mathcal{S}$ under edge deletions that G_ℓ undergoes. However, to avoid spending $O(m)$ time per single H_i on this, for this part we need to consider all the graphs H_i simultaneously. Note that for a fixed i , \mathcal{S} only grows and contains at most $2n - 1$ elements. Hence, the total size of the stored sets $\partial_{G_\ell}(C)$, $C \in \mathcal{S}$, is $O(n\delta)$ (whp). For each $e \in E(G_\ell)$ we maintain a list of pointers to such stored boundaries with $e \in \partial_{G_\ell}(S)$, through all H_i . The total number of pointers ever inserted into these lists is clearly $O(n\delta\ell) = O(n\delta \log n)$. When an edge e is removed from G_ℓ , we scan the attached list of e and remove this edge from the required boundaries it was contained in. The total time spent on this can be seen to be no more than the total number of insertions into the lists, i.e., $O(n\delta \log n)$.

Theorem 1.2. *There exists a Monte Carlo randomized algorithm for the decremental c -certificate problem with total update time $O(m+n(c+\log n) \cdot T_c(n) \log^3 n + nc \log^7 n)$. The maintained certificate undergoes $O(nc \log^4 n)$ edge insertions and deletions throughout. The algorithm is correct with high probability. Within this time bound, the algorithm offers a final self-check after processing all updates.*

Proof. Recall that the simple reduction from Lemma 5.4 had $O(m) + O(n(c + \log n) \cdot T_c(n) \log^3 n)$ operation cost of the fully-dynamic c -edge connected components data structures. It also required setting δ to at least $c \log^3 n$. The cost of maintaining the needed small boundaries is dominated by the application of Lemma 5.6 for each $i = 1, \dots, \ell$. The statement about the running time of the algorithm follows.

We now turn to proving the statement that the algorithm offers a final self-check after processing all updates. Notice that we only need to check that no edge between distinct c -edge-connected components of the certificate was missing at any point throughout the execution of the algorithm; indeed, even if an edge with both endpoints in the same c -edge-connected component of the certificate was missing that wouldn't affect any answers to c -edge-connectivity queries on the certificate.

First note that in a correct execution of our algorithm all edges between distinct c -edge-connected components of H_l (and hence, of the certificate) are always present in the certificate. That is, we only need to verify that no edge between two distinct (c -edge-) connected components of H_l is added and that each deleted edge from G is either present in the certificate or both of its endpoints belong to the same (c -edge-) connected component of H_l . We assume that each vertex has access to the ID of its (c -edge-) connected component in H_l , so that we can check in constant time whether the two vertices are part of the same (c -edge-) connected component. These IDs are provided by invocation of the Lemma 5.2 on the certificate. Whenever an edge e is deleted from the graph G (and hence from the certificate), we simply check that e is part of the certificate if its endpoints are in distinct (c -edge-) connected components of H_l ; if that is not the case, we mark the execution of the algorithm invalid, as edge e should have been part of the certificate. On the other hand, if both the endpoints of a deleted edge e were part of the same (c -edge-) connected component of H_l , then no query might have been answered incorrectly.

Finally, edges might be added to the certificate due to the update in our data structures following an edge deletion from G . Again, we need to make sure that no edge is added to the certificate that was supposed to be there before the edge deletion and is omitted due to an error. Specifically, for the edges added to the certificate we need to check that both of their endpoints are in the same (c -edge-) connected component of H_l right before the last edge deletion from G (which potentially caused the splitting of connected components of H_l). If that is not the case, then we again flag the execution invalid as the endpoints of these edges were part of distinct (c -edge-) connected components of H_l and should already be part of the certificate. Notice that the splits of (c -edge-) connected components of H_l are described by the output of the data structure of Lemma 5.2, and hence the queries can be answered efficiently (even an $O(\log n)$ bound per query would be enough to keep the running time within the stated bound due to the limited number of updates to the certificate). \square

5.4 Small boundary oracle

In this section we prove Theorem 5.5. Recall that the goal is to have a data structure that maintains a fully dynamic graph $G = (V, E)$ and supports queries regarding $\partial_G(S)$, where $S \subseteq V$ is a query parameter.

First of all, we will leverage the well-known XOR trick [1, 2] for deciding if a boundary of some subset of vertices is non-empty. We now briefly describe this method. Suppose each $e \in E$ is assigned a random bit-string x_e of length $\Theta(\log n)$ that fits in $O(1)$ machine words. Let $x_v = \bigoplus_{vw=e \in E} x_e$ denote the XOR of the respective bit-strings of edges incident to v . Then, one can prove that, given $S \subseteq V$, with high probability the XOR $\bigoplus_{u \in S} x_u$ is non-zero if and only if $\partial_G(S) \neq \emptyset$. So, emptiness of $\partial_G(S)$ can be tested in $O(|S|)$ time.

Let $s \geq 1$ be an integral parameter. The main idea is as follows. We partition the edge set E into E_1, \dots, E_s . Each $e \in E$ is assigned to one of these sets uniformly at random. Let us apply the XOR-trick for each E_i separately. To this end, now x_v is a vector of s bit-strings, where $x_v(i) = \bigoplus_{vw=e \in E_i} x_e$. Given that, in $O(s|S|)$ time we can find the set I of all i such that $\partial_G(S) \cap E_i \neq \emptyset$ (whp). Clearly, in order to find $\partial_G(S)$, we only need to look for this boundary's elements in $(\bigcup_{i \in I} E_i) \cap E_G(S, V)$. If $|\partial_G(S)|$ is small compared to s , one can prove that, with high probability, this strategy is more efficient than iterating through the entire set $E(S, V)$. We prove this formally below.

Lemma 5.7. *Let $S \subseteq V$. Then, with high probability, the query procedure computes $\partial_G(S)$ correctly in $O\left(s|S| + |E_G(S, V)| \cdot \frac{|\partial_G(S)|}{s} + \log n\right)$ time.*

Proof. Let the bit-strings x_e consist of $\gamma = O(1)$ machine words, each with at least $\lceil \log_2 n \rceil$ bits. As argued before, computing the bit-strings $y(i) = \bigoplus_{v \in S} x_v(i)$ for all $i = 1, \dots, s$ and finding the set $I = \{i : y(i) \neq 0\}$ takes $O(s|S|)$ worst-case time.

Now, let us consider the number of edges searched. Suppose that $e \in E_G(S, S)$ has endpoints $u, v \in S$ and belongs to E_j . Then x_e does not contribute to $y(j)$ as it is present twice in the XOR, once in $x_v(j)$ and once in $x_u(j)$. Hence, if $y(j) \neq 0$, i.e., $j \in I$, then there is an edge of E_j that has only one endpoint in S and thus belongs to $\partial_G(S)$. Since each $e \in \partial_G(S)$ contributes to a single element of y , $|I| \leq |\partial_G(S)|$. Now, for $e \in E$, let Y_e be the indicator of the event ($e \in \bigcup_{i \in I} E_i$), i.e., $Y_e = 1$ if $e \in \bigcup_{i \in I} E_i$ and $Y_e = 0$ otherwise. The set I is entirely determined by the variables $(x_e)_{e \in \partial_G(S)}$, so for the remaining edges, $E_G(S, V) \setminus \partial_G(S) = E_G(S, S)$, the random variables $\{Y_e\}_{e \in E_G(S, S)}$ are mutually independent and independent of the choice of I except for its size, $|I|$. It follows that the sum $Y = \sum_{e \in E_G(S, S)} Y_e$ satisfies

$$\mathbb{E}[Y] = |E_G(S, S)| \cdot \frac{|I|}{s} \leq |E_G(S, V)| \cdot \frac{|\partial_G(S)|}{s}.$$

Since Y is a sum of independent random variables, we may write $\mu = |E(S, V)| \cdot |\partial_G(S)|/s$. Let $t > 0$ be a constant and $\mu' = \mu + 3t \log n$, and apply the Chernoff bound of Theorem 2.1 to get

$$\Pr[Y > 2\mu + 6t \log(n)] = \Pr[Y > (1 + 1)\mu'] \leq \exp(-\max\{\mu, 3t \log(n)\}/3) \leq n^{-t}.$$

Thus, with high probability the number of edges checked by the algorithm is

$$O\left(|\partial_G(S)| + |E_G(S, V)| \cdot \frac{|\partial_G(S)|}{s} + \log n\right),$$

and, as a result, the running time of the query procedure is

$$O\left(s|S| + |\partial_G(S)| + |E_G(S, V)| \cdot \frac{|\partial_G(S)|}{s} + \log n\right).$$

To obtain the desired bound note that if $s \geq |\partial_G(S)|$, then the term $|\partial_G(S)|$ above is dominated by $s|S|$, and otherwise it is dominated by the third term.

Finally, let us consider the probability that the output of the query is correct. It is not hard to see that the output is correct if and only if the set I corresponds to the set $J = \{j \in [s] \mid E_j \cap \partial_G(S) \neq \emptyset\}$, since in that case, the algorithm searches all groups containing an edge of $\partial_G(S)$. We have already established that $I \subset J$. So let $j \in J$ be given. Then there is some edge $e \in \partial_G(S) \cap E_j$. We have:

$$y(j) = \bigoplus_{v \in S} x_v(j) = \bigoplus_{f \in \partial_G(S) \cap E_j} x_f$$

since for every edge e of $E_G(S, S)$, x_e appears twice in the XOR. The probability that $y(j) = 0$, or equivalently $j \notin I$, is hence the probability that $x_e = \bigoplus_{f \in (\partial_G(S) \setminus \{e\}) \cap E_j} x_f$. Since x_e is independent of the right-hand side, this probability is exactly $2^{-|x_e|} \leq 2^{-\gamma \log_2 n} \leq n^{-\gamma}$. By a union bound, it follows that $J \subset I$ with probability at least $1 - |J| \cdot n^{-\gamma} \leq 1 - sn^{-\gamma} \leq 1 - n^{\gamma-1}$. \square

When an edge $e = uv$ is inserted into G , all we have to do is pick a random set E_j for e , sample a random bit-string x_e , and update $x_w(j) := x_w(j) \oplus x_e$ for $w \in \{u, v\}$. To handle the deletion of e , all we have to do is to repeat the last step of insertion and remove e from E_j . So an edge update can be clearly performed in $O(\gamma) = O(1)$ worst-case time. The data structure can be initialized in $O(ns + m)$ time by first filling the values $x_v(i)$ with zeros and then inserting all the edges.

Finally, to guarantee high-probability correctness and query time bounds for poly(n) queries, it is enough to set constants γ and t (from the proof of Lemma 5.7) sufficiently large. The full pseudocode of the data structure is given in Algorithm 2.

Algorithm 2: Small boundary oracle.

Input : A graph $G = (V, E)$ on n vertices
Parameters: Positive integers $\gamma = O(1)$ and $s \leq n$.

- 1 **Procedure Initialize():**
- 2 Initialize s sets E_1, \dots, E_s ;
- 3 Fill values $x_v(j)$ for $v \in V$ and $j \in [s]$ with all-zero bit-strings of length $\gamma \cdot \lceil \log_2 n \rceil$;
- 4 **for** $e \in E$ **do**
- 5 **Insert**(e);
- 6 **Procedure Insert**($e = \{u, v\}$):
- 7 Insert e into some E_j of E_1, \dots, E_s uniformly at random;
- 8 Let $x_e \in \{0, 1\}^{\gamma \cdot \lceil \log_2 n \rceil}$ be a bit-string chosen uniformly at random;
- 9 Let $x_v(j) := x_v(j) \oplus x_e$;
- 10 Let $x_u(j) := x_u(j) \oplus x_e$;
- 11 **Procedure Delete**($e = \{u, v\}$):
- 12 Let E_j be the set containing e ;
- 13 Delete e from E_j ;
- 14 Let $x_v(j) := x_v(j) \oplus x_e$;
- 15 Let $x_u(j) := x_u(j) \oplus x_e$;
- 16 **Function FindBoundary**(S): /* Find $\partial_G(S)$ for a subset $S \subset V$ */
- 17 Let $B := \emptyset$;
- 18 Let $y := \bigoplus_{v \in S} x_v$;
- 19 Let $I := \{i \in [s] \mid y(i) \neq 0\}$;
- 20 **for** $uv = e \in \bigcup_{i \in I} (E_i \cap E(S, V))$ **with** $u \in S$ **do**
- 21 **if** $v \notin S$ **then** $B := B \cup \{e\}$;
- 22 **return** B

6 Decremental c -Edge-Connectivity

In this section we briefly explain how Theorem 1.2 implies decremental c -edge-connectivity algorithms with $O(m)$ total update time for sufficiently dense graphs.

It is important to note at this point that there are two settings that might be of interest. First, we might want to have a decremental algorithm maintaining c -edge-connected *components*, that is, supporting queries whether two vertices belong to the same c -edge-connected component of G . However, we might alternatively want to have a decremental algorithm maintaining the c -edge-connected *classes*, i.e., supporting queries whether there exist c edge-disjoint paths between some two vertices. Recall that these settings are equivalent for $c = 1, 2$, but differ for $c \geq 3$: then a pair of c -edge-connected vertices might not belong to the same c -edge-connected component.

Let us first consider the decremental c -edge-connected components problem. Then we have:

Theorem 6.1. *There exists a Monte Carlo randomized decremental c -edge-connected components algorithm with $O(m + nc(\log^7 n + \log^4 n \cdot T_c(n)))$ total update time. The algorithm is correct with high probability.*

Proof. We maintain a c -certificate H of G using Theorem 1.2. Observe that H has the same c -edge-connected components as G . We additionally maintain a fully-dynamic c -edge-cut data structure for H , and a data structure of Lemma 5.2. These two combined allow us to prune the

certificate from $< c$ -edge-cuts and explicitly maintain the c -edge-connected components of H (which enables constant-time queries about the component a vertex belongs to). Since H undergoes only $O(nc \log^4 n)$ edge updates, and each can be processed in $O(T_c(n) + \log^2 n)$ amortized time, the theorem follows. \square

By plugging in the specific known upper bounds on $T_c(n)$ for $c = 1, 2$, we obtain:

Theorem 1.4. *There exists Monte Carlo randomized decremental connectivity and decremental 2-edge-connectivity algorithms with $O(m + n \log^7 n)$ total update time and $O(1)$ query time.*

For $c \geq 3$, $T_c(n) = O(n^{1/2} \text{poly}(c))$ has been proved [38], and therefore for $c = O(n^{o(1)})$ the c -edge-connected components can be maintained under deletions in $O(m) + \tilde{O}(n^{3/2+o(1)})$ total time.

Theorem 1.6. *Let $c = O(n^{o(1)})$. There exists a Monte Carlo randomized decremental c -edge-connected components data structure with $O(m + n^{3/2+o(1)})$ total update time and $O(1)$ query time.*

Now consider the decremental c -edge-connected classes problem., i.e., decremental pairwise c -edge-connectivity.

Theorem 6.2. *Suppose there exists a fully-dynamic c -edge-cut algorithm with $T_c(n)$ amortized update time, and a fully-dynamic pairwise c -edge-connectivity algorithm with $U_c(n)$ amortized update time and $Q_c(n)$ query time. There exists a Monte Carlo randomized decremental c -edge-connected components algorithm with $O(m + nc \log^7 n + n(c + \log n)T_c(n) \log^2 n + nc \log^4 n \cdot U_c(n))$ total update time and $O(Q_c(n))$ query time. The algorithm is correct with high probability.*

Proof. We maintain a c -certificate H of G using Theorem 1.2. Recall that H has the same c -edge-connected components as G . So, we additionally maintain the certificate using the assumed fully-dynamic c -edge-connected classes data structure and use it to answer queries. \square

Jin and Sun [25] have recently showed that for $c = (\log n)^{o(1)}$, a deterministic fully-dynamic c -edge-connected classes data structure with $U_c(n) = O(n^{o(1)})$ and $Q_c(n) = O(n^{o(1)})$ exists. By combining their result with the fully-dynamic c -edge-cut algorithm of Thorup [38] with $T_c(n) = O(n^{1/2} \text{poly}(c))$, we obtain the following.

Theorem 1.5. *Let $c = (\log n)^{o(1)}$. There exists a Monte Carlo randomized decremental c -edge-connectivity data structure which can answer queries to whether two vertices are in the same c -edge connected class in $O(n^{o(1)})$ time, and which has $O(m) + \tilde{O}(n^{3/2})$ total update time.*

7 Reducing the Number of Random Bits

In this section we show that our algorithms can be tuned to require only $O(c \text{poly} \log n)$ random bits over all updates. We take advantage of the pseudorandom number generator by Christiani and Pagh [7], which, given only a $O(c \text{poly} \log n)$ truly random bits, can generate $O(n)$ random numbers, such that each number is between 1 and n' , $n \leq n' \leq 2n$ and the generated numbers are $\Theta(c \text{poly} \log n)$ -independent. Generating each number takes $O(1)$ time, whereas initialization takes $O(c \text{poly} \log n)$ time.

The key property that we use is the fact that $\Theta(\log n)$ -independence is sufficient for a Chernoff-like bounds to hold [7].

Our algorithm uses randomness for three purposes:

1. In order to initially sample the graphs H_i^0 (for all i) and R .
2. Within the data structure of Theorem 1.1 to maintain a c -certificate of each H_i .
3. Within the small-boundary oracle, to partition the edges of E into sets E_1, \dots, E_s .
4. Within the small-boundary oracle, to generate the random bits associated with each edge.

We now discuss how to implement each item using the pseudorandom generator of [7].

We already argued in Section 4 that for sampling H_1^0, \dots, H_ℓ^0 , a polylogarithmic number of random bits is sufficient. For the sampled graph R we only used Chernoff bounds for a polynomial number of sums of indicator variables, so indeed polylogarithmic independence is enough, and the sampling can be performed using the pseudorandom generator of [7].

When making use of the partition in item (3), for efficiency we only apply the Chernoff bound to a polynomial number of sums of independent indicator variables Y_e with the same mean for edges e in some subset of E . Hence, $O(\text{polylog } n)$ -independence between the variables Y_e is sufficient. The partition of E can be performed by sampling each E_i to be a $\left[(m - \sum_{j < i} |E_j|) / (s - i + 1) \right]$ -subset of $E \setminus \left(\bigcup_{j < i} E_j \right)$. This is easily implemented using the pseudorandom generator.

Consider item (4). Whenever the decremental certificate algorithm performs a query on the small boundary oracle, with high probability the requested boundary $\partial_G(S)$ contains $O(c \cdot \text{polylog } n)$ edges. As a result, only $O(c \cdot \text{polylog } n)$ edge bit-strings x_f participate in each computed value $y(j) = \bigoplus_{f \in \partial_G(S) \cap E_j} x_f$. Therefore, it is enough that the edge bit-strings are $O(c \cdot \text{polylog } n)$ independent instead of fully independent.⁵ As a result, the individual bit-strings can be obtained from the pseudorandom generator in $O(m)$ time.

Finally, dealing with item (2) is less straightforward. This is because in the analysis of the data structure of [36, Theorem 6], Thorup invokes a result due to Karger [28] saying that if a graph G is c' -edge-connected graph, where $c' = \Omega((c + \log n)/p')$, and $p' \in (0, 1)$ then $G(p')$ is c -edge-connected with high probability (depending on the constant hidden in the Ω notation). This is then used to show that the number of edges between different c -edge-connected components of a certificate (which is $G(p')$ augmented with the edges of G connecting distinct c -edge-connected components of $G(p)$, where p' is any constant less than 1) that Thorup uses is $O(c'n)$ with high probability. This is where the $c + \log n$ term in the bounds in Theorem 1.1 comes from. Unfortunately, roughly speaking, Karger's proof applies a Chernoff bound to an *exponential* number of cuts in G and therefore requires sampling with full independence, i.e., $\Theta(m)$ random bits.

We can eliminate the need for full independence in [36], albeit at the cost of replacing the $c + \log n$ terms in the bounds of Theorem 1.1 with $c \log n$. To this end, one can leverage Lemma 4.5 and replace the uniformly sampled subgraph $G(p')$ with the graph H_ℓ^0 from our construction with p set to p'/ℓ (computable in $O(mpl) = O(mp')$ time using $O(\text{polylog } n)$ random bits), at the cost of replacing the $c + \log n$ terms with $c \log n$ in the bounds of Theorem 1.1.

8 Omitted Proofs

Lemma 3.1 (Benczúr and Karger [5]). *Let c and n be positive integers. Every graph on n vertices with strictly more than $(c - 1)(n - 1)$ edges contains a non-trivial c -edge-connected component.*

⁵In general, the XOR trick can be used with polylogarithmic independence even for testing non-emptiness of large (i.e., up to size n) boundaries [18, 39]. Although a single bit of an edge bit-string can be generated in constant time [39], we need $\Theta(\log n)$ bits per edge to guarantee high probability correctness. As a result, using known tools, generating all edge bit-strings would cost $\Theta(m \log n)$ time which is too expensive for our application.

Proof. We proceed by strong induction on n . The statement is clearly true for $n = 1, 2$. Consider now a graph G on $n > 2$ vertices and $(c - 1)(n - 1) + 1$ edges. If no simple cut of G of size $d < c$ exists, then G is c -edge-connected and we are done. Otherwise, deleting the simple cut from G , we obtain subgraphs G_1 and G_2 of G of sizes n_1 and n_2 , respectively, such that $n_1 + n_2 = n$. After deleting the simple cut there are at least $(c - 1)(n - 2) + 1 = (c - 1)(n_1 - 1) + (c - 1)(n_2 - 1) + 1$ edges left. Hence, by the pigeonhole principle and the induction hypothesis, either G_1 or G_2 contains a non-trivial c -edge-connected component. \square

Corollary 3.2. *Let c be a positive integer and G be a graph on n vertices. Denote by q_c the number of c -edge-connected components of G . Then the number of edges connecting distinct c -edge-connected components of G is at most $(c - 1)(q_c - 1)$.*

Proof. Contracting the c -edge-connected components of G we arrive at a graph G' on q_c vertices with no c -edge-connected components. All edges of G that connect distinct c -edge-connected components of G are still present in G' . By Lemma 3.1, G' contains at most $(c - 1)(q_c - 1)$ edges, which completes the proof. \square

Lemma 5.2. *Let G be a graph subject to edge insertions and deletions. Suppose the endpoints of each edge inserted are connected in G immediately prior to the insertion. Let m be the number of initial edges in G plus the number of insertions issued. There is a data structure that maintains the connected components $\mathcal{C} = \{C_1, \dots, C_k\}$ of G , and an explicit mapping $q : V \rightarrow \{1, \dots, |\mathcal{C}|\}$ such that $v \in C_{q(v)}$. Moreover, after each edge deletion that increases the number of components of G , the data structure outputs a pair (j, A) describing how \mathcal{C} evolves: the component C_j is split into $C_j \setminus A$ and A , where $|A| \leq |C_j \setminus A|$, and we set $C_j := C_j \setminus A$ and $C_{k+1} := A$, and update $k \leftarrow k + 1$. The total update time is $O(m \log^2 n)$, whereas the sum of sizes of sets A output is $O(n \log n)$.*

Proof. First of all, we store G in a fully-dynamic connectivity data structure with $O(\log^2 n)$ amortized update time and $O(\log n)$ query time [21]. This data structure also maintains a spanning forest explicitly and allows $O(\log n)$ -time queries about the size of the component containing a given vertex. If an edge is inserted, we just pass the insertion to the fully-dynamic data structure – this insertion does not change the connected components of G . If an edge $\{u, v\}$ is deleted, we additionally check if u and v are still connected after removing $\{u, v\}$. If not, assume wlog. that the component of u is not smaller than that of v afterwards. We set A to be the vertices of the tree containing v in the spanning forest. Let $q(u) = C_j$. For each $x \in A$ we remove x from C_j , and set $q(x) := k + 1$. Finally, we set $C_{k+1} := A$, increment k , and output (j, A) . To bound the total time spent outside the fully-dynamic connectivity data structure, note that each time we spend time proportional to the size of the output set A . Whenever a vertex x belongs to A , the size of the component of x decreases by a factor of at least two due to the edge update. As a result, each x can occur $O(\log n)$ times in the output sets A , and hence the total size of these sets is $O(n \log n)$. \square

Lemma 5.3. *Let $G = (V, E)$ be a fully dynamic graph. Let \mathcal{C} be the set of connected components of some (possibly unrelated) decremental graph on V . Suppose the updates to \mathcal{C} are given in the same form as in the output of the data structure of Lemma 5.2. Then, the boundaries $\partial_G(\mathcal{C})$ for $C \in \mathcal{C}$ can be maintained explicitly subject to edge insertions/deletions issued to G , and updates to \mathcal{C} in $O((n + m) \log n)$ total time, where m is the number of initial edges of G plus the number of edge insertions issued to G .*

Proof. Note that similarly as in Lemma 5.2, the updates to \mathcal{C} are given in such a way that we can explicitly maintain, for each $v \in V$, the component from \mathcal{C} it belongs to. This takes $O(n \log n)$ total time. It also enables us to decide in $O(1)$ time whether an edge $\{u, v\}$ belongs to two (if u, v are

disconnected) or zero boundaries (otherwise). Each boundary $\partial_G(C)$ is stored in a linked list $L(C)$. Each edge of G connecting endpoints in different components of \mathcal{C} has associated two pointers to its places in the two respective lists. Hence, whenever an edge is inserted/deleted from G , the lists storing the boundaries can be easily updated in constant time.

Now suppose \mathcal{C} is updated: some $C_j \in \mathcal{C}$ gets split into $C_j \setminus A$ and A , where $|A| \leq |C_j \setminus A|$. Clearly, only the lists $L(C_j)$ and $L(A)$ may need to be fixed at this point. We now iterate through all $\{u, v\} = e \in E_G(A, V)$ and proceed as follows. Suppose wlog. that $u \in A$. If $v \in C_j \setminus A$, then we add e to $L(C_j)$ (it was not there before the split) and update all the auxiliary pointers. Otherwise, if $v \in V \setminus (C_j \cup A)$, then e is removed from $L(C_j)$ and inserted into $L(A)$. Otherwise, if $v \in A$, then we skip that edge as it remains an intra-component edge after the split. It is easy to verify that the lists represent the required boundaries after this step, which takes $O(\sum_{u \in A} \deg(u))$ time.

Finally, the $O(m \log n)$ total update time bound follows since the incident edges of each vertex v are traversed $O(\log n)$ times – when this happens, the size of v 's component in \mathcal{C} halves. \square

Lemma 4.4. *Let $D := G \setminus G_\ell$. $H_\ell \cup D$ constitutes a c -certificate for G .*

Proof. First, we prove that $H_\ell \cup D$ preserves the c -edge-connected components of G . For convenience, denote $G_c = H_\ell \cup D$. Assume, by contradiction, this is not true. Note that for each c -edge-connected component C' of H_ℓ it holds that $C' \subset C$ for some c -edge-connected C component of G , as otherwise $G[C']$ contains a $< c$ -cut and so does $H_\ell[C']$ since $H_\ell \subset G$; a contradiction. Let C be a c -edge-connected component of G that is not preserved in G_c . Then, there exists a k -cut S , for $k < c$, in $G_c[C]$ that is not a k -cut in $G[C]$. Let C_1, C_2 be the two different connected components of $G_c[C] \setminus S$. To conclude the argument, we next show that all edges in $(C_1 \times C_2) \cap E$ are present in G_c , which implies that if S is a k -cut in $G_c[C]$ it is also a k -cut in $G[C]$, and hence we contradict the assumption that C is a c -edge-connected component in $G[C]$ but not in $G_c[C]$. Take any edge in $uv \in (C_1 \times C_2) \cap E$. Vertices u and v belong to different c -edge-connected components of H_ℓ , as otherwise, there would be no k -cut separating u, v in G_c which contains H_ℓ . Hence, the edge $uv \in D \subseteq G_c$, since D contains all edges of G between components of H_ℓ . This concludes the proof that G_c preserves the c -edge-connected components of G .

Now we turn to proving that G_c preserves also the c -edge-connected classes of G . To this end, we first show that every $< c$ -cut of G_c is a $< c$ -cut of G . Let S be a $< c$ -cut of G_c . For contradiction, suppose some $u, v \in V$ are connected in $G \setminus S$ but not in $G_c \setminus S$. Let P be a $u \rightarrow v$ path in $G \setminus S$. The endpoints of some edge $xy \in P$ have to be disconnected in $G_c \setminus S$, as otherwise a path from u to v would exist in $G_c \setminus S$. However, if xy is contained in G_ℓ , then x and y lie in the same c -edge-connected component of H_ℓ , i.e., they are connected in $H_\ell \setminus S$ by $|S| < c$. Otherwise, since $D = G \setminus G_\ell$, we have $xy \in D \setminus S$, so x and y are connected in $G_c \setminus S$ as well. This contradicts the fact that x and y are disconnected in $G_c \setminus S$.

Now, let $u, v \in V$. If u and v are c -edge-connected in a subgraph of G , in particular G_c , then they are c -edge-connected in G . Conversely, if u and v are not c -edge-connected in G_c then there is a cut of size $< c$ separating them in G_c . Such a cut is also a cut in G by the previous claim, so u and v are not c -edge-connected in G either. This proves that the c -edge-connected classes of G and G_c are identical. \square

References

- [1] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 459–467. SIAM, 2012.
- [2] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In Michael Benedikt, Markus Krötzsch, and Maurizio Lenzerini, editors, *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 5–14. ACM, 2012.
- [3] S. Alstrup, T. Husfeldt, and T. Rauhe. Marked ancestor problems. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 534–543, 1998.
- [4] Stephen Alstrup, Jens Peter Secher, and Maz Spork. Optimal on-line decremental connectivity in trees. *Information Processing Letters*, 64(4):161 – 164, 1997.
- [5] András A. Benczúr and David R. Karger. Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM Journal on Computing*, 44(2):290–319, 2015.
- [6] Larry Carter and Mark N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.
- [7] Tobias Christiani and Rasmus Pagh. Generating k-independent variables in constant time. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 196–205. IEEE Computer Society, 2014.
- [8] Julia Chuzhoy, Yu Gao, Jason Li, Danupon Nanongkai, Richard Peng, and Thatchaphol Saranurak. A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond. *CoRR*, abs/1910.08025, 2019.
- [9] David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *J. ACM*, 44(5):669–696, 1997.
- [10] Paul Erdos and Alfred Renyi. On the evolution of random graphs. *Publ. Math. Inst. Hungary. Acad. Sci.*, 5:17–61, 1960.
- [11] Shimon Even and Yossi Shiloach. An on-line edge-deletion problem. *J. ACM*, 28(1):1–4, 1981.
- [12] Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput.*, 14(4):781–798, 1985.
- [13] M. Fredman and M. Saks. The cell probe complexity of dynamic data structures. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing (STOC)*, STOC '89, page 345–354, New York, NY, USA, 1989. Association for Computing Machinery.
- [14] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.
- [15] Harold N. Gabow, Haim Kaplan, and Robert Endre Tarjan. Unique maximum matching algorithms. *J. Algorithms*, 40(2):159–183, 2001.

- [16] Zvi Galil and Giuseppe F. Italiano. Maintaining the 3-edge-connected components of a graph on-line. *SIAM J. Comput.*, 22(1):11–28, 1993.
- [17] Dora Giammarresi and Giuseppe F. Italiano. Decremental 2- and 3-connectivity on planar graphs. *Algorithmica*, 16(3):263–287, 1996.
- [18] David Gibb, Bruce M. Kapron, Valerie King, and Nolan Thorn. Dynamic graph connectivity with improved worst case update time and sublinear space. *CoRR*, abs/1509.06464, 2015.
- [19] Michael T Goodrich and Michael Mitzenmacher. Invertible bloom lookup tables. In *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 792–799. IEEE, 2011.
- [20] Monika Rauch Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, 1999.
- [21] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, July 2001.
- [22] Jacob Holm and Eva Rotenberg. Good r-divisions imply optimal amortised decremental biconnectivity. *CoRR*, abs/1808.02568, 2018.
- [23] Jacob Holm, Eva Rotenberg, and Mikkel Thorup. Dynamic bridge-finding in $\tilde{O}(\log^2 n)$ amortized time. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 35–52, 2018.
- [24] Shang-En Huang, Dawei Huang, Tsvi Kopelowitz, and Seth Pettie. Fully dynamic connectivity in $O(\log n(\log \log n)^2)$ amortized expected time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 510–520, 2017.
- [25] Wenyu Jin and Xiaorui Sun. Fully dynamic c-edge connectivity in subpolynomial time. *CoRR*, abs/2004.07650, 2020.
- [26] Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1131–1142, 2013.
- [27] Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '13*, page 1131–1142, USA, 2013. Society for Industrial and Applied Mathematics.
- [28] David R. Karger. Random sampling in cut, flow, and network design problems. *Math. Oper. Res.*, 24(2):383–413, 1999.
- [29] Casper Kejlberg-Rasmussen, Tsvi Kopelowitz, Seth Pettie, and Mikkel Thorup. Faster worst case deterministic dynamic connectivity. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 53:1–53:15, 2016.

- [30] Jakub Lacki and Piotr Sankowski. Optimal decremental connectivity in planar graphs. *Theory Comput. Syst.*, 61(4):1037–1053, 2017.
- [31] Hiroshi Nagamochi and Toshihide Ibaraki. A linear-time algorithm for finding a sparse k -connected spanning subgraph of a k -connected graph. *Algorithmica*, 7(5&6):583–596, 1992.
- [32] Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 950–961, 2017.
- [33] Mihai Pătraşcu and Erik D. Demaine. Lower bounds for dynamic connectivity. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing, STOC '04*, page 546–553, New York, NY, USA, 2004. Association for Computing Machinery.
- [34] Mihai Pătraşcu and Mikkel Thorup. Don't rush into a union: take time to find your roots. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 559–568, 2011.
- [35] Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, April 1975.
- [36] Mikkel Thorup. Decremental dynamic connectivity. *Journal of Algorithms*, 33(2):229 – 243, 1999.
- [37] Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 343–350, 2000.
- [38] Mikkel Thorup. Fully-dynamic min-cut. *Comb.*, 27(1):91–127, 2007.
- [39] Mikkel Thorup. $\text{Sample}(x) = (a * x \leq t)$ is a distinguisher with probability $1/8$. *SIAM J. Comput.*, 47(6):2510–2526, 2018.
- [40] Zhengyu Wang. An improved randomized data structure for dynamic graph connectivity. *CoRR*, abs/1510.04590, 2015.
- [41] Christian Wulff-Nilsen. Faster deterministic fully-dynamic graph connectivity. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1757–1769. SIAM, 2013.

Appendix F

k -Edge Connected Components and Minimum Degree

k -Edge Connected Components and Minimum Degree

Anders Aamand Peter M. R. Rasmussen Mikkel Thorup

July 6, 2021

Abstract

A k -edge connected component of a graph is a maximal k -edge connected induced subgraph. Let $k > 1$ be an integer. We prove that the number of k -edge connected components of a simple graph on n vertices with minimum degree $(2 + \varepsilon)(k - 1)$, $\varepsilon \geq 1/(k - 1)$, is at most $\frac{10n}{\varepsilon k}$. We further prove that this bound is tight in two ways. First, for every $k > 1$ and $\varepsilon \geq 1/(k - 1)$ there exist a simple graph on n vertices with minimum degree $(2 + \varepsilon)(k - 1)$ and at least $\frac{n}{3\varepsilon k}$ k -edge connected components. Hence, the bound is asymptotically tight in ε and k . Second, the limit at $2(k - 1)$ is significant since for every $0 < \gamma < 1$ there exists a simple graph on n vertices with minimum degree $2(k - 1)$ and at least γn k -edge connected components, while every simple graph on n vertices with minimum degree $2k - 1$ has at most $\frac{2k-1}{3k-1}n$ k -edge connected components.

1 Introduction

Throughout this paper, $G = (V, E)$ denotes an undirected simple graph with vertex set V and edge set E . A central notion in graph theory is the notion of k -edge connectivity. Two distinct vertices u, v of a graph G are *k -edge connected* if there exist k pairwise edge-disjoint paths connecting them. Equivalently, u and v are k -edge connected if they cannot be disconnected by removing $< k$ edges from G . A graph G is *k -edge connected* if every pair of distinct vertices of G are k -edge connected. Equivalently, a graph G is k -edge connected if after the removal of any set of $< k$ edges, G is still connected. Note that by this definitions, the trivial graph is k -edge connected.

Our main objects of study are the *k -edge connected components* of G . We shall define a k -edge connected components of a graph G to be a maximal k -edge connected induced subgraph of G , i.e., H is a k -edge connected component of G if $H = G[U]$ for some vertex subset $U \subset V$ and for every strictly larger vertex subset, $U \subsetneq U' \subseteq V$, $H' = G[U']$ is not k -edge connected. Note that each vertex v of G is contained in a k -edge connected component since v by itself constitutes a k -edge connected induced subgraph of G . We call a k -edge connected component consisting of a single vertex a *trivial k -edge connected*

component. The k -edge connected components of a graph G partitions the vertices of G .

Remark 1. *In the literature, the k -edge connected components of a graph sometimes refers instead to the equivalence classes of vertices generated by k -edge connectivity, i.e., considering two vertices equivalent if they are k -edge connected. It is important to keep in mind that whenever $k > 2$, these two views of k -edge connected components are not equivalent.*

Whenever a graph with a fixed number of vertices contains sufficiently many edges, non-trivial k -edge connected components begin to emerge. This is the content of a classical result by Mader.

Theorem 1 (Mader [1]). *Let k and $n > k$ be positive integers. Every graph on n vertices and strictly more than $(k - 1)(n - k/2)$ edges contains a non-trivial k -edge connected component. Furthermore, there exists a graph on n vertices and $(k - 1)(n - k/2)$ edges containing only trivial k -edge connected components.*

In this exposition, we consider a related question, establishing a connection between the minimum degree and the number of k -edge connected components of a graph. Let $k > 1$ be an integer and G be a graph on n vertices. Our main result, stated as Theorem 2, is that if G has minimum degree $(2 + \varepsilon)(k - 1)$ for any $\varepsilon > 0$, then G contains at most $\frac{10n}{\varepsilon k}$ distinct k -edge connected components.

We further show that this bound is tight in two ways. First, the upper bound is tight asymptotically since for every integer $k > 1$ and $\varepsilon \geq 1/(k - 1)$ there exists a graph G with some n vertices and at least $\frac{n}{3\varepsilon k}$ distinct k -edge connected components, proven in Theorem 5. Second, the limit at $2(k - 1)$ is natural in the sense that for every $0 < \gamma < 1$ there exists a simple graph on n vertices with minimum degree $2(k - 1)$ and at least γn k -edge connected components, proven in Proposition 3. Furthermore, every graph on n vertices with minimum degree $2k - 1$ has at most $\frac{2k-1}{3k-1}n$ k -edge connected components, proven in Proposition 4.

Remark 2. *The main focus of this paper is to establish the asymptotic number of k -edge connected components as a function of the minimum degree and furthermore to establish a natural limit at minimum degree $2(k - 1)$. Thus, the constants stated in the theorems of this paper are almost certainly not optimal. We leave it as an open problem to improve upon them.*

2 Preliminaries

For every $n \in \mathbb{N}$, we denote by $[n]$ the set $\{1, \dots, n\}$.

A *cut* of a graph $G = (V, E)$ is formally a partition of the vertices into disjoint subsets $V = V_1 \cup V_2$. However, we shall often identify a cut with its *cut-set*, the edges of G connecting the two sides of the cut, V_1 and V_2 . In this vein, we shall call a cut non-trivial if the cut-set is non-empty and refer to the *size* of a cut by the size of its cut-set. Furthermore, we shall often discuss cuts

of a connected component C of G . When doing so, we only care about the partition of the vertices of C and the edges of C crossing the partition. Lastly, a *minimum cut* of a graph G or a connected component of G is a non-trivial cut where the cut-set has minimal size, i.e., there is no non-trivial cut with a smaller cut-set.

3 Bound on k -Edge Connected Components

In this section, we prove our main result, an upper bound on the number of k -edge connected components of a graph by its minimum degree. First, two simple lemmas.

Lemma 2. *Let $G = (V, E)$ be a graph and let $A \subset V$ be a non-empty subset of vertices. Suppose that every vertex of A has degree at least δ and that strictly less than δ edges leave A . Then $|A| \geq \delta + 1$.*

Proof. We prove the contrapositive statement. Suppose that $1 \leq |A| = t \leq \delta$. Then every vertex of A has at least $\delta - t + 1$ incident edges leaving A . Thus, at least $t(\delta - t + 1) \geq \delta$ edges leave A . \square

Lemma 3. *Let G be a graph on n vertices and q k -edge connected components. Then G contains at most $(k - 1)(q - 1)$ edges that connect distinct k -edge connected components of G .*

Proof. Remove non-trivial cuts of G of size $< k$ until no such cut remains. What is left is exactly the k -edge connected components of G . The removal of a non-trivial cut increases the number of components by at least one, so at most $q - 1$ cuts were removed. Furthermore, every cut contained at most $k - 1$ edges. Hence, at most $(k - 1)(q - 1)$ edges were removed from the graph and none of the remaining edges connect distinct k -edge connected components of G . \square

With this at hand, we proceed with the main theorem of the section.

Theorem 4. *Let $k > 1$ be an integer and $\varepsilon > 0$. Every graph on n vertices with minimum degree $(2 + \varepsilon)(k - 1)$ has at most $\frac{10n}{\varepsilon k}$ distinct k -edge connected components.*

Proof. Let G be a graph on n vertices of minimum degree $\delta \geq (2 + \varepsilon)(k - 1)$. We may assume $n > 2k \geq 4$.

Consider the following process of repeatedly deleting non-trivial cuts of size $< k$ from G until no such cuts remain. Let $G_0 = G$ and for $i \geq 0$, if G_i contains a vertex v of minimum positive degree and the degree of v is strictly less than k , remove the edges incident to v in G_i to obtain the graph G_{i+1} ; else, if G_i contains a component with a minimum cut with cut-set S of size $0 < |S| < k$, let $G_{i+1} = G_i \setminus S$; and finally, if no such cut exists, then all components of G_i are k -edge connected and the process terminates. We call the removal of all edges incident to a single vertex v , corresponding to the first case, a *vertex cut* and the vertex v the *center* of the vertex cut. Furthermore, the removal of a cut

of a component, corresponding to the second case, is called a *non-vertex cut*. When the process terminates after some t steps, the remaining components, the components of the graph G_t , are the k -edge connected components of G .

Claim 1. Strictly less than $\left\lfloor \frac{n}{k+1} \right\rfloor$ non-vertex cuts occur in the process.

Proof. The key observation is the following. Suppose that a non-vertex cut occurs during the process, dividing a component C of G_i into two new components C_1 and C_2 in G_{i+1} by the removal of a cut of size $< k$. Note that the component may only break into two new components since non-vertex cuts are minimum cuts. Now, since no vertex cut occurred, every vertex of C_1 and C_2 has degree $\geq k$ in G_i . Hence, $|C_1|, |C_2| \geq k+1$ by Lemma 1 as strictly less than k edges leave C_1 and C_2 , respectively, in G_i .

We now proceed by strong induction on n . Define $a(n) = \left\lfloor \frac{n}{k+1} \right\rfloor - 1$ and let $b(n)$ denote the maximum number of non-vertex cuts that may occur for a graph with n vertices. By the observation above, it immediately follows that $a(n) = 0$ for $k < n \leq 2k+1$ and $a(n) = 1$ for $2k+2 \leq n \leq 3k+2$. Thus, $b(n) \leq a(n)$ for $n \leq 3k+2$ as desired.

Let G be a graph on $n > 3k+2$ vertices and suppose that $b(m) \leq a(m)$ for all $m < n$. Running the above process on G , there are three cases. If the process terminates immediately, there is nothing to prove. If the process starts with a vertex cut, then G experiences no more than $b(n-1) \leq a(n-1) \leq a(n)$ non-vertex cuts and we are done. Finally, if the process starts with a non-vertex cut, G is divided into two components C_1 and C_2 each on at least $k+1$ vertices. The number of additional non-vertex cuts occurring in the process is now at most $b(|C_1|) + b(|C_2|)$. It follows by the induction hypothesis that the total number of non-vertex cuts is at most

$$\begin{aligned} 1 + b(|C_1|) + b(|C_2|) &\leq 1 + \left\lfloor \frac{|C_1|}{k+1} \right\rfloor - 1 + \left\lfloor \frac{|C_2|}{k+1} \right\rfloor - 1 \\ &\leq \left\lfloor \frac{|C_1| + |C_2|}{k+1} \right\rfloor - 1 \\ &= a(n). \end{aligned}$$

The conclusion follows. ■

Suppose that $C = \{v\}$ is a trivial k -edge connected component of G , and let $i \geq 1$ be such that v is an isolated vertex of G_i but not of G_{i-1} . The vertex v has degree $< k$ in G_{i-1} as it can be isolated by removing $< k$ edges. Hence, G_i was produced by a vertex cut. It follows that any trivial k -edge connected component of G is isolated during a vertex cut. This observation along with Claim 1 yields the following.

Claim 2. The graph G contains strictly less than $\frac{2n}{\varepsilon(k+1)}$ trivial k -edge connected components.

Proof. For $i \geq 0$ and v a vertex of G , let x_v^i denote the number of edges of G incident to v that are not present in G_i . Furthermore, let W_i be the set of non-isolated vertices of G_i and $y_i = \sum_{v \in W_i} x_v^i$. Finally, denote by I_i the set of vertices of G that are isolated in G_i but not in G_{i-1} . Initially, $y_0 = 0$ and $x_v^0 = 0$ for every vertex v .

Let $i \geq 1$. If a non-vertex cut occurs as step i , at most $k-1$ edges are removed from G_{i-1} , so $y_i \leq y_{i-1} + 2(k-1)$. Furthermore, $|I_i| = 0$ by the remark preceding the claim. If a vertex cut occurs, at most $k-1$ edges are removed and the vertices of I_i are isolated, so $y_i \leq y_{i-1} - \sum_{v \in I_i} x_v^{i-1} + k-1$. Since each $v \in I_i$ has degree at most $k-1$ in G_{i-1} ,

$$x_v^{i-1} \geq (2 + \varepsilon)(k-1) - (k-1) \geq (1 + \varepsilon)(k-1).$$

Thus,

$$y_i \leq y_{i-1} - |I_i|(1 + \varepsilon)(k-1) + k-1 \leq y_{i-1} - |I_i|\varepsilon(k-1),$$

where the last inequality follows as $|I_i| \geq 1$.

To sum up, for every non-vertex cut y_i is increased by at most $2(k-1)$, and for every vertex isolated, y_i is decreased by at least $\varepsilon(k-1)$. Since y_i is never negative, it follows that the number of vertices isolated cannot exceed $2(k-1)/(\varepsilon(k-1)) = 2/\varepsilon$ times the number of non-vertex cuts. The conclusion follows from Claim 1. \blacksquare

Let \mathcal{C} denote the k -edge connected components of G . Partition \mathcal{C} as $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3$, where \mathcal{C}_1 contains the trivial k -edge connected components of G ; \mathcal{C}_2 contains every component $C \in \mathcal{C}$ with $k+1 \leq |C| \leq (2 + \varepsilon)(k-1)$; and \mathcal{C}_3 contains every component $C \in \mathcal{C}$ with $|C| > (2 + \varepsilon)(k-1)$. This is indeed a partition since no k -edge connected (simple) graph contains strictly between 1 and $k+1$ vertices. From Claim 2 it follows that $|\mathcal{C}_1| < \frac{2n}{\varepsilon(k+1)}$. Furthermore, by the size of the components, it trivially holds that $|\mathcal{C}_3| \leq \frac{n}{(1+\varepsilon/2)(k-1)} \leq \frac{4n}{\varepsilon k}$. Thus, it only remains to bound the size of \mathcal{C}_2 . To that end, denote by D the vertices of G that have degree $< (1 + \varepsilon/2)(k-1)$ in G_t . Each of them is incident to $\geq (1 + \varepsilon/2)(k-1)$ edges of G that are not present in G_t . By Lemma 2, at most $(k-1)(n-1)$ edges of G are not present in G_t . It follows that

$$|D| \leq \frac{2(k-1)(n-1)}{(1 + \varepsilon/2)(k-1)} \leq \frac{2n}{1 + \varepsilon/2} \leq 4n/\varepsilon.$$

For every $C \in \mathcal{C}_2$, every vertex of C is contained in D . Hence, as each $C \in \mathcal{C}_2$ has size at least $k+1$, $|\mathcal{C}_2| \leq \frac{4n}{\varepsilon(k+1)}$.

In conclusion, G has at most

$$|\mathcal{C}_1| + |\mathcal{C}_2| + |\mathcal{C}_3| \leq \frac{2n}{\varepsilon(k+1)} + \frac{4n}{\varepsilon(k+1)} + \frac{4n}{\varepsilon k} \leq \frac{10n}{\varepsilon k}$$

distinct k -edge connected components. \square

4 On Tightness

The limit at minimum degree $2(k-1)$ proposed by the theorem is in fact tight in the following sense. For fixed $k > 1$, there is a constant $\delta < 1$ such that every graph on some n vertices with minimum degree $2k-1$ has at most δn distinct k -edge connected components. However, there is no such constant for graphs of minimum degree $2(k-1)$. This is laid out in the following two propositions.

Proposition 5. *If G has minimum degree $2k-1$, the number of k -edge connected components is at most $\frac{2k-1}{3k-1}n$.*

Proof. Let G be a graph with minimum degree $2k-1$. Denote by q the number of k -edge connected components of G and by ℓ the number of trivial k -edge connected components of G . Every trivial k -edge connected component is adjacent to at least $2k-1$ edges of G that connect distinct k -edge connected components of G . Since each such edge connects at most two trivial k -edge connected components, it follows by Lemma 2 that $\ell(2k-1) \leq 2(k-1)(q-1)$. Hence, $\ell \leq q(1-1/(2k-1))$. Furthermore, each non-trivial k -edge connected component contains at least $k+1$ vertices, so $q \leq \ell + (n-\ell)/(k+1)$. Combining these inequalities, we find that

$$q \leq \left(1 - \frac{1}{2k-1}\right) \left(1 - \frac{1}{k+1}\right) + \frac{n}{k+1}.$$

Rearranging terms yields $q \leq \frac{2k-1}{3k-1}n$ as desired. \square

Proposition 6. *For every integer $k > 1$ and real $\gamma \in (0, 1)$ there exists $n \in \mathbb{N}$ and a graph G on n vertices satisfying that G has minimum degree $2(k-1)$ and the number of k -edge connected components of G is at least γn .*

Proof. We construct the graph G_s , illustrated in Fig. 1. Let C_1, \dots, C_k and D_1, \dots, D_k be copies of K_{2k-1} , the complete graph on $2k-1$ vertices, and for each $i \in [k]$, let c_i be a distinguished vertex of C_i and d_i a distinguished vertex of D_i . Furthermore, for each $i \in [k]$ and $j \in [s]$, add an additional vertex $v_{i,j}$ to G_s . Denote by V_0 the set $\{c_i\}_{i=1}^k$, by V_{s+1} the set $\{d_i\}_{i=1}^k$, and for each $j \in [s]$, denote by V_j the set $\{v_{i,j}\}_{i=1}^k$. Introduce edges such that for every $j \in [s+1]$, the induced subgraph on $V_{j-1} \cup V_j$ is a $(k-1)$ -regular bipartite graph with independent sets V_{j-1} and V_j .

Now, G_s contains $n = 2k(2k-1) + sk$ vertices, and every vertex of G_s has degree at least $2(k-1)$. We proceed to count the number of k -edge connected components of G_s . First, observe that each of the subgraphs C_1, \dots, C_k and D_1, \dots, D_k is a k -edge connected component of $G_{s,t}$ since it has $k-1$ edges on its boundary. Second, removing these components from G_s , it is clear that each vertex of V_1 or V_k is by itself a k -edge connected component of G_s since all these vertices now have degree $k-1$. Finally, continuing to remove k -edge connected components in this manner, it is clear that each vertex of $\{v_{i,j}\}_{i \in [k], j \in [s]}$ is a k -edge connected component of G_s . Hence, the number of k -edge connected components of G_s is $(s+2)k$. Thus, G_s contains γn k -edge connected

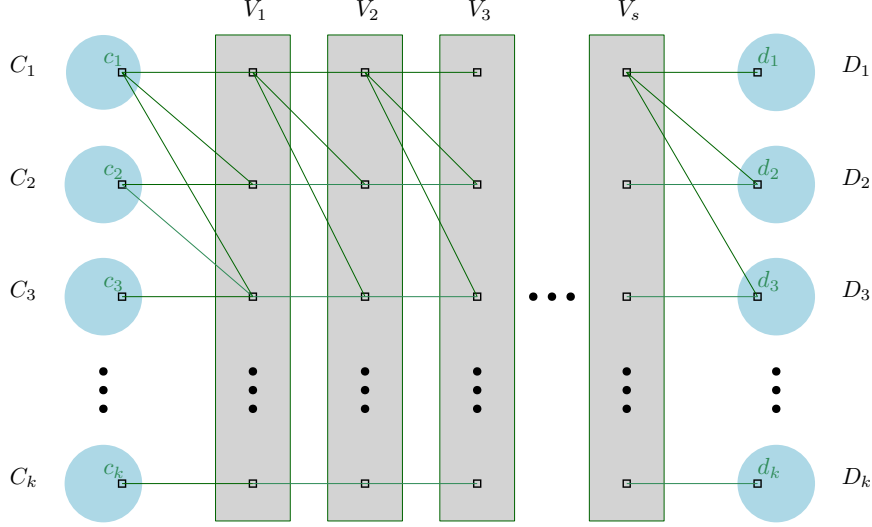


Figure 1: An illustration of the graph $G_{s,t}$ from the proof of Proposition 4.

components, where

$$\gamma = \frac{(s+2)k}{2k(2k-1) + sk} = \frac{s+2}{4k+s-2}.$$

Since $\gamma \rightarrow 1$ as $s \rightarrow \infty$, the conclusion follows. \square

Next, the result of Theorem 2 is asymptotically tight in terms of ε and k .

Theorem 7. *For every integer $k > 1$ and every $\varepsilon \geq 1/(k-1)$ there exists $n \in \mathbb{N}$ and a graph G with n vertices, minimum degree $\geq (2+\varepsilon)(k-1)$, and at least $\frac{n}{3\varepsilon k}$ distinct k -edge connected components.*

Proof. For each $s \geq 2$, we construct a graph H_s similar to the graph G_s from the proof of Proposition 4. Let the sequence $(n_i)_{i \geq 0}$ be given by $n_i = \lfloor (1+\varepsilon)^i k \rfloor$. Starting from the empty graph, add the following to H_s . Let C_1, \dots, C_{n_0} and D_1, \dots, D_{n_s} be copies of $K_{(2+\varepsilon)k}$; for each $i \in [n_0]$, let c_i be a distinguished vertex of C_i ; and for each $i \in [n_s]$, let d_i be a distinguished vertex of D_i . Furthermore, for each $j \in [s-1]$ and $i \in [n_j]$, let $v_{i,j}$ be a vertex. Denote by V_0 the set $\{c_i\}_{i \in [n_0]}$, by V_s the set $\{d_i\}_{i \in [n_s]}$, and for each $j \in [s-1]$, denote by V_j the set $\{v_{i,j}\}_{i \in [n_j]}$.

Insert edges as follows. For each $j \in [s]$, the induced subgraph of $V_{j-1} \cup V_j$ in H_s is a bipartite graph with independent sets V_{j-1} and V_j such that every vertex of V_j has degree $k-1$ and every vertex of V_{j-1} has degree at least $(1+\varepsilon)(k-1)$. Such a bipartite graph is always possible to construct since $|V_j| = n_j \geq (1+\varepsilon)n_i = (1+\varepsilon)|V_{j-1}|$. It is then clear that every vertex of H_s has degree at least $(2+\varepsilon)(k-1)$.

Now, let n denote the number of vertices of G_s . Then

$$\begin{aligned} n &= (n_0 + n_s)(2 + \varepsilon)k + \sum_{i=1}^{s-1} n_i \\ &\leq k \left((1 + (1 + \varepsilon)^s)(2 + \varepsilon)k + (1 + \varepsilon) \frac{(1 + \varepsilon)^{s-1} - 1}{\varepsilon} \right) \\ &= k \left((1 + \varepsilon)^s ((2 + \varepsilon)k + 1/\varepsilon) + A \right), \end{aligned}$$

where $A = (2 + \varepsilon)k - (1 + \varepsilon)/\varepsilon$. Furthermore, denote by q the number of k -edge connected components of H_s . For every $i \in [n_s]$, D_i is a k -edge connected component of H_s since it has $k - 1$ edges on its boundary. Removing D_i from H_s for every $i \in [n_s]$, it becomes clear that every $v \in V_{s-1}$ is a trivial k -edge connected components since it now has degree $k - 1$. Continuing to remove one layer at a time, it is clear that for every $j \in [s - 1]$, every vertex of V_j is a trivial k -edge connected component. Finally, it follows that C_i is a k -edge connected component for every $i \in [n_0]$, since removing each of the previously mentioned k -edge components from G leaves each C_i isolated. In conclusion,

$$q = \sum_{i=0}^s n_i \geq k \sum_{i=0}^s (1 + \varepsilon)^i - s - 1 = k \frac{(1 + \varepsilon)^{s+1}}{\varepsilon} - s - 1.$$

Combining the estimates for n and q ,

$$\begin{aligned} \frac{n}{q} &\leq \frac{k \left((1 + \varepsilon)^s ((2 + \varepsilon)k + 1/\varepsilon) + A \right)}{k \frac{(1 + \varepsilon)^{s+1}}{\varepsilon} - s - 1} \\ &= \frac{(2 + \varepsilon)\varepsilon k + 1 + \varepsilon A / (1 + \varepsilon)^s}{1 + \varepsilon - \frac{\varepsilon(s-1)}{(1 + \varepsilon)^s k}}. \end{aligned}$$

As $s \rightarrow \infty$, the last expression converges to

$$\frac{(2 + \varepsilon)\varepsilon k + 1}{1 + \varepsilon} < 3\varepsilon k.$$

Hence, for s sufficiently large, $n/q \leq 3\varepsilon k$, which yields the conclusion. \square

References

- [1] MADER, W. Minimalen-fach kantenzusammenhängende graphen. *Mathematische Annalen* 191 (03 1971), 21–28.

Appendix G

Classifying Convex Bodies by Their Contact and Intersection Graphs

Classifying Convex Bodies by Their Contact and Intersection Graphs

Anders Aamand  

BARC, University of Copenhagen, Denmark

Mikkel Abrahamsen  

BARC, University of Copenhagen, Denmark

Jakob Bæk Tejs Knudsen  

BARC, University of Copenhagen, Denmark

Peter Michael Reichstein Rasmussen  

BARC, University of Copenhagen, Denmark

Abstract

Let A be a convex body in the plane and A_1, \dots, A_n be translates of A . Such translates give rise to an *intersection graph* of A , $G = (V, E)$, with vertices $V = \{1, \dots, n\}$ and edges $E = \{uv \mid A_u \cap A_v \neq \emptyset\}$. The subgraph $G' = (V, E')$ satisfying that $E' \subset E$ is the set of edges uv for which the interiors of A_u and A_v are disjoint is a *unit distance graph* of A . If furthermore $G' = G$, i.e., if the interiors of A_u and A_v are disjoint whenever $u \neq v$, then G is a *contact graph* of A .

In this paper, we study which pairs of convex bodies have the same contact, unit distance, or intersection graphs. We say that two convex bodies A and B are equivalent if there exists a linear transformation B' of B such that for any slope, the longest line segments with that slope contained in A and B' , respectively, are equally long. For a broad class of convex bodies, including all strictly convex bodies and linear transformations of regular polygons, we show that the contact graphs of A and B are the same if and only if A and B are equivalent. We prove the same statement for unit distance and intersection graphs.

2012 ACM Subject Classification Theory of computation \rightarrow Computational geometry; Mathematics of computing \rightarrow Graph theory; Mathematics of computing \rightarrow Discrete mathematics

Keywords and phrases convex body, contact graph, intersection graph

Digital Object Identifier 10.4230/LIPIcs.SoCG.2021.3

Related Version *Full Version*: <https://arxiv.org/abs/1902.01732> [1]

Funding The authors are part of BARC, Basic Algorithms Research Copenhagen, supported by the VILLUM Foundation grant 16582.

Acknowledgements We thank Tillmann Miltzow for asking when the translates of two different convex bodies induce the same intersection graphs which inspired us to work on these problems.

1 Introduction

Consider a convex body A , i.e., a convex, compact region of the plane with non-empty interior, and let $\mathcal{A} = \{A_1, \dots, A_n\}$ be a set of n translates of A . Then \mathcal{A} gives rise to an *intersection graph* $G = (V, E)$, where $V = \{1, \dots, n\}$ and $E = \{uv \mid A_u \cap A_v \neq \emptyset\}$, and a *unit distance graph* $G' = (V, E')$, where $uv \in E'$ if and only if $uv \in E$ and A_u and A_v have disjoint interiors. In the special case that $G = G'$ (i.e., the convex bodies of \mathcal{A} have pairwise disjoint interiors), we say that G is a *contact graph* (also known as a *touch graph* or *tangency graph*). Thus, A defines three classes of graphs, namely the intersection graphs $I(A)$, the unit distance graphs $U(A)$, and the contact graphs $C(A)$ of translates of A .



© Anders Aamand, Mikkel Abrahamsen, Jakob Bæk Tejs Knudsen, and Peter Michael Reichstein Rasmussen;

licensed under Creative Commons License CC-BY 4.0

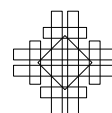
37th International Symposium on Computational Geometry (SoCG 2021).

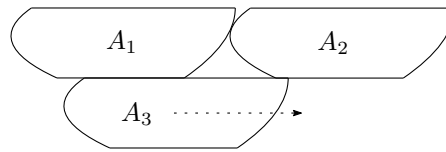
Editors: Kevin Buchin and Éric Colin de Verdière; Article No. 3; pp. 3:1–3:16



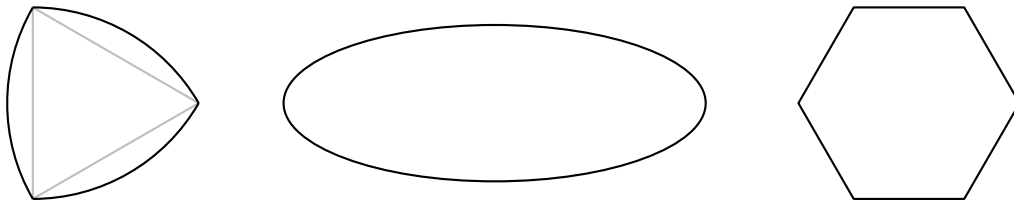
Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany





■ **Figure 1** Translates of a convex body not having the URTC property. The disk A_3 can “slide” along A_1 and A_2 .



■ **Figure 2** Reuleaux triangle (left), ellipse (middle), and regular hexagon (right).

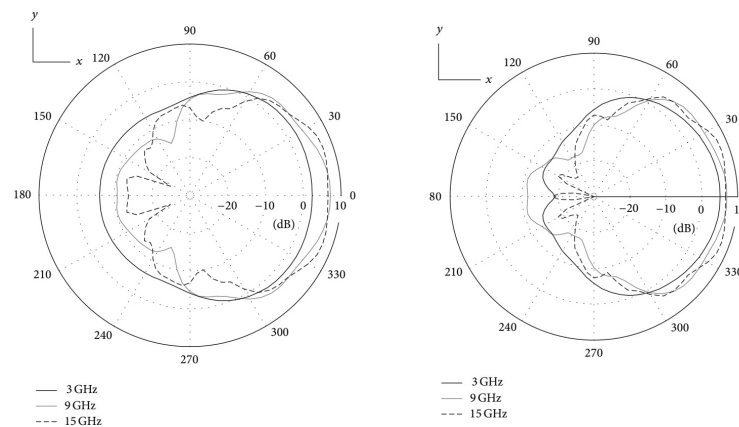
The study of intersection graphs has been an active research area in discrete and computational geometry for the past three decades. For instance, numerous papers consider the problem of solving classical graph problems efficiently on various classes of geometric intersection graphs; see Section 1.1 for some references. Meanwhile, the study of contact graphs of translates of a convex body has much older roots. It is closely related to the packings of such a body, which has a very long and rich history in mathematics going back (at least) to the seventeenth century, where research on the packings of circles of varying and constant radii was conducted and Kepler famously conjectured upon a 3-dimensional counterpart of such problems, the packing of spheres.

In this paper we investigate the question of when two convex bodies A and B give rise to the same classes of graphs. We restrict ourselves to convex bodies A that have the URTC property (*unique regular triangle constructibility*). This is the property that given two interior disjoint translates A_1, A_2 of A that touch, there are exactly two ways to place a third translate A_3 such that A_3 is interior disjoint from A_1 and A_2 , but touches both. Convex bodies with the URTC property include all linear transformations of regular polygons except squares and all strictly convex bodies [15]. Thus, almost all convex bodies in a measure theoretic sense have the property [15, 18, 30]. A convex body without the property must have a sufficiently long line segment on the boundary (to be made precise in Section 1.3); see Figure 1 for an example.

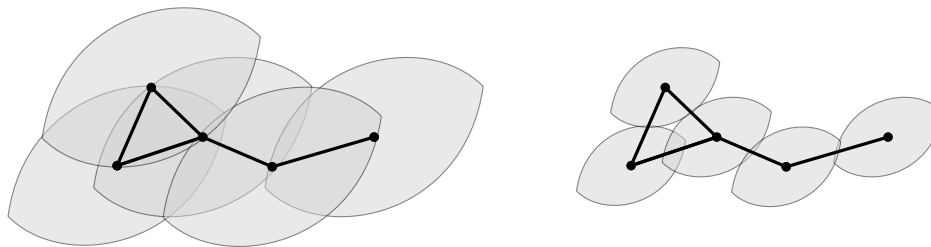
The main result of the paper is summarized in the following theorem.

► **Theorem 1.** *Let A and B be convex bodies with the URTC property. Then each of the identities $I(A) = I(B)$, $U(A) = U(B)$, and $C(A) = C(B)$ holds if and only if the following condition is satisfied: there is a linear transformation B' of B such that for any slope, the longest segments contained in A and B' , respectively, with that slope are equally long.*

If the condition from the theorem is satisfied, we say that A and B are *equivalent*. The length of the longest segment with a given slope contained in a convex body A is often called the *width* of A in the corresponding direction. A circle has constant width but there are other convex bodies of constant width, the simplest example being the Reuleaux triangle; see Figure 2. As an example it follows from the theorem that circles and Reuleaux triangles have the same contact, unit distance, and intersection graphs, which in turn are the same as those for ellipses (ellipses are linear transforms of circles). It also follows that these classes are different from those of regular hexagons.



■ **Figure 3** The strength of radiation in every direction and at various frequencies for two different transmitters described in [25]. In engineering circles, this known as the *radiation pattern*.



■ **Figure 4** When the reachable region of a device is symmetric and the devices are oriented in the same way, a communication network is the intersection graph of the reachable region scaled by $1/2$. Left: A network of five identical devices with the reachable regions shown. Right: The intersection graph of the reachable regions scaled by $1/2$.

1.1 Practical Implications

From a practical point of view, the research on intersection graphs is often motivated by the applicability of these graphs when modeling wireless communication networks and facility location problems. If a device is located at some point in the plane and is able to transmit to and receive from all other devices within some distance, then the devices can be represented as unit disks in such a way that two devices can communicate if and only if their disks overlap. Many highly-cited papers gave this motivation for studying unit disk intersection graphs [9, 14, 16, 21, 29] and it remains a motivation for new research [7, 12, 13, 24].

However, it is in general not the case that a transmitter emits an equally strong signal in all directions. For a real-world example of how the signal strength may vary in different directions; see Figure 3. If the networks that can be made with devices of a given type are not the unit disk intersection graphs, the algorithms for unit disk graphs cannot be expected to work when applied to the actual networks. It is therefore necessary to study how the possible networks that can be made with devices of different types depend on the radiation pattern of the devices. See Figure 4 for a demonstration of the connection between communication networks of a device with a non-circular radiation pattern and intersection graphs of the corresponding convex body.

1.2 Other Related Work

An important notion in the area of contact graphs is that of the *Hadwiger number* of a body K , which is the maximum possible number of pairwise interior-disjoint translates K_i of K that each touch but do not overlap K . The Hadwiger number of K is thus the maximum degree of a contact graph of translates of K . In the plane, the Hadwiger number is 8 for parallelograms and 6 for all other convex bodies. We refer the reader to the books and surveys by László and Gábor Fejes Tóth [28, 10] and Böröczky [3].

Another noteworthy result on contact graphs is the Circle Packing Theorem (also known as the Koebe–Andreev–Thurston Theorem): A graph is simple and planar if and only if it is the contact graph of some set of circular disks in the plane (the radii of which need not be equal). The result was proven by Koebe in 1935 [19] (see [11] for a streamlined, elementary proof). Schramm [26] generalized the circle packing theorem by showing that if a planar convex body with smooth boundary is assigned to each vertex in a planar graph, then the graph can be realized as a contact graph where each vertex is represented by a homothet (i.e., a scaled translation) of its assigned body.

Several papers have compared classes of intersection graphs of various geometric objects, see for instance [4, 6, 8, 17, 20]. Most of the results are inclusions between classes of intersection graphs of one-dimensional objects such as line segments and curves.

A survey by Swanepoel [27] summarizes results on minimum distance graphs and unit distance graphs in normed spaces, including bounds on the minimum/maximum degree, maximum number of edges, chromatic number, and independence number.

In the area of computational geometry, Müller, van Leeuwen, and van Leeuwen [23] gave sharp upper and lower bounds on the size of an integer grid used to represent an intersection graph of translates of a convex polygon with corners at rational coordinates. Their results imply that for any convex polygon R with rational corners, the problem of recognizing intersection graphs of translates of R is in NP. On the contrary, it is open whether recognition of unit disk intersection graphs in the Euclidean plane is in NP. Indeed, the problem is $\exists\mathbb{R}$ -complete (and thus in PSPACE), and using integers to represent the center coordinates and radii of the disks in some graphs requires exponentially many bits [5, 22].

Bonnet, Grelier, and Miltzow [2] showed how well-known algorithms for the clique problem in unit disk intersection graphs and disk intersection graphs can be adjusted to work for intersection graphs of translates or homothets of an arbitrary centrally symmetric convex body.

1.3 Preliminaries

We begin by defining some basic geometric concepts and terminology.

For a subset $A \subset \mathbb{R}^2$ of the plane we denote by A° the interior of A , that is,

$$A^\circ = \{x \in A \mid \exists \text{ open } U \subset \mathbb{R}^2 \text{ such that } \{x\} \subset U \subset A\}.$$

We say that A is a *convex body* if A is compact, convex, and has non-empty interior. We say that A is *symmetric* if whenever $x \in A$, then $-x \in A$. It is well-known that if A is a symmetric convex body, then the map $\|\cdot\|_A : \mathbb{R}^2 \rightarrow \mathbb{R}_{\geq 0}$ defined by

$$\|x\|_A = \inf\{\lambda \geq 0 \mid x \in \lambda A\},$$

is a norm. Moreover $A = \{x \in \mathbb{R}^2 \mid \|x\|_A \leq 1\}$ and $A^\circ = \{x \in \mathbb{R}^2 \mid \|x\|_A < 1\}$.

It follows from these properties that for translates $A_1 = A + v_1$ and $A_2 = A + v_2$ it holds that $A_1 \cap A_2 \neq \emptyset$ if and only if $\|v_1 - v_2\|_A \leq 2$ and $A_1^\circ \cap A_2^\circ \neq \emptyset$ if and only if $\|v_1 - v_2\|_A < 2$. This means that when studying contact, unit distance, and intersection

graphs of a symmetric convex body A , we can shift our viewpoint from translates of A to point sets in \mathbb{R}^2 and their $\|\cdot\|_A$ -distances: If $\mathcal{A} \subset \mathbb{R}^2$ is a set of points we define $I_A(\mathcal{A})$ and $U_A(\mathcal{A})$ to be the graphs with vertex set \mathcal{A} and edge sets $\{(x, y) \in \mathcal{A}^2 \mid x \neq y \text{ and } \|x - y\|_A \leq 2\}$ and $\{(x, y) \in \mathcal{A}^2 \mid \|x - y\|_A = 2\}$, respectively. Moreover, if for all distinct points $x, y \in \mathcal{A}$ it holds that $\|x - y\|_A \geq 2$, we say that \mathcal{A} is *compatible with A* and define $C_A(\mathcal{A})$ to be the graph with vertex set \mathcal{A} and edge set $\{(x, y) \in \mathcal{A}^2 \mid \|x - y\|_A = 2\}$. Then $I_A(\mathcal{A}), U_A(\mathcal{A})$, and $C_A(\mathcal{A})$, respectively, are isomorphic to the intersection, unit distance, and contact graph of A realized by the translates $(A + a)_{a \in \mathcal{A}}$. When studying contact, unit distance, and intersection graphs of a symmetric, convex body A we will view them as being induced by point sets rather than by translates of A .

We say that a (not necessarily symmetric) convex body A in the plane has the URTC property if the following holds: For any two interior disjoint translates of A , A_1 and A_2 , satisfying $A_1 \cap A_2 \neq \emptyset$, there exists precisely two vectors $v \in \mathbb{R}^2$ such that for $i \in \{1, 2\}$, $(A + v)^\circ \cap A_i^\circ = \emptyset$ but $(A + v) \cap A_i \neq \emptyset$. If A is symmetric, this amounts to saying that for any two points $v_1, v_2 \in \mathbb{R}^2$ with $\|v_1 - v_2\|_A = 2$, the set $\{v \in \mathbb{R}^2 \mid \|v - v_1\|_A = \|v - v_2\|_A = 2\}$ has size two. Gehér [15] proved that a symmetric convex body A has the URTC property if and only if the boundary ∂A does not contain a line segment of length more than 1 in the $\|\cdot\|_A$ -norm. See Figure 1 for an example of a convex body not having the URTC property.

A *drawing* of a graph $G \in I(A)$ as an intersection graph of a symmetric convex body A is a point set $\mathcal{A} \subset \mathbb{R}^2$ and a set of straight line segments \mathcal{L} such that $I_A(\mathcal{A})$ is isomorphic to G and \mathcal{L} is exactly the line segments between the points $u, v \in \mathcal{A}$ which are connected by an edge in G . We define a drawing of a graph G as a contact and unit distance graph similarly.

For a norm $\|\cdot\|$ on \mathbb{R}^2 and a line segment ℓ with endpoints a and b we will often write $\|\ell\| = \|ab\|$ instead of $\|a - b\|$. Also, if A is a symmetric convex body and $U, V \subset \mathbb{R}^2$, we define $d_A(U, V) := \inf\{\|uv\|_A \mid (u, v) \in U \times V\}$.

1.4 Structure and Techniques of the Paper

Establishing the sufficiency of the condition of Theorem 1, i.e., showing that if A and B are equivalent then $I(A) = I(B)$, $U(A) = U(B)$, and $C(A) = C(B)$, is relatively straightforward and has been deferred to the full version of the paper. It is also relatively easy to reduce Theorem 1 to the case where the convex bodies are symmetric so this too is deferred to the full version. When both A and B are symmetric, they are equivalent according to the condition of Theorem 1 if and only if they are linear transformations of each other.

Thus, left with the task of proving the necessity of the condition of Theorem 1 in the symmetric case, we proceed in two steps. First, in Section 2, we prove the following result, which for contact and unit distance graphs is a generalization of this direction of Theorem 1.

► **Theorem 2.** *Let A and B be symmetric convex bodies with the URTC property such that A is not a linear transformation of B . There exists a graph $G \in C(A)$ such that for all $H \in C(B)$ and all subgraphs $H' \subseteq H$, G is not isomorphic to H' . In particular $C(A) \setminus C(B) \neq \emptyset$.*

As we will also discuss in Section 2 the same result holds if $C(X)$ is replaced by $U(X)$ for $X \in \{A, B\}$ everywhere in the theorem above and the proof is identical.

The core idea in proving Theorem 2 is to consider a graph G satisfying that any drawing of G as a contact graph of A has certain structural properties. Concretely, we ensure that any drawing of G as a contact graph of A consists of many large hollow hexagons. In the interior of each hexagon, we force there to be a “bridge” of translates of A connecting the

sides of the hexagon. We show that if B is not a linear transformation of A , then the contact graph cannot be realized by translates of B if we make sufficiently many and sufficiently large hexagons with bridges of different slopes. See Figures 6 and 7 for illustrations.

To include intersection graphs, we proceed with the second step. In Section 3, we prove the following result which combined with Theorem 2 immediately yields the necessity of the condition of Theorem 1 for intersection graphs.

► **Theorem 3.** *Let A and B be symmetric convex bodies. If there exists a graph $G \in C(A)$ such that for all $H \in C(B)$ and all subgraphs $H' \subset H$, G is not isomorphic to H' , then $I(A) \neq I(B)$.*

This result holds for general symmetric convex bodies. An improvement of Theorem 2 to general symmetric convex bodies (not necessarily having the URTC property) would thus yield a version of Theorem 1 that also holds for general convex bodies.

In order to prove Theorem 3, we proceed as follows. For every positive integer k we construct a gadget $Q_k \in I(A)$ which contains as a subgraph a distinguished cycle $\alpha_k \subset Q_k$. We prove that in any drawing of Q_k as an intersection graph of translates of A , α_k is contained in a translation of the annulus $kA \setminus (k-1)A$ (here, $kA = \{ka \mid a \in A\}$ is the scaling of A by k). This allows us to view α_k as an upscaled copy of the boundary of A with a precision error decreasing in k . Similarly, in any drawing of the same gadget Q_k as an intersection graph of another body, B , the cycle α_k appears as an upscaled copy of B . The idea is then to simulate a contact graph $G \in C(A)$ using distinct copies of Q_k , where each copy plays the role of a single vertex in G . If A is not a linear transformation of B , we can choose G with the properties promised in Theorem 2. We are then able to prove that if we choose k sufficiently large (i.e., obtaining sufficiently high resemblance between α_k and an upscaled copy of A resp. B), then we can realize the simulation of G as an intersection graph using translates of A , but not using translates of B . This then implies $I(A) \neq I(B)$ as desired.

Beyond aiding us in the proof of our main theorem, we believe that this proof technique – the reduction from intersection to contact graphs – is of independent interest. It appears a novel approach with the potential to answer other questions on intersection graphs.

2 Contact and Unit Distance Graphs

In this section we prove Theorem 2. The proof for unit distance graphs is completely identical so we will merely provide a remark justifying this claim by the end of the section.

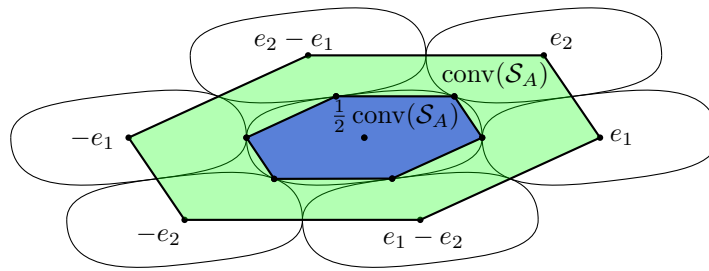
Throughout the section A and B will denote symmetric convex bodies. For $\theta \in [0, 2\pi)$ we define $e_A(\theta)$ to be the vector of argument θ and with $\|e_A(\theta)\|_A = 1$. We also define $\rho_A(\theta) = 2\|e_A(\theta)\|_2$. Then $\rho_A(\theta)$ can be thought of as the “diameter” of A in direction θ . One of our most important tools is the following lemma.

► **Lemma 4.** *Let A, B be symmetric convex bodies in \mathbb{R}^2 . If for every finite set $\Theta \subset [0, \pi)$ and for every $\varepsilon > 0$, there exists a linear map $T: \mathbb{R}^2 \rightarrow \mathbb{R}^2$ satisfying that $|\rho_{T(B)}(\theta) - \rho_A(\theta)| < \varepsilon$ for all $\theta \in \Theta$, then there exists a linear map $T: \mathbb{R}^2 \rightarrow \mathbb{R}^2$ with $T(B) = A$.*

Due to space limitations we have left out the proof, but it can be found in the full version.

We proceed to describe certain lattices which give rise to contact graphs that can only be realized in an essentially unique way. We start with the following definition.

► **Definition 5.** *Let $A \subset \mathbb{R}^2$ be a symmetric convex body, and $\|\cdot\|_A$ the associated norm. Let $e_1, e_2 \in \mathbb{R}^2$ be such that $\|e_1\|_A = \|e_2\|_A = \|e_1 - e_2\|_A = 2$. We define the lattice $\mathcal{L}_A(e_1, e_2) = \{a_1e_1 + a_2e_2 \mid (a_1, a_2) \in \mathbb{Z}^2\}$.*



■ **Figure 5** A symmetric convex body A and the sets $\frac{1}{2} \text{conv}(\mathcal{S}_A)$ and $\text{conv}(\mathcal{S}_A)$ (blue and green respectively) which satisfies $\frac{1}{2} \text{conv}(\mathcal{S}_A) \subseteq A \subseteq \text{conv}(\mathcal{S}_A)$.

The left hand side of figure Figure 6 illustrates the lattice structure. Let us assume that A has the URTC property and describe a few properties of the lattice $\mathcal{L}_A(e_1, e_2)$. After choosing e_1 with $\|e_1\|_A = 2$, there are precisely two vectors v with $\|v\|_A = \|v - e_1\|_A = 2$, using the URTC property. If one is v_2 the second is $e_1 - v_2$ so regardless how we choose e_2 we obtain the same lattice. Using the triangle inequality and the URTC property of A it is easily verified that for distinct $x, y \in \mathcal{L}_A(e_1, e_2)$, $\|x - y\|_A \geq 2$ with equality holding exactly if $x - y \in \mathcal{S}_A := \{e_1, e_2, e_2 - e_1, -e_1, -e_2, e_1 - e_2\}$. This implies that the contact graph $G_0 := C_A(\mathcal{L}_A(e_1, e_2))$ is in fact (isomorphic to) an infinite triangular grid.

Another useful fact is the following:

► **Lemma 6.** *With \mathcal{S}_A as above it holds that $\frac{1}{2} \text{conv}(\mathcal{S}_A) \subset A \subset \text{conv}(\mathcal{S}_A)$. Here $\text{conv}(\mathcal{S}_A)$ is the convex hull of \mathcal{S}_A . If in particular B is another symmetric convex body for which $\|e_1\|_B = \|e_2\|_B = \|e_1 - e_2\|_B = 2$, then for all $x \in \mathbb{R}^2$ it holds that $\frac{1}{2} \|x\|_A \leq \|x\|_B \leq 2 \|x\|_A$.*

Proof. See Figure 5. As $\frac{1}{2} \mathcal{S}_A \subset A$ and A is convex the first inclusion is clear. For the second inclusion we note that all points y on the hexagon connecting the points $e_1, e_2, e_2 - e_1, -e_1, -e_2, e_1 - e_2$ of \mathcal{S}_A in this order have $\|y\|_A \geq 1$ by the triangle inequality and so $A \subset \text{conv}(\mathcal{S}_A)$.

For the last statement of the lemma note that if $x \in \mathbb{R}^2$ then

$$\|x\|_B \geq \inf_{\lambda \geq 0} \{x \in \lambda \text{conv}(\mathcal{S}_B)\} = \inf_{\lambda \geq 0} \{x \in \lambda \text{conv}(\mathcal{S}_A)\} \geq \inf_{\lambda \geq 0} \{x \in 2\lambda A\} = \frac{1}{2} \|x\|_A,$$

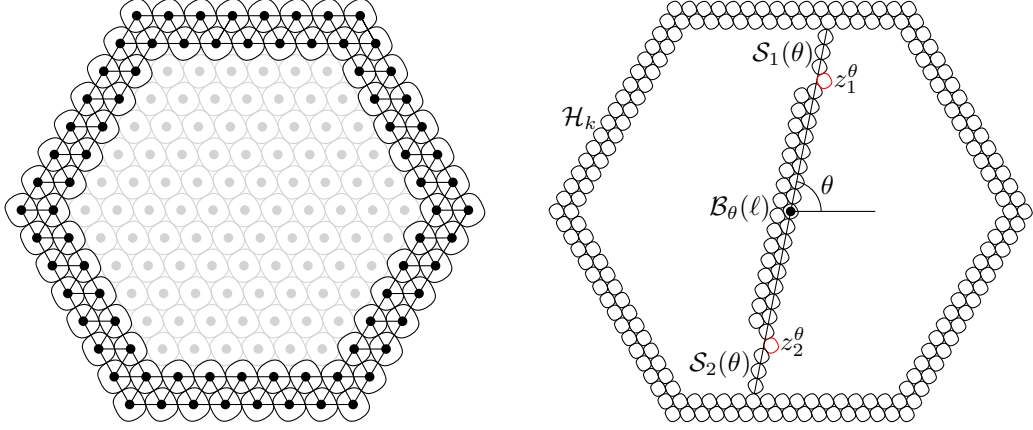
and similarly $\|x\|_A \geq \frac{1}{2} \|x\|_B$. ◀

► **Definition 7.** *We say that a graph $G = (V, E)$ is lattice unique if $|V| = n \geq 3$ and there exists an enumeration of its vertices v_1, \dots, v_n such that*

- *The vertex induced subgraph $G[v_1, v_2, v_3] \simeq K_3$ is a triangle.*
- *For $i > 3$ there exists distinct $j, k, l < i$ such that $G[v_j, v_k, v_l] \simeq K_3$ and both (v_i, v_j) and (v_i, v_k) are edges of G .*

Suppose that A is a symmetric convex body with the URTC property, that $\mathcal{A} \subset \mathbb{R}^2$ is compatible with A , and that $G = C_A(\mathcal{A})$ is lattice unique. Enumerate the points of $\mathcal{A} = \{v_1, \dots, v_n\}$ according to the definition of lattice uniqueness. Without loss of generality assume that $v_1 = 0$. Then the URTC property of A combined with the lattice uniqueness of G gives that v_4, \dots, v_n are uniquely determined from v_2 and v_3 and all contained in $\mathcal{L}_A(v_2, v_3)$. If moreover B is another convex body with the URTC property, $\mathcal{B} = \{v'_1, \dots, v'_n\} \subset \mathbb{R}^2$ has $v'_1 = 0$ and is compatible with B , and $C_B(\mathcal{B}) \simeq C_A(\mathcal{A})$ via the graph isomorphism $\varphi : v'_i \mapsto v_i$, then the linear map $T : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ defined by $T : a_1 v'_2 + a_2 v'_3 \mapsto a_1 v_2 + a_2 v_3$ satisfies that $T|_{\mathcal{B}} = \varphi$.

We will use this observation in the proof of Theorem 2 which we now provide.



■ **Figure 6** Left: The points of \mathcal{H}_6 along with the corresponding lattice unique subgraph $G_0[\mathcal{H}_6]$. Right: The attachment of the beam $\mathcal{B}_\theta(\ell)$.

Proof of Theorem 2. We choose $e_1, e_2 \in \mathbb{R}^2$ be such that $\|e_1\|_A = \|e_2\|_A = \|e_1 - e_2\|_A = 2$ and define the lattice $\mathcal{L} := \mathcal{L}_A(e_1, e_2)$. We also define the infinite graph $G_0 := C_A(\mathcal{L})$ which by the remarks following Definition 5 is isomorphic to the infinite triangular grid. Without loss of generality we can assume that e_1 and e_2 satisfy that $\|e_1\|_2 = \|e_2\|_2 = \|e_1 - e_2\|_2 = 2$, since there exists a non-singular linear transformation T such that $\|T(e_1)\|_2 = \|T(e_2)\|_2 = \|T(e_1) - T(e_2)\|_2 = 2$, and $C(A) = C(T(A))$. Note that in this setting we can use Lemma 6 to compare A to the disk of radius 1 and obtain $\frac{1}{2}\|x\|_2 \leq \|x\|_A \leq 2\|x\|_2$ for every $x \in \mathbb{R}^2$.

We will construct G by specifying a finite point set $\mathcal{A} \subset \mathbb{R}^2$ compatible with A and define $G = C_A(\mathcal{A})$. The construction of \mathcal{A} can be divided into several sub-constructions. We start by describing a hexagon of points \mathcal{H}_k for $k \in \mathbb{N}$ which satisfies that $C_A(\mathcal{H}_k)$ is lattice unique.

► **Construction 8** (\mathcal{H}_k). For an illustration of the construction see the left-hand side of Figure 6. For $x, y \in \mathcal{L}$ we write $d(x, y)$ for the distance between x and y in the graph G_0 , and for $k \in \mathbb{N}$ we define $\mathcal{H}_k = \{x \in \mathcal{L} \mid d(x, 0) \in \{k, k+1\}\}$.

Using that G_0 is the infinite triangular grid, it is easy to check that $G_0[\mathcal{H}_k]$ is a lattice unique graph by specifying an enumeration of its vertices satisfying the condition in Definition 7. Moreover, using that e_1 and e_2 satisfy $\|e_1\|_2 = \|e_2\|_2 = \|e_1 - e_2\|_2 = 2$ it follows that the points $\{x \in \mathcal{L} \mid d(x, 0) = k\} \subset \mathcal{H}_k$ lie on a regular hexagon H_k whose corners have Euclidean distance exactly $2k$ to the origin. In particular any point $p \in \mathcal{H}_k$ has $\|p\|_2 \geq \sqrt{3}k$, and thus $\|p\|_A \geq \frac{\sqrt{3}}{2}k$ by Lemma 6.

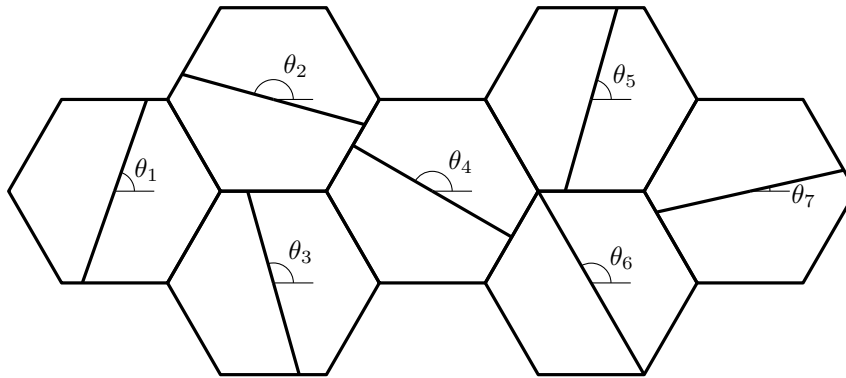
For a given $\theta \in [0, \pi)$ and $\ell \in \mathbb{N}$ we will construct a set of points $\mathcal{B}_\theta(\ell) \subset \mathbb{R}^2$ compatible with A which constitute a “beam” of argument θ :

► **Construction 9** ($\mathcal{B}_\theta(\ell)$). See Figure 6 (right). Let $e_\theta \in \mathbb{R}^2$ be the vector of argument θ with $\|e_\theta\|_A = 2$, and let $f_\theta \in \mathbb{R}^2$ be such that $\|f_\theta\|_A = \|f_\theta - e_\theta\|_A = 2$ (by the URTC property we have two choices for f_θ). For a given $\ell \in \mathbb{N}$ we define

$$\mathcal{B}_\theta(\ell) = \{ae_\theta \mid a \in \{-\ell, \dots, \ell\}\} \cup \{ae_\theta + f_\theta \mid a \in \{-\ell, \dots, \ell-1\}\}$$

As $\mathcal{B}_\theta(\ell) \subset \mathcal{L}_A(e_\theta, f_\theta)$ it is compatible with A . Moreover it is easy to specify an enumeration of the vertices of $C(\mathcal{B}_\theta(\ell))$ showing that it is lattice unique.

For a given k we want to choose ℓ as large as possible such that $\mathcal{B}_\theta(\ell)$ “fits inside” $G_0[\mathcal{H}_k]$. We then wish to “attach” $\mathcal{B}_\theta(\ell)$ to $G_0[\mathcal{H}_k]$ with extra points \mathcal{S} , the number of which does neither depend on k nor on θ . We wish to do it in such a way that $\mathcal{A}_1^k(\theta) := \mathcal{B}_\theta(\ell) \cup G_0[\mathcal{H}_k] \cup \mathcal{S}$ is compatible with A . The precise construction is as follows:



■ **Figure 7** The final point set \mathcal{A} where the point sets $\mathcal{C}_k(\theta)$ are “glued” together by translating them such that the contact graph realized by the union of the subsets $\mathcal{H}_k \subset \mathcal{C}_k(\theta)$ is lattice unique.

► **Construction 10** ($\mathcal{C}_k(\theta)$). See Figure 6 (right). Consider the open line segment $L_\theta = \{re_\theta \mid r \in (-r_{\max}, r_{\max})\}$ where r_{\max} is maximal with the property that for all points $x \in L_\theta$ and all $y \in H_k$ it holds that $\|x - y\|_A > 4$. Also let $\ell \in \mathbb{N}$ be maximal such that $\{ae_\theta \mid a \in \{-\ell, \dots, \ell\}\} \subset L_\theta$. Note that

$4 < d_A(\{\ell e_\theta\}, H_k) \leq 6$. Observe moreover that $\ell \geq \frac{\sqrt{3}}{4}k - 3$ as the points $p \in H_k$ have $\|p\|_A \geq \frac{\sqrt{3}}{2}k$. In particular we have the following property which we highlight for later use:

$$\text{If } k > \frac{12}{\sqrt{3} - 1} \text{ it holds that } \ell > \frac{k}{4}. \tag{1}$$

When ℓ is chosen in this fashion, we have that $\mathcal{B}_\theta(\ell)$ is contained in the interior of H_k . Now, $\mathcal{B}_\theta(\ell)$ will constitute our beam in direction θ and we will proceed to show that we can attach it to \mathcal{H}_k , as illustrated, using only a constant number of extra points. That this can be done is conceptually unsurprising but requires a somewhat technical proof.

We define $\mathcal{S}_1(\theta)$ to be extra points going from the boundary of H_k and z_1^θ to be the extra point which connects $\mathcal{B}_\theta(\ell)$ and $\mathcal{S}_1(\theta)$. This attaches one end of the beam, $\mathcal{B}_\theta(\ell)$, to H_k , and we similarly define $\mathcal{S}_2(\theta)$ and z_2^θ to attach the other end. See Figure 6 (right). In the full version we show that $|\mathcal{S}_i(\theta)| \leq 13$ for $i \in \{1, 2\}$. Letting $\mathcal{C}_k(\theta) = \mathcal{H}_k \cup \mathcal{B}_\theta(\ell) \cup \mathcal{S}_1(\theta) \cup \mathcal{S}_2(\theta) \cup \{z_1^\theta, z_2^\theta\}$ be the combination of the components completes the construction.

We are now ready to construct \mathcal{A} which will consist of several translated copies $\mathcal{C}_k(\theta)$.

► **Construction 11** (\mathcal{A}). By Lemma 4 we can find an $\varepsilon \in (0, 1)$ and a finite set of directions $\Theta \subset [0, \pi)$ such that for all linear maps $T : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ there exists $\theta \in \Theta$ such that

$$\left| \frac{\rho_A(\theta)}{\rho_{T(B)}(\theta)} - 1 \right| \geq \varepsilon. \tag{2}$$

That we can scale the deviation to be multiplicative rather than additive is possible because $0 < \inf_{\theta \in [0, \pi)} \rho_A(\theta) \leq \sup_{\theta \in [0, \pi)} \rho_A(\theta) < \infty$.

For each $\theta \in \Theta$ we construct a copy of $\mathcal{C}_k(\theta) = \mathcal{H}_k \cup \mathcal{B}_\theta(\ell) \cup \mathcal{S}_1(\theta) \cup \mathcal{S}_2(\theta) \cup \{z_1^\theta, z_2^\theta\}$ where k is yet to be fixed (ℓ is of course determined by k and θ). We then choose translations $t_\theta \in \mathbb{R}^2$ for each $\theta \in \Theta$ such that the sets $(\mathcal{H}_k + t_\theta)_{\theta \in \Theta}$ are pairwise disjoint, and $\bigcup_{\theta \in \Theta} (\mathcal{H}_k + t_\theta) \subset \mathbb{R}^2$ is compatible with A and induces a lattice unique contact graph. We can choose $(t_\theta)_{\theta \in \Theta}$ in numerous ways to satisfy this. One is depicted in Figure 7. Another is obtained by

3:10 Classifying Convex Bodies by Their Contact and Intersection Graphs

enumerating $\Theta = \{\theta_1, \dots, \theta_q\}$ and defining $t_{\theta_i} = ((2k+3)e_1 - (k+1)e_2) \times (i-1)$. The exact choice is not important and picking one, we define $\mathcal{A}(k) = \bigcup_{\theta \in \Theta} (C_k(\theta) + t_\theta)$ which is a point set compatible with A . Lastly, we set $\mathcal{A} = \mathcal{A}(\lceil \frac{180}{\varepsilon} \rceil)$.

We are now ready for the final step of the proof:

Proving that no graph in $C(B)$ contains a subgraph isomorphic to $G = C_A(\mathcal{A})$.

Suppose for contradiction that there exists a set of points $\mathcal{B} \subset \mathbb{R}^2$ such that G is isomorphic to a subgraph of $C_B(\mathcal{B})$. We may clearly assume that $|\mathcal{A}| = |\mathcal{B}|$ and we let $\varphi : \mathcal{A} \rightarrow \mathcal{B}$ be a bijection which is also a graph homomorphism when considered as a map $C_A(\mathcal{A}) \rightarrow C_B(\mathcal{B})$. The points $\bigcup_{\theta \in \Theta} (\mathcal{H}_k + t_\theta)$ induce a lattice unique contact graph of A . Thus, we may write $\bigcup_{\theta \in \Theta} (\mathcal{H}_k + t_\theta) = \{p_1, \dots, p_n\}$ such that p_1, p_2 and p_3 induce a triangle of G and such that for $i > 3$ there exist distinct $j, k, l < i$ such that p_j, p_k and p_l induce a triangle and such that (p_i, p_k) and (p_i, p_l) are edges of G . By translating the point sets \mathcal{A} and \mathcal{B} we may assume that $\varphi(p_1) = p_1 = 0$. Then applying an appropriate linear transformation T , thus replacing B by $T(B)$, we may assume that $\varphi(p_2) = p_2$ and $\varphi(p_3) = p_3$. Finally, the discussion succeeding Definition 7 implies that in fact $\varphi|_{\bigcup_{\theta \in \Theta} (\mathcal{H}_k + t_\theta)}$ is the identity.

As noted in Construction 11, there exists $\theta \in \Theta$ such that $\left| \frac{\rho_A(\theta)}{\rho_{T(B)}(\theta)} - 1 \right| \geq \varepsilon$. The outline of the remaining argument is as follows: The Euclidean distance between the ‘‘endpoints’’ of the beam $\mathcal{B}_\theta(\ell)$ is $2\ell\rho_A(\theta)$, but the rigidity of $\bigcup_{\theta \in \Theta} (\mathcal{H}_k + t_\theta)$ means that it is also $2\ell\rho_{T(B)}(\theta) + O(1)$. When k (and hence ℓ) is large, this will contradict the inequality above.

We refer the reader to the full version of the paper for the technical details. ◀

► **Remark 12.** We claimed that the proof of the part of Theorem 1 concerning unit distance graphs is identical to the proof above. In fact, if we replace $C(X)$ by $U(X)$ for $X \in \{A, B\}$ in the statement of Theorem 2, the result remains valid. To prove it we would construct \mathcal{A} in precisely the same manner. The important point is then that the comments immediately prior to Theorem 1 concerning the rigidity of the realization of lattice unique graphs remains valid. If in particular $\mathcal{B} \subset \mathbb{R}^2$ satisfies that $U_A(\mathcal{A}) \simeq U_A(\mathcal{B})$ via the isomorphism $\varphi : \mathcal{A} \rightarrow \mathcal{B}$, we may assume that $\varphi|_{\bigcup_{\theta \in \Theta} (\mathcal{H}_k + t_\theta)}$ is the identity as in the proof above. The remaining part of the argument comparing the lengths of the beams then carries through unchanged. In conclusion, we are only left with the task of proving Theorem 1 for intersection graphs.

3 Intersection Graphs

In this section we prove Theorem 3. Consider two convex bodies A and B . We are going to prove that if $I(A) = I(B)$, then for every graph $G \in C(A)$, there exists a graph $H_k(G) \in I(A)$ with properties as stated in the following lemma.

► **Lemma 13.** *Assume that $I(A) = I(B)$. For any $G \in C(A)$ and $k \geq 7$, there exists a graph $H_k(G) \in I(A)$ satisfying the following: Let $X \in \{A, B\}$. For any vertex w of G , there is a corresponding vertex $s_0(w)$ of $H_k(G)$ with the following properties. Consider an arbitrary drawing of $H_k(G)$ as in intersection graph of X and any two vertices w, w' of G and let $s_0 := s_0(w)$ and $s'_0 := s_0(w')$. Then $\|s_0 s'_0\|_X \geq 4k - 18$. Furthermore, if ww' is an edge of G , then $\|s_0 s'_0\|_X \leq 4k + 2$.*

As is evident from the lemma, the vertices $(s_0(u))_{u \in V(G)}$, of any drawing of $H_k(G)$ as an intersection graph of X , are placed approximately as the vertices of a drawing of G as a contact graph of scaled convex body $2kX$. To capture the uncertainty, we introduce the concept of ε -overlap graphs.

► **Definition 14** (ε -overlap Graph). Let $\varepsilon > 0$ and $A \subset \mathbb{R}^2$ be a symmetric convex body, and let $v_1, \dots, v_n \in \mathbb{R}^2$ be n points in the plane. Suppose that for any $i, j \in [n]$, $\|v_i v_j\|_A \geq 2 - \varepsilon$. A graph G with vertex set $[n]$ and edge set satisfying $E(G) \subseteq \{(i, j) \in [n]^2 \mid \|v_i v_j\|_A \leq 2\}$ is called an ε -overlap graph of A . We say that $\{v_1, \dots, v_n\}$ realize the graph G as an ε -overlap graph of A . Further, we denote by $C_\varepsilon(A)$ the set of graphs that can be realized as ε -overlap graphs of A .

We next show how Lemma 13 leads to a proof of Theorem 3. First, the following lemma provides a reduction from ε -overlap graphs to contact graphs. The proof is a standard compactness argument and can be found in the full version of the paper.

► **Lemma 15.** Let $G_1 = (V, E_1)$ be a graph and A a convex body. If for every $\varepsilon > 0$, it holds that $G_1 \in C_\varepsilon(A)$, then there is a graph $G_2 = (V, E_2) \in C(A)$ such that $E_1 \subseteq E_2$.

The following lemma uses Lemma 13 to show that if $I(A) = I(B)$, then any $G \in C(A)$ is an ε -overlap graph of B for all $\varepsilon > 0$.

► **Lemma 16.** Assume that $I(A) = I(B)$. For any $G \in C(A)$, and any $\varepsilon > 0$, $G \in C_\varepsilon(B)$.

Proof. Write $G = (V, E)$ and let $k \geq 7$ be arbitrary. The assumption $I(A) = I(B)$ in particular implies that $H_k(G) \in I(B)$. Consider a drawing of $H_k(G)$ as an intersection graph of B and define $\mathcal{B} := \left\{ \frac{s_0^u}{2k+1} \mid u \in V \right\}$. It follows from Lemma 13 that $I_B(\mathcal{B})$ is a drawing of G as a $\left(2 - \frac{4k-18}{2k+1}\right)$ -overlap graph of B . Since $\frac{4k-18}{2k+1} \geq 2 - 10/k$, it follows that G is an $10/k$ -overlap graph of B . As $k \geq 7$ was arbitrary, the desired result follows. ◀

Theorem 3 is an easy consequence of Lemma 15 and Lemma 16:

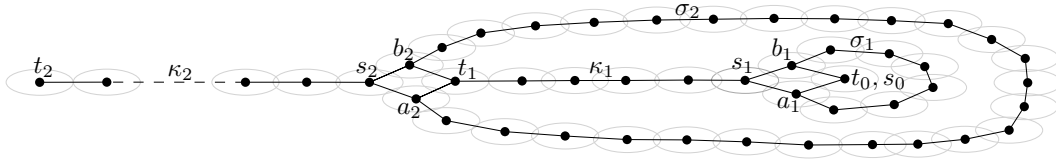
Proof of Theorem 3. Let the graph $G \in C(A)$ have the properties of the theorem, i.e., for all $H \in C(B)$, G is not isomorphic to a subgraph of H . Suppose that $I(A) = I(B)$. By Lemma 15 and 16, there is a graph $H = (V, E) \in C(B)$ such that $E' \subseteq E$, which is a contradiction. ◀

It remains to prove Lemma 13. We will proceed to describe the construction of $H_k(G)$ and provide several lemmas needed in order to prove that it satisfies the desired properties.

The proofs of these lemmas and of Lemma 13 are deferred to the full version of the paper.

For each vertex $u \in V(G)$, we make a copy of a graph Q_k to be defined in the following. The vertices of $H_k(G)$ will in turn be the union of the vertices of these copies. We will construct Q_k to have a designated vertex s_0 and a cycle α_k with the property that for every drawing of Q_k as an intersection graph of $X \in \{A, B\}$, the cycle α_k is contained in (and winds all the way around) the annulus $\{x \in \mathbb{R}^2 \mid \|s_0 x\|_X \in (2k - 3, 2k]\}$. We may then informally view α_k as an upscaled copy of X up to a slight imprecision that, compared to the size, decreases in k . In order to construct Q_k , we first define another graph P_k (which will be contained in Q_k) with a vertex s_0 such that in every drawing of P_k as an intersection graph, s_0 is contained in k nested disjoint cycles. A priori, it is not clear what it means for s_0 to be contained in a cycle of the graph in every drawing, since the drawing is not necessarily a plane embedding of the graph. However, as the following lemma shows, it is well-defined if P_k is triangle-free. We believe the result to be well-known but have been unable to find the exact formulation that we require in the literature.

► **Lemma 17.** If G is a triangle-free graph then every drawing of G as an intersection graph is a plane embedding.



■ **Figure 8** The construction of P_2 .

Proof. See the full version of the paper. ◀

We are now ready to define $P_k \in I(A)$ for any $k > 0$. Besides being triangle-free, our aim is that P_k should have the following properties:

1. There is a vertex s_0 such that in any drawing of P_k as an intersection graph of A and B , s_0 is contained in k nested disjoint, simple cycles $\sigma_1, \dots, \sigma_k$.
2. There is a path κ_k from a vertex s_k to a leaf t_k such that in any drawing of P_k as an intersection graph of A or B , the path κ_k is on the boundary of the outer face.

► **Construction 18** (P_k). See Figure 8. We define $P_k = I_A(\mathcal{A}_k)$, where \mathcal{A}_k is a set of points to be defined inductively. Let $\mathcal{A}_0 = \{0\}$ and $P_0 = I_A(\mathcal{A}_0)$ be the trivial graph consisting of one vertex $s_0 = t_0$, which is also the path κ_0 . Suppose now that $P_{k-1} = I_A(\mathcal{A}_{k-1})$ has been defined. In order to define P_k , we first add vertices a_k, b_k, s_k and edges such that $\tau_k := (a_k, t_{k-1}, b_k, s_k)$ is a 4-cycle. We now add vertices and edges that together with the path a_k, s_k, b_k form a cycle σ_k . We make σ_k so long that there exists a drawing as an intersection graph in which P_{k-1} is contained in σ_k with respect to both A and B . We finish the construction of P_k by adding vertices and edges that together with s_k form a path κ_k from s_k to a vertex t_k , where κ_k is so long that it cannot be contained in the cycle σ_k , neither as an intersection graph of A nor B . (Note that a path of length n contains $n/2$ independent vertices. A simple volume argument implies a bound on the number of independent vertices contained in the region enclosed by a cycle of a plane intersection embedding.) Let \mathcal{A}_k consist of \mathcal{A}_{k-1} together with all the added points.

► **Lemma 19.** *The graph P_k has properties 1–2.*

Proof. See the full version of the paper. ◀

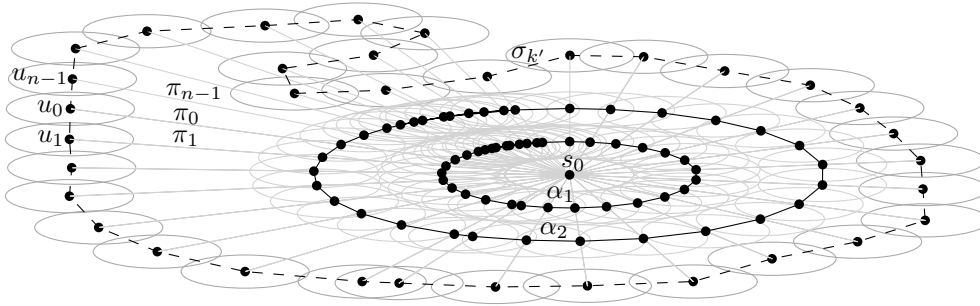
The most important property of P_k is that every vertex $u \in \sigma_k$ has distance $\Omega(k)$ to s_0 in any drawing of P_k as intersection graph of any $X \in \{A, B\}$ in the norm $\|\cdot\|_X$. This is exactly what we will use when constructing Q_k .

► **Lemma 20.** *Let $X \in \{A, B\}$. Consider any drawing of P_k as an intersection graph of X . For any vertex $u \in \sigma_k$, we have $\|s_0 u\|_X > 2(k/9 - 1)$.*

Proof. See the full version of the paper. ◀

Having defined P_k we are now ready for the construction of Q_k .

► **Construction 21** (Q_k). We here define a graph $Q_k \in I(A)$ by specifying a drawing of Q_k as an intersection graph of A . Let $k' := 18(k + 1)$. We start with $P_{k'}$ and explain what to add to obtain Q_k . Let u_0, \dots, u_{n-1} be the vertices of $\sigma_{k'}$ in cyclic, counter-clockwise order. Consider an arbitrary drawing of $P_{k'}$ as an intersection graph of A and a vertex u_i . Note that $d := \left\lceil \frac{\|s_0 u_i\|_A - 2}{2} \right\rceil$ is the number of vertices needed to add in order to create a path from s_0 to u_i . It follows from Lemma 20 that $d \geq 2k$.



■ **Figure 9** A part of a graph Q_k . The vertices $v_i(j)$ are only shown for $j \in \{1, 2\}$, and only edges on paths π_i and cycles $\alpha_1, \alpha_2, \sigma_{k'}$ are shown.

We want to minimize the vector of these values d with respect to each vertex $u_i \in \sigma_{k'}$. To be precise, we define

$$(d_0, \dots, d_{n-1}) := \min \left(\left\lceil \frac{\|s_0 u_0\|_A - 2}{2} \right\rceil, \dots, \left\lceil \frac{\|s_0 u_{n-1}\|_A - 2}{2} \right\rceil \right),$$

where the minimum is with respect to the lexicographical order and taken over all drawings of $P_{k'}$ as an intersection graph. Consider a drawing of $P_{k'}$ as an intersection graph realizing the minimum and let \mathcal{P} be the set of vertices in the drawing. For each vertex u_i , we create a path π_i from s_0 to u_i as follows. Let \mathbf{v}_i be the unit-vector in direction $u_i - s_0$. We add new vertices placed at the points $v_i(j) := s_0 + 2j\mathbf{v}_i$ for $j \in \{1, \dots, d_i\}$. We now define the vertices of Q_k as $\mathcal{Q} := \mathcal{P} \cup \bigcup_{i=0}^{n-1} \{v_i(1), \dots, v_i(d_i)\}$ and define $Q_k = I_A(\mathcal{Q})$. See Figure 9.

► **Remark 22.** By construction, there exists a drawing of Q_k as an intersection graph of A . If there does not exist one of B , we are done, since we then clearly have that $I(A) \neq I(B)$. Now suppose that there exists a drawing of $P_{k'}$ as an intersection graph of B such that

$$\left(\left\lceil \frac{\|s_0 u_0\|_B - 2}{2} \right\rceil, \dots, \left\lceil \frac{\|s_0 u_{n-1}\|_B - 2}{2} \right\rceil \right) \prec (d_0, \dots, d_{n-1}), \tag{3}$$

where \prec denotes the lexicographical order. We can now define a graph $Q_k^B \in I(B)$ from $P_{k'}$ in a similar way as we defined Q_k by adding $\left\lceil \frac{\|s_0 u_i\|_B - 2}{2} \right\rceil$ vertices to form a path from s_0 to each u_i . It then follows from (3) that $Q_k^B \notin I(A)$, so in this case we have likewise succeeded in proving $I(A) \neq I(B)$. In the following, we therefore assume that $Q_k \in I(B)$ for any k and that no drawing of $P_{k'}$ as an intersection graph of B satisfying (3) exists.

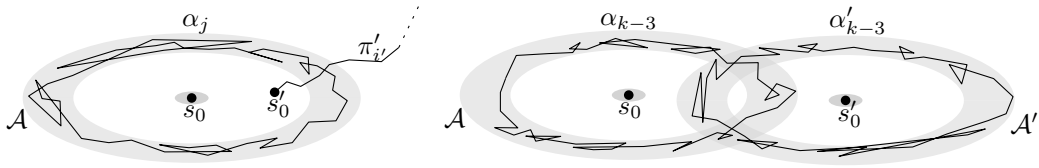
First we need to show that Q_k contains a cycle α_k as described earlier.

► **Lemma 23.** *The set of edges of Q_k contains the pairs $v_i(j)v_{i+1}(j)$ for any $i \in [n]$ and $j \in \{1, \dots, k\}$, and for each $j \in \{1, \dots, k\}$, these edges thus form a cycle α_j . In the specific drawing of Q_k as an intersection graph defined in Construction 21, the cycle α_j is contained in the annulus $\{x \in \mathbb{R}^2 \mid \|s_0 x\|_A \in [2j - 1, 2j]\}$.*

Proof. See the full version of the paper. ◀

The above lemma shows that the cycle α_k behaves nicely in one particular drawing of Q_k as an intersection graph. To see that something similar holds for every drawing, we refer the reader to the full version.

We now provide the definition of the graph $H_k(G)$, as mentioned in the beginning of this section.



■ **Figure 10** From the proof of Lemma 13 (notation as in Lemma 25). There is either an intersection between α_j and $\pi'_{i'}$ (left) or between α_{k-3} and α'_{k-3} (right) that violates Lemma 25.

► **Construction 24** ($H_k(G)$). For any $G \in C(A)$, consider a fixed drawing of G as a contact graph of A . For each vertex w of G , we make a copy of the drawing of Q_k as an intersection graph as defined in Construction 21 which we translate so that s_0 is placed at $s_0^w := (2k-2)w$. We then add all edges induced by the vertices, and the result is denoted as $H_k(G)$.

The following lemma characterizes some of the edges of $H_k(G)$ and will be crucial in the proof of Lemma 13.

► **Lemma 25.** Consider two vertices w, w' of a drawing of a graph G as a contact graph. Denote by Q and Q' the copies of Q_k in $H_k(G)$ corresponding to w and w' , respectively, such that $s_0, \pi_i, \alpha_j, v_i(j)$ denote objects in Q and $s'_0, \pi'_{i'}, \alpha'_j, v'_i(j)$ denote objects in Q' . If $v_i(j)v'_{i'}(j')$ is an edge of $H_k(G)$, then $j + j' \geq 2k - 4$.

If ww' is an edge of G , then there is an edge $v_i(k)v'_{i'}(k)$ in $H_k(G)$.

Proof. See the full version of the paper. ◀

As mentioned, the final proof of Lemma 13 is deferred to the full version, but we can now provide the main ideas. We first need to prove that in any drawing of Q_k as an intersection graph with respect to $X \in \{A, B\}$, any cycle α_j is contained in an annulus only slightly wider than as stated in Lemma 23. Furthermore, α_j winds around s_0 in the sense that if we trace the full curve α_j , the change of argument with respect to s_0 will be $\pm 2\pi$. To prove the lower bound $\|s_0 s'_0\|_X \geq 4k - 18$ in Lemma 13, we exclude that the distance is smaller by dividing into two cases depending on the actual distance. Figure 10 depicts the two cases for each of which we prove that there would be an edge in $H_k(G)$ violating Lemma 25. The upper bound $\|s_0 s'_0\|_X \leq 4k + 2$ when ww' is an edge of G is likewise an easy consequence of Lemma 25, as otherwise, an edge $v_i(k)v'_{i'}(k)$ would be missing from $H_k(G)$.

4 Concluding remarks

It is natural to investigate the special case of convex bodies with the URTC property. Here our proof of Theorem 2 fails since the hexagons are not rigid structures. Together with Konrad Swanepoel, we have promising progress in generalizing Theorem 1 to also handle this case.

Another interesting direction is to consider convex bodies in three and higher dimensions. Already in three dimensions, it appears to be very difficult to characterize when two bodies induce the same graph classes.

References

- 1 Anders Aamand, Mikkel Abrahamsen, Jakob Bæk Tejs Knudsen, and Peter Michael Reichstein Rasmussen. Classifying convex bodies by their contact and intersection graphs, 2019. Preprint. [arXiv:1902.01732](https://arxiv.org/abs/1902.01732).

- 2 Édouard Bonnet, Nicolas Grelier, and Tillmann Miltzow. Maximum clique in disk-like intersection graphs. *Preprint*, 2020. [arXiv:2003.02583](https://arxiv.org/abs/2003.02583).
- 3 Károly Böröczky Jr. *Finite packing and covering*, volume 154 of *Cambridge Tracts in Mathematics*. Cambridge University Press, 2004.
- 4 Sergio Cabello and Miha Ježič. Refining the hierarchies of classes of geometric intersection graphs. *The Electronic Journal of Combinatorics*, 24(1):1–19, 2017.
- 5 Jean Cardinal. Computational geometry column 62. *SIGACT News*, 46(4):69–78, 2015.
- 6 Jean Cardinal, Stefan Felsner, Tillmann Miltzow, Casey Tompkins, and Birgit Vogtenhuber. Intersection graphs of rays and grounded segments. In Hans L. Bodlaender and Gerhard J. Woeginger, editors, *Graph-Theoretic Concepts in Computer Science*, pages 153–166, Cham, 2017. Springer International Publishing.
- 7 Timothy M. Chan and Dimitrios Skrepetos. Approximate shortest paths and distance oracles in weighted unit-disk graphs. *Journal of Computational Geometry*, 10(2), 2019.
- 8 Steven Chaplick, Stefan Felsner, Udo Hoffmann, and Veit Wiechert. Grid intersection graphs and order dimension. *Order*, 35:363–391, 2018.
- 9 Brent N. Clark, Charles J. Colbourn, and David S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1-3):165–177, 1990.
- 10 László Fejes Tóth. *Lagerungen in der Ebene auf der Kugel und im Raum*, volume 65 of *Die Grundlehren der mathematischen Wissenschaften*. Springer, second edition, 1972.
- 11 Stefan Felsner and Günter Rote. On primal-dual circle representations. In *34th European Workshop on Computational Geometry (EuroCG 2018)*, pages 72:1–72:6, 2018.
- 12 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Finding, hitting and packing cycles in subexponential time on unit disk graphs. *Discrete & Computational Geometry*, 62(4):879–911, 2019.
- 13 Fedor V. Fomin, Daniel Lokshtanov, Fahad Panolan, Saket Saurabh, and Meirav Zehavi. ETH-Tight Algorithms for Long Path and Cycle on Unit Disk Graphs. In *36th International Symposium on Computational Geometry (SoCG 2020)*, pages 44:1–44:18, 2020. [doi:10.4230/LIPIcs.SoCG.2020.44](https://doi.org/10.4230/LIPIcs.SoCG.2020.44).
- 14 Stefan Funke, Alexander Kesselman, Ulrich Meyer, and Michael Segal. A simple improved distributed algorithm for minimum CDS in unit disk graphs. *ACM Transactions on Sensor Networks*, 2(3):444–453, 2006. [doi:10.1145/1167935.1167941](https://doi.org/10.1145/1167935.1167941).
- 15 György Pál Gehér. A contribution to the Aleksandrov conservative distance problem in two dimensions. *Linear Algebra and its Applications*, 481:280–287, 2015.
- 16 Albert Gräf, Martin Stumpf, and Gerhard Weißenfels. On coloring unit disk graphs. *Algorithmica*, 20(3):277–293, 1998.
- 17 Svante Janson and Jan Kratochvíl. Thresholds for classes of intersection graphs. *Discrete Mathematics*, 108:307–326, 1992.
- 18 Victor Klee. Some new results on smoothness and rotundity in normed linear spaces. *Mathematische Annalen*, 139(1):51–63, 1959.
- 19 P. Koebe. Kontaktprobleme der konformen Abbildung. *Berichte über die Verhandlungen der Sächsischen Akademie der Wissenschaften zu Leipzig, Mathematisch-Physische Klasse*, 88:141–164, 1936.
- 20 Jan Kratochvíl and Jiří Matoušek. Intersection graphs of segments. *Journal of Combinatorial Theory Series B*, 62(2):289–315, 1994.
- 21 M. V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25(2):59–68, 1995. [doi:10.1002/net.3230250205](https://doi.org/10.1002/net.3230250205).
- 22 Colin McDiarmid and Tobias Müller. Integer realizations of disk and segment graphs. *Journal of Combinatorial Theory, Series B*, 103(1):114–143, 2013.
- 23 Tobias Müller, Erik Jan van Leeuwen, and Jan van Leeuwen. Integer representations of convex polygon intersection graphs. *SIAM Journal on Discrete Mathematics*, 27(1):205–231, 2013.

3:16 Classifying Convex Bodies by Their Contact and Intersection Graphs

- 24 Fahad Panolan, Saket Saurabh, and Meirav Zehavi. Contraction decomposition in unit disk graphs and algorithmic applications in parameterized complexity. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2019)*, pages 1035–1054, 2019.
- 25 Marco A. Peyrot-Solís, Giselle M. Galvan-Tejada, and Hildeberto Jardon-Aguilar. Proposal of a planar directional UWB antenna for any desired operational bandwidth. *International Journal of Antennas and Propagation*, 2014:1–12, 2014.
- 26 Oded Schramm. Combinatorially prescribed packings and applications to conformal and quasiconformal maps. *Preprint*, 2007. [arXiv:0709.0710](https://arxiv.org/abs/0709.0710).
- 27 Konrad Swanepoel. Combinatorial distance geometry in normed spaces. In Gergely Ambrus, Imre Bárány, Károly J. Böröczky, Gábor Fejes Tóth, and János Pach, editors, *New Trends in Intuitive Geometry*, volume 27 of *Bolyai Society Mathematical Studies*. Springer, 2018.
- 28 G. Fejes Tóth. *New Results in the Theory of Packing and Covering*, pages 318–359. Birkhäuser Basel, Basel, 1983. doi:10.1007/978-3-0348-5858-8_14.
- 29 Weili Wu, Hongwei Du, Xiaohua Jia, Yingshu Li, and Scott C-H Huang. Minimum connected dominating sets and maximal independent sets in unit disk graphs. *Theoretical Computer Science*, 352(1-3):1–7, 2006.
- 30 Tudor Zamfirescu. Nearly all convex bodies are smooth and strictly convex. *Monatshefte für Mathematik*, 103(1):57–62, 1987.

Appendix H

Tiling with Squares and Packing Dominoes in Polynomial Time

Tiling with Squares and Packing Dominos in Polynomial Time

Anders Aamand* Mikkel Abrahamsen* Thomas D. Ahle*
Peter M. R. Rasmussen*

August 9, 2021

Abstract

A polyomino is a polygonal region with axis parallel edges and corners of integral coordinates, which may have holes. In this paper, we consider planar tiling and packing problems with polyomino pieces and a polyomino container P . We give two polynomial time algorithms, one for deciding if P can be tiled with $k \times k$ squares for any fixed k which can be part of the input (that is, deciding if P is the union of a set of non-overlapping $k \times k$ squares) and one for packing P with a maximum number of non-overlapping and axis-parallel 2×1 dominos, allowing rotations by 90° . As packing is more general than tiling, the latter algorithm can also be used to decide if P can be tiled by 2×1 dominos.

These are classical problems with important applications in VLSI design, and the related problem of finding a maximum packing of 2×2 squares is known to be NP-Hard [J. Algorithms 1990]. For our three problems there are known pseudo-polynomial time algorithms, that is, algorithms with running times polynomial in the *area* or *perimeter* of P . However, the standard, compact way to represent a polygon is by listing the coordinates of the corners in binary. We use this representation, and thus present the first polynomial time algorithms for the problems. Concretely, we give a simple $O(n \log n)$ algorithm for tiling with squares, and a more involved $O(n^3 \text{ polylog } n)$ algorithm for packing and tiling with dominos, where n is the number of corners of P .

*Basic Algorithms Research Copenhagen (BARC), University of Copenhagen. BARC is supported by the VILLUM Foundation grant 16582.

1 Introduction

A chessboard has been mutilated by removing two diagonally opposite corners, leaving 62 squares. Philosopher Max Black asked in 1946 whether one can place 31 dominoes of size 1×2 so as to cover all of the remaining squares? Tiling problems of this sort are popular in recreational mathematics, such as the mathematical olympiads¹ and have been discussed by Golomb [14] and Gamow & Stern [12]. The mutilated chessboard and the dominos are examples of the type of polygon called a *polyomino*, which is a polygonal region of the plane with axis parallel edges and corners of integral coordinates. We allow polyominoes to have holes.

From an algorithmic point of view, it is natural to ask whether a given (large) polyomino P can be *tilled* by copies of another fixed (small) polyomino Q , which means that P is the union of non-overlapping copies of Q that may or may not be rotated by 90° and 180° . As the answer is often a boring *no*, one can ask more generally for the largest number of copies of Q that can be *packed* into the given container P without overlapping. Algorithms answering this question (for various Q) turn out to have important applications in very large scale integration (VLSI) circuit technology. As a concrete example, Hochbaum & Maass [16] gave the following motivation for their development of a polynomial time approximation scheme for packing 2×2 squares into a given polyomino P (using the area representation of P , to be defined later):

“For example, 64K RAM chips, some of which may be defective, are available on a rectilinear grid placed on a silicon wafer. 2×2 arrays of such nondefective chips could be wired together to produce 256K RAM chips. In order to maximize yield, we want to pack a maximal number of such 2×2 arrays into the array of working chips on a wafer.”

Although the mentioned amounts of memory are small compared to those of present day technology, the basic principles behind the production of computer memory are largely unchanged, and methods for circumventing defective cells of wafers (the cells are also known as *dies* in this context) is still an active area of research in semiconductor manufacturing [7, 9, 18, 21].

The most important result in tiling is perhaps the combinatorial group theory approach by Conway & Lagarias [8]. Their algorithmic technique is used to decide whether a given finite region consisting of cells in a regular lattice (triangular, square, or hexagonal) can be tiled by pieces drawn from a finite set of tile shapes. Thurston [26] gives a nice introduction to the technique and shows how it can be used to decide if a polyomino without holes can be tiled by dominos. The running time is $O(a \log a)$, where a is the *area* of P . Pak, Sheffer, & Tassy [22] described an algorithm with running time $O(p \log p)$, where p is the *perimeter* of P .

The problem of *packing* a maximum number of dominos into a given polyomino P was apparently first analyzed by Berman, Leighton, & Snyder [5] who observed that this problem can be reduced to finding a maximum matching of the incidence graph $G(P)$ of the cells in P : There is a vertex for each 1×1 cell in P , and two vertices are connected if the two cells share a geometrical edge. The

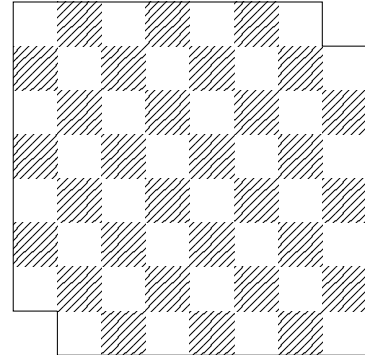


Figure 1: The chessboard polyomino envisioned by Max Black.

¹See e.g. the “hook problem” of the International Mathematical Olympiad 2004.

graph $G(P)$ is bipartite, so the problem can be solved in $O(n^{3/2})$ time using the Hopcroft–Karp algorithm, where n is the number of cells (i.e., the area of P).

On the flip-side, a number of hardness results have been obtained for simple tiling and packing problems: Beauquier, Nivat, Remila, & Robson [2] showed that if P can have holes, the problem of deciding if P can be tiled by translates of two rectangles $1 \times m$ and $k \times 1$ is NP-complete as soon as $\max\{m, k\} \geq 3$ and $\min\{m, k\} \geq 2$. Pak & Yang [23] showed that there exists a set of at most 10^6 rectangles such that deciding whether a given *hole-free* polyomino can be tiled with translates from the set is NP-complete. Other generalizations have even turned out to be undecidable: Berger [3] proved in 1966 that deciding whether pieces from a given finite set of polyominoes can tile the plane is Turing complete. For packing, Fowler, Paterson, & Tanimoto [11] showed already in the early 80s that deciding whether a given number of 3×3 squares can be packed into a polyomino (with holes) is NP-complete, and the result was strengthened to 2×2 squares by Berman, Johnson, Leighton, Shor, & Snyder [4].

As it turns out, for all of the above results, it is assumed that the container P is represented either as a list of the individual cells forming the interior of P or as a list of the boundary cells. We shall call these representations the *area representation* and *perimeter representation*, respectively. The area and perimeter representations correspond to a unary rather than binary representation of integers and the running times of the existing algorithms are thus only pseudo-polynomial. It is much more efficient and compact to represent P by the coordinates of the corners, where the coordinates are represented as binary numbers. This is the way one would usually represent polygons (with holes) in computational geometry: The corners are given in cyclic order as they appear on the boundary of P , one cycle for the outer boundary and one for each of the holes of P . We shall call such a representation a *corner representation*. With a corner representation, the area and perimeter can be exponential in the input size, so the known algorithms which rely on an area or perimeter representation to be polynomial, are in fact exponential when using this more efficient encoding of the input. Problems that are NP-complete in the area or perimeter representation are also NP-hard in the corner representation, but NP-membership does not necessarily follow. In our practical example of semiconductor manufacturing, the corner representation also seems to be the natural setting for the problem. Hopefully, there are only few defective cells to be avoided when grouping the chips, so the total number of corners of the usable region is much smaller than its area.

El-Khechen, Dulieu, Iacono, & Van Omme [10] showed that even using a corner representation for a polyomino P , the problem of deciding if m squares of size 2×2 can be packed into P is in NP. That was not clear before since the naive certificate specifies the placement of each of the m squares, and so, would have exponential length. Beyond this, we know of no other work using the corner representation for polyomino tiling or packing problems.

Our contribution. While the complexity of the problem of packing 2×2 squares into a polyomino P has thus been settled as NP-complete, the complexity of the tiling problem was left unsettled. Tiling and packing are closely connected in this area of geometry, but their complexities can be drastically different. Indeed, we show in Section 3 that it can be decided in $O(n \log n)$ time by a surprisingly simple algorithm whether P can be tiled by $k \times k$ squares for any fixed $k \in \mathbb{N}$ which can even be part of the input. Here, n is the number of corners of P .² With the area and perimeter

²We assume throughout the paper that we can make basic operations (additions, subtractions, comparisons) on the coordinates in $O(1)$ time. Otherwise, the time complexities of our two algorithms will be $O(nt \log n)$ and $O(n^3 t + n^3 \text{polylog } n)$, respectively, where t is the time it takes to make one such operation.

representations, it is trivial to decide if P can be tiled in polynomial time (see Section 3), but as noted above, using the corner representation, it is not even immediately obvious that the problem is in NP.

In Section 4.1-4.5, we provide and analyse an algorithm that can decide in $O(n^3 \text{ polylog } n)$ time if m dominos (i.e., rectangles of size 1×2 that can be rotated 90°) can be packed in a given polyomino P . This algorithm is more complicated and we consider it our most important contribution. The algorithm implicitly constructs a maximum packing, and the same algorithm can be used to decide if P can be tiled by dominos. In Section 4.6, we further describe a much simpler algorithm that works by truncating long edges of P (using a multiple-sink multiple-source maximum flow algorithm as a black box). The simplicity of this algorithm comes at the cost of a higher running time of $O(n^4 \text{ polylog } n)$. The proof that the simple algorithm works uses the same structural results as we developed for the faster but more complicated algorithm. Table 1 summarises the known and new results.



Shapes	Tiling	Packing
	$O(n^3 \text{ polylog } n)$ [This paper]	$O(n^3 \text{ polylog } n)$ [This paper]
	$O(n \log n)$ [This paper]	NP-complete [4, 10]

Table 1: Complexities of the four fundamental tiling and packing problems. Here, n is the number of corners of the container P . The algorithm for tiling with squares works for any size $k \times k$.

Further related work. The technique by Conway & Lagarias [8] has been adapted to obtain algorithms for tiling with other shapes than dominos: Kenyon & Kenyon [19] showed how to decide whether a given hole-free polyomino P can be tiled with translates of the rectangles $1 \times m$ and $k \times 1$ for fixed integers m and k . The running time is again linear in the area of P . They also described an algorithm to decide if a polyomino can be tiled by the rectangles $k \times m$ and $m \times k$ with running time quadratic in the area. Rémila [25] generalized the work by Kenyon and Kenyon and obtained a quadratic time algorithm for deciding whether a given hole-free polyomino can be tiled by translates of two fixed rectangles $k \times m$ and $k' \times m'$. Wijshoff & van Leeuwen [27] and Beauquier & Nivat [1] gave algorithms for deciding whether a given polyomino tiles the entire plane. For work on packing trominos, that is, polyominos consisting of three unit squares, see [17].

1.1 Our techniques

Tiling with $k \times k$ squares. We sort the corners of the given polyomino P by the x -coordinates and use a vertical sweep-line ℓ that sweeps over P from left to right. The intuition is that the algorithm keeps track of how the tiling looks in the region of P to the left of ℓ if a tiling exists. As ℓ sweeps over P , we keep track of how the tiling pattern changes under ℓ . Each vertical edge of P that ℓ sweeps over causes changes to the tiling, and we must update our data structure accordingly.

Packing with dominos. Our basic approach is to reduce the packing problem in the polyomino P (with n corners) to a maximum matching problem in a graph G^* with only $O(n^3)$ vertices and edges.

We prove that a maximum matching in G^* corresponds to a maximum packing of dominos in P . The construction of G^* requires many techniques and the correctness relies on several structural results on domino packings and technical lemmas regarding the particular way we define the intermediate polyominoes and graphs that are used to eventually arrive at G^* .

We first find the maximum subpolyomino $P_1 \subset P$ such that all corners of P_1 have even coordinates. We then use a hole-elimination technique: By carving channels in P_1 from the holes to the boundary, we obtain a hole-free subpolyomino $P_2 \subset P_1$. The particular way we choose the channels is important in order to ensure that the final graph G^* has size only $O(n^3)$. We now apply a technique of reducing P by removing everything far from the boundary of P_2 : We consider the subpolyomino $Q \subset P_2$ of all cells with at least some distance $\Omega(n)$ to the boundary of P_2 , and then we define $P_3 := P \setminus Q$ (note that Q is removed from P and not from P_2). The main insight is that any packing of dominos in P_3 can be extended to a packing of all of P that, restricted to Q , is a *tiling*. For this to hold, it turns out to be important that P_2 has no holes.

A crucial step is to prove that every cell in the polyomino P_3 has distance $O(n)$ to the boundary of P_3 and that P_3 has $O(n)$ corners. There may, however, still be an exponential number of cells in P_3 due to long *pipes* (corridors). We then develop a technique for contracting these long pipes. The contraction is not carried out geometrically, but in the incidence graph $G_3 := G(P_3)$ of the cells of P_3 , by contracting long horizontal and vertical paths to single edges, and the resulting graph is G^* .

All vertices of G^* correspond to cells of P_3 with distance at most $O(n)$ from a corner of P_3 , and since P_3 has $O(n)$ corners, we get that G^* has size $O(n^3)$. We then compute a maximum matching in G^* using a multiple-source multiple-sink maximum flow algorithm by Borradaile, Klein, Mozes, Nussbaum, & Wulff-Nilsen [6], which has since been improved slightly by Gawrychowski & Karczmarz [13]. This results in a running time of $O(n^3 \text{polylog } n)$. The number of dominos in a maximum packing in the original polyomino P is then the size of the maximum matching plus half of the area of everything that has been removed from P .

2 Preliminaries

We define a *cell* to be a 1×1 square of the form $[i, i + 1] \times [j, j + 1]$, $i, j \in \mathbb{Z}$. A subset $P \subseteq \mathbb{R}^2$ is called a *polyomino* if it is a finite union of cells. For a polyomino P , we define $G(P)$ to be the graph which has the cells in P as vertices and an edge between two cells if they share a (geometrical) edge. We say that P is *connected* if $G(P)$ is a connected graph. Figure 2 (a) illustrates a connected polyomino. For a simple closed curve $\gamma \subset \mathbb{R}^2$, we denote by $\text{Int } \gamma$ the interior of γ . An alternative way to represent a connected polyomino is by a sequence of simple closed curves $(\gamma_0, \gamma_1, \dots, \gamma_h)$ such that (1) each of the curves follows the horizontal and vertical lines of the integral grid \mathbb{Z}^2 , (2) for each $i \in \{1, \dots, h\}$, $\text{Int } \gamma_i \subseteq \text{Int } \gamma_0$, (3) for each distinct $i, j \in \{1, \dots, h\}$, $\text{Int } \gamma_i \cap \text{Int } \gamma_j = \emptyset$, and (4) for distinct $i, j \in \{0, \dots, h\}$, $\gamma_i \cap \gamma_j \subseteq \mathbb{Z}^2$. For a connected polyomino P , there exists a unique such sequence (up to permutations of $\gamma_1, \dots, \gamma_h$) with $P = \overline{\text{Int } \gamma_0} \setminus (\bigcup_{i=1}^h \text{Int } \gamma_i)$. It is standard to reduce our tiling and packing problems to corresponding tiling and packing problems for connected polyominoes, so for simplicity we will assume that the input polyominoes to our algorithms are connected. The *corners* of a polyomino P (specified by a sequence $(\gamma_0, \gamma_1, \dots, \gamma_h)$), are the corners of the curves $\gamma_0, \dots, \gamma_h$. We assume that an input polyomino with n corners is represented using $O(n)$ words of memory by describing the corners of each of the curves $\gamma_0, \dots, \gamma_h$ in cyclic order.

In this paper we will exclusively work with the L_∞ -norm when measuring distances. For two

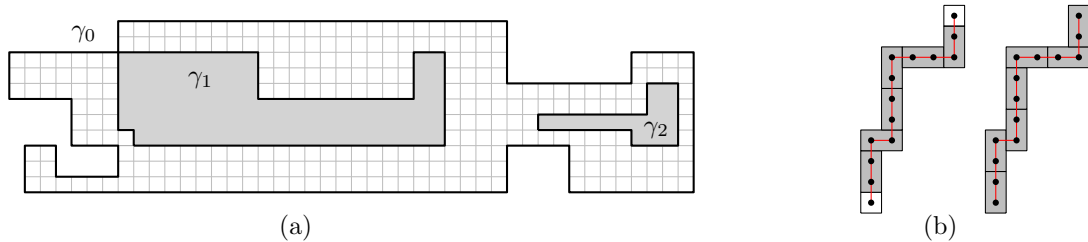


Figure 2: (a) A polyomino with two holes. (b) Extending a domino packing using an augmenting path in $G(P)$.

points $a, b \in \mathbb{R}^2$ we define $\text{dist}(a, b) = \|a - b\|_\infty$. For two subsets $A, B \subseteq \mathbb{R}^2$ we define

$$\text{dist}(A, B) = \inf_{(a,b) \in A \times B} \text{dist}(a, b).$$

In our analysis, A and B will always be closed and bounded (they will in fact be polyomios), and then the inf can be replaced by a min. Finally, we need the notion of the *offset* $B(A, r)$ of a set $A \subseteq \mathbb{R}^2$ by a value $r \in \mathbb{R}$. If $r \geq 0$, we define

$$B(A, r) := \{x \in \mathbb{R}^2 \mid \text{dist}(x, A) \leq r\},$$

and otherwise, we define $B(A, r) := B(A^c, -r)^c$. Note that if $r \geq 0$, we have $A \subset B(A, r)$ and otherwise, we have $B(A, r) \subset A$.

Note that a domino packing of P naturally corresponds to a matching of $G(P)$ and we will often take this viewpoint. We therefore require some basic matching terminology and a result on how to extend matchings. Let G be a graph and M a matching of G . A path (v_1, \dots, v_{2k}) of G is said to be an *augmenting path* if v_1 and v_{2k} are unmatched in M and for each $1 \leq i \leq k - 1$, v_{2i} and v_{2i+1} are matched to each other in M . Modifying M restricted to $\{v_1, \dots, v_{2k}\}$ by instead matching (v_{2i-1}, v_{2i}) for $1 \leq i \leq k$, we obtain a larger matching which now includes the two vertices v_1 and v_{2k} . See Figure 2 (b) for an illustration in the context of domino packings. We require the following basic result by Berge which guarantees that any non-maximum matching of G can always be extending to a larger matching using an augmenting path as above.

Lemma 1 (Berge). *Let G be a graph and M a matching of G which is not maximum. Then there exists an augmenting path between two unmatched vertices G .*

3 Tiling with squares

Naive algorithm. The naive algorithm to decide if P can be tiled with $k \times k$ tiles works as follows. Consider any convex corner c of P . A $k \times k$ square S must be placed with a corner at c . If S is not contained in P , we conclude that P cannot be tiled with $k \times k$ squares. Otherwise, we recurse of the uncovered part $P \setminus S$. When nothing is left, we conclude that P can be tiled. This algorithm runs in time polynomial in the area of P and also shows that if P can be tiled, there is a unique way to do it.

Sweep line algorithm. For the ease of presentation, we focus on the case of deciding tileability using 2×2 squares. It is straightforward to adapt the algorithm to decide tileability by $k \times k$ squares for any fixed $k \in \mathbb{N}$, as explained in the end of this section.

Our algorithm for deciding if a given polyomino P can be tiled with 2×2 squares uses a vertical sweep line that sweeps over P from left to right. The intuition is that the algorithm keeps track of how the tiling looks in the region of P to the left of ℓ if a tiling exists. As ℓ sweeps over P , we keep track of how the tiling pattern changes under ℓ . Each vertical edge of P that ℓ sweeps over causes changes to the tiling, and we must update our data structures accordingly.

Recall that if P is tileable, then the tiling is unique. We define $T(P) \subset P$ to be the union of the boundaries of the tiles in the tiling of P , i.e., such that $P \setminus T(P)$ is a set of open 2×2 squares. If P is not tileable, we define $T(P) := \perp$.

Consider the situation where the sweep line is some vertical line ℓ with integral x -coordinate $x(\ell)$. The algorithm stores a set \mathcal{I} of pairwise interior-disjoint closed intervals $\mathcal{I} = I_1, \dots, I_m \subset \mathbb{R}$, ordered from below and up. Each interval I_i has endpoints at integers and represents the segment $I'_i := \{x(\ell)\} \times I_i$ on ℓ . In the simple case that no vertical edge of P has x -coordinate $x(\ell)$ (so that no change to the set $P \cap \ell$ happens at this point), the intervals \mathcal{I} together represent the part of ℓ in P , i.e., we have $P \cap \ell = \bigcup_{i \in [m]} I'_i$. If one or more vertical edges of P have x -coordinate $x(\ell)$, then $P \cap \ell$ changes at this point and the intervals \mathcal{I} must be updated accordingly.

For each interval I_i we store a *parity* $p(I_i) \in \{0, 1\}$, which encodes how the tiling must be at I'_i if P is tileable. To make this precise, we state the following *parity invariant* of the algorithm under the assumption that P is tileable; see also Figure 3.

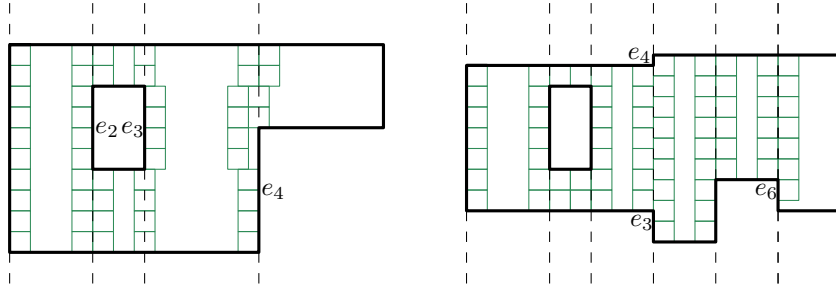


Figure 3: Two instances that cannot be tiled. Left: The edge e_2 splits the only interval in \mathcal{I} into two smaller intervals. Then e_3 introduces a new interval with a different parity than the existing two. The edge e_4 makes the algorithm conclude that P cannot be tiled since e_4 overlaps an interval with the wrong parity. Right: The edges e_3 and e_4 introduce new intervals that are merged with the existing one. Edge e_6 introduces an interval which is merged with the existing interval and the result has odd length, so the algorithm concludes that P cannot be tiled.

- If $p(I_i)$ and $x(\ell)$ have the same parity, then $I'_i \subset T(P)$, i.e., I'_i follows the boundaries of some tiles and does not pass through the middle of any tile.
- Otherwise, $I'_i \cap T(P)$ consists of isolated points, i.e., I'_i passes through the middle of some of the tiles and does not follow the boundary of any tile.

We say that two neighboring intervals I_i, I_{i+1} of \mathcal{I} are *true neighbors* if I_i and I_{i+1} share an endpoint. In addition to the parity invariant, we require \mathcal{I} to satisfy the following *neighbor invariant*: Any pair of true neighbors of \mathcal{I} have different parity.

The pseudocode of the algorithm is shown in Algorithm 1. Initially, we sort all vertical edges after their x -coordinates and break ties arbitrarily. We then run through the edges in this order. Each edge makes a change to the set $P \cap \ell$, and we need to update the intervals \mathcal{I} accordingly so that the parity and the neighbor invariants are satisfied after each edge has been handled. Figure 3 shows the various events.

Consider the event that the sweep line ℓ reaches a vertical edge $e_j = \{x\} \times [y_0, y_1]$. If the interior of P is to the left of e_j , then $P \cap \ell$ shrinks. Each interval $I_i \in \mathcal{I}$ that overlaps $[y_0, y_1]$ must then also shrink, be split into two, or disappear from \mathcal{I} . This is handled by the for-loop at line 5. If the parity of one of these intervals I_i does not agree with the parity of e_j , we get from the parity invariant that P cannot be tiled, and hence the algorithm returns “no tiling” at line 7.

If on the other hand the interior of P is to the right of e_j , then $P \cap \ell$ expands and a new interval I must be added to \mathcal{I} . This is handled by the else-part at line 9. The new interval I may have one or two true neighbors in \mathcal{I} . If one or two such neighbors also have the same parity as I , we merge these intervals into one interval of \mathcal{I} . This ensures that the neighbor invariant is satisfied after e_j has been handled.

In line 13, we consider the case that we finished handling all vertical edges at some specific x -coordinate so that the sweep line will move to the right in order to handle the next edge e_{j+1} in the next iteration. If there is an interval I_i of odd length in \mathcal{I} , it follows from the parity invariant together with the neighbor invariant that P cannot be tiled, so the algorithm returns “no tiling” at line 14.

Algorithm 1:

```

1 Let  $e_1, \dots, e_k$  be the vertical edges of  $P$  in sorted order.
2 for  $j = 1, \dots, k$  do
3   Let  $[y_0, y_1]$  be the interval of  $y$ -coordinates of  $e_j$ .
4   if the interior of  $P$  is to the left of  $e_j$ 
5     for each  $I_i \in \mathcal{I}$  that overlaps  $[y_0, y_1]$  do
6       if  $I_i$  and  $x(e_j)$  have different parity
7         return “no tiling”
8       Remove  $I_i$  from  $\mathcal{I}$ , let  $J := \overline{I_i \setminus [y_0, y_1]}$ , and if  $J \neq \emptyset$ , add the interval(s) in  $J$  to  $\mathcal{I}$ .
9   else
10    Make a new interval  $I := [y_0, y_1]$  with the parity  $p(I) := x(e_j) \bmod 2$  and add  $I$  to  $\mathcal{I}$ .
11    if  $I$  has one or two true neighbors in  $\mathcal{I}$  that also have the same parity as  $I$ 
12      Merge those intervals in  $\mathcal{I}$ .
13  if  $j < k$  and  $x(e_{j+1}) > x(e_j)$  and some  $I_i \in \mathcal{I}$  has odd length
14    return “no tiling”
15 return “tileable”

```

The above explanation of the algorithm argues that if the invariants hold before edge e_j is handled, they also hold after. It remains to argue that they also hold before the next edge e_{j+1} is handled in the case that the sweep line ℓ jumps to the right in order to sweep over e_{j+1} . In the open strip between the vertical lines containing e_j and e_{j+1} , there are no vertical segments of P . Hence, the pattern of the tiling $T(P)$ must continue as described by the parities $p(I_i)$ in between

the edges e_j and e_{j+1} , so the parity invariant also holds before e_{j+1} is handled.

We already argued that if the algorithm returns “no tiling”, then P is not tileable. Suppose on the other hand that the algorithm returns “tileable”. In order to prove that P can then be tiled, we define for each $j \in [k]$ a polyomino $P_j \subset P$. We consider the situation where the sweep line ℓ contains e_j and e_j has just been handled by the algorithm. We then define P_j to be the union of

- the part of P to the left of ℓ , and
- the rectangle $[x(\ell), x(\ell) + 1] \times I_i$ for each $I_i \in \mathcal{I}$ with a different parity than $x(\ell)$.

We first see that for each $j \in [k]$, we have $P_j \subset P$. To this end, we just have to check that the rectangles $[x(\ell), x(\ell) + 1] \times I_i$ are in P . If one such interval was not in P , there would be an edge of P overlapping the segment $\{x(\ell)\} \times I_i$. Since I_i has a different parity than $x(\ell)$, this would make the algorithm report “no tiling” at line 7, contrary to our assumption.

We now prove by induction on j that each P_j can be tiled. Since $P = P_k$, this is sufficient. Along the way, we will also establish that $P_1 \subset P_2 \subset \dots \subset P_k$. When $j = 1$, we see that P_j is empty, so the statement is trivial. Suppose now that P_j can be tiled and consider P_{j+1} . Note that if $x(e_j) = x(e_{j+1})$, so that ℓ does not move, then $P_j = P_{j+1}$, since all intervals that are created or modified when handling e_{j+1} have the same parity as $x(\ell)$, so in this case, P_{j+1} is tileable because P_j is.

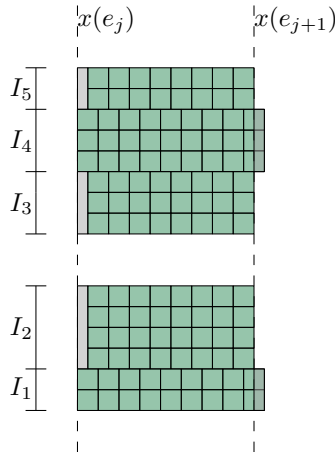


Figure 4: The polyomino P_j is the part of P to the left of the line $x = x(e_j)$ (this part of P_j is not shown) plus the grey rectangles along the line. Here, the difference $x(e_{j+1}) - x(e_j)$ is odd. The difference $\overline{P_{j+1} \setminus P_j}$ has been tiled with green 2×2 squares.

Consider now the case $x(e_j) < x(e_{j+1})$. Note that as $P_j \subset P$ and P_j is to the left of the vertical line $x = x(e_j) + 1$, we have $P_j \subset P_{j+1}$. We now consider the set $\overline{P_{j+1} \setminus P_j}$ and argue that it is tileable; see Figure 4. Let I_1, \dots, I_m be the intervals in \mathcal{I} after e_j was handled. For each I_i , we add a rectangle $X \times I_i$ to P_j in order to obtain P_{j+1} , where $X \subset \mathbb{R}$ is an interval with lower endpoint $x(e_j)$ or $x(e_j) + 1$ and upper endpoint $x(e_{j+1})$ or $x(e_{j+1}) + 1$, and by the definition of P_j and P_{j+1} , it follows that X has even length. Since each I_i also has even length (otherwise, the algorithm would have returned “no tiling” at line 14 when e_j was handled), the difference $\overline{P_{j+1} \setminus P_j}$ is a union of rectangles with even edge lengths, so P_{j+1} is tileable since P_j is.

Runtime analysis. Assuming that we can compare two coordinates in $O(1)$ time, we sort the vertical edges by their x -coordinates in $O(n \log n)$ time. Since the intervals of \mathcal{I} are pairwise interior-disjoint, we can implement \mathcal{I} as a balanced binary search tree, where each leaf stores an interval I_i .

We now argue that each vertical edge e_j , with y -coordinates $[y_0, y_1]$, takes only $O(\log n)$ time to handle, since we need to make only $O(1)$ updates to \mathcal{I} . If the interior of P is to the left of e_j , then $[y_0, y_1] \subset \bigcup_{i \in [m]} I_i$. It then follows from the neighbor invariant that if $[y_0, y_1]$ overlaps more than one interval I_i , then the algorithm will return “no tiling”. We therefore do at most $O(1)$ updates to \mathcal{I} , so it takes $O(\log n)$ time to handle e_j .

On the other hand, if the interior of P is to the right of e_j , we need to insert a new interval into \mathcal{I} and possibly merge it with one or two neighbors in \mathcal{I} , so this also amounts to $O(1)$ changes to \mathcal{I} .

At line 13, we need to check the $O(1)$ intervals that were added or changed due to the edge e_j , so this can be done in $O(1)$ time. Hence, the algorithm has runtime $O(n \log n)$.

Adaptation to $k \times k$ squares. In order to adapt the algorithm to $k \times k$ squares, we need to compare coordinates modulo k instead of modulo 2. Specifically, each interval in \mathcal{I} stores a number $p(I) \in \{0, 1, \dots, k-1\}$, which is set to $x(e_j) \bmod k$ at line 10. We fail at line 6 if $x(e_j) \bmod k \neq p(I_i)$ and at line 13 if some I_i has a length not divisible by k . At line 12, we merge I with the true neighbours that have the same p -value. With these modifications, all arguments carry over to the case of $k \times k$ squares.

4 Packing dominos

In this section we will present our polynomial time algorithm for finding the maximum number of 1×2 dominos that can be packed in a polyomino P . We assume that the dominos must be placed with axis parallel edges, but they can be rotated by 90° . In any such packing, we can assume the pieces to have integral coordinates: if they do not, we can translate the pieces as far down and to the left as possible, and the corners will arrive at positions with integral coordinates. We first describe a naive algorithm which runs in polynomial time in the area of the polyomino.

Naive algorithm. The naive algorithm considers the graph $G(P) = (V, E)$ where V is the set of cells of P and $e = (u, v) \in E$ if and only if the two cells u and v have a (geometrical) edge in common. The maximum number of 1×2 dominos that can be packed in P is exactly the size of a maximum matching of G and it is well known that such a maximum matching can be found in polynomial time in $|V|$, i.e., in the area of P .

Our goal is to find an algorithm running in polynomial time, even with the compact representation of P described in Section 2. In essence, we take the graph $G = G(P)$ above and construct from it a smaller graph, G^* , with $O(n^3)$ vertices. A maximum matching of G^* yields an (implicit) description of a maximum matching of G and we show that the maximum matching of G^* can be found in time $O(n^3 \text{ polylog } n)$.

4.1 Polynomial-time algorithm

We will next describe the steps of our algorithm for finding the maximum domino packing of a polyomino P . We first introduce the notion of a *pipe* (see Figure 5) and *consistent parity*.

Definition 1. Let P and Q be polyominoes with $Q \subset P$. We say that Q is a *pipe* of P if Q is rectangular and both vertical edges of Q or both horizontal edges of Q are contained in edges of P . The *width* of the pipe is the distance between this pair of edges. The *length* of the pipe is the distance between the other pair of edges. We say that a pipe is *long* if its length is at least 3 times its width.

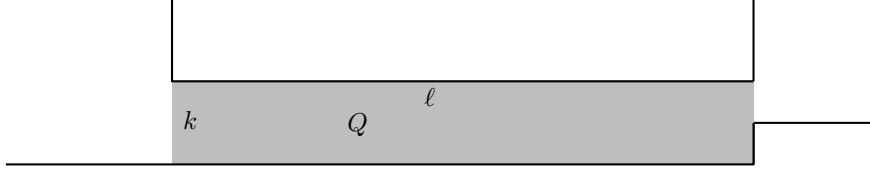


Figure 5: A pipe of width k and length ℓ .

Definition 2. We say that a polyomino P has *consistent parity* if all first coordinates of the corners of P have the same parity and vice versa for the second coordinates. Equivalently, P has consistent parity if there exists an open 2×2 square, S , such that for all choices of integers i, j and $S' = S + (2i, 2j)$, either $S' \subseteq P$ or $S' \cap P = \emptyset$.

Next we present the steps of the algorithm. Figures 6–8 demonstrates the steps on a concrete polyomino P .

Step 1: Compute the unique maximal polyomino $P_1 \subset P$ with all coordinates even.

We define P_1 to be the union of all 2×2 squares S of the form $S = [2i, 2i + 2] \times [2j, 2j + 2]$ with $i, j \in \mathbb{Z}$ and $S \subseteq P$. See the upper left and bottom part of Figure 6. It is readily checked that P_1 has at most n corners. As we will see, P_1 can be computed in time $O(n \log n)$.

Step 2: Compute a polyomino $P_2 \subset P_1$ with no holes and consistent parity by carving channels in P_1 .

Define $P'_0 := P_1$. For $i = 0, 1, \dots$, we do the following. If there are holes in P'_i , we find a set of minimum size of 2×2 squares S_1, \dots, S_k contained in P'_i and with even corner coordinates that connects an edge of a hole to an edge of the outer boundary of P'_i . To be precise, an edge of S_1 should be contained in the boundary of a hole of P'_i , an edge of S_k should be contained in the outer boundary of P'_i , and for each $j \in \{1, \dots, k - 1\}$, S_j and S_{j+1} should share an edge. We choose these squares such that they together form a $2 \times 2k$ or $2k \times 2$ rectangle or an L-shape, which is clearly always possible. We then define the polyomino $P'_{i+1} := P'_i \setminus \bigcup_{j=1}^k S_j$, which has less holes than P'_i . We stop when there are no more holes and define $P_2 := P'_i$ to be the resulting hole-free polyomino. Note that in iterations $i \geq 1$, the holes may get connected to holes that were eliminated in earlier iterations or to channels carved in earlier iterations. See the upper right part of Figure 6. We will later see that P_2 has strictly less than $3n$ corners and that it can be computed in time $O(n^3)$.

Step 3: Compute the offset $Q := B(P_2, -\lfloor 3n/2 \rfloor)$ and then $P_3 := P \setminus Q$.

See the left part of Figure 7. Note that we remove Q from the original polyomino P in order to get P_3 , and not from P_2 . It is easy to check that Q has at most $3n$ corners and consistent parity. Hence $P_3 := P \setminus Q$ has at most $4n$ corners and, as we will see, P_3 has the property that for any $x \in P_3$, we have $\text{dist}(x, \partial P_3) = O(n)$. We will show how this step can be carried out in time $O(n \log n)$.

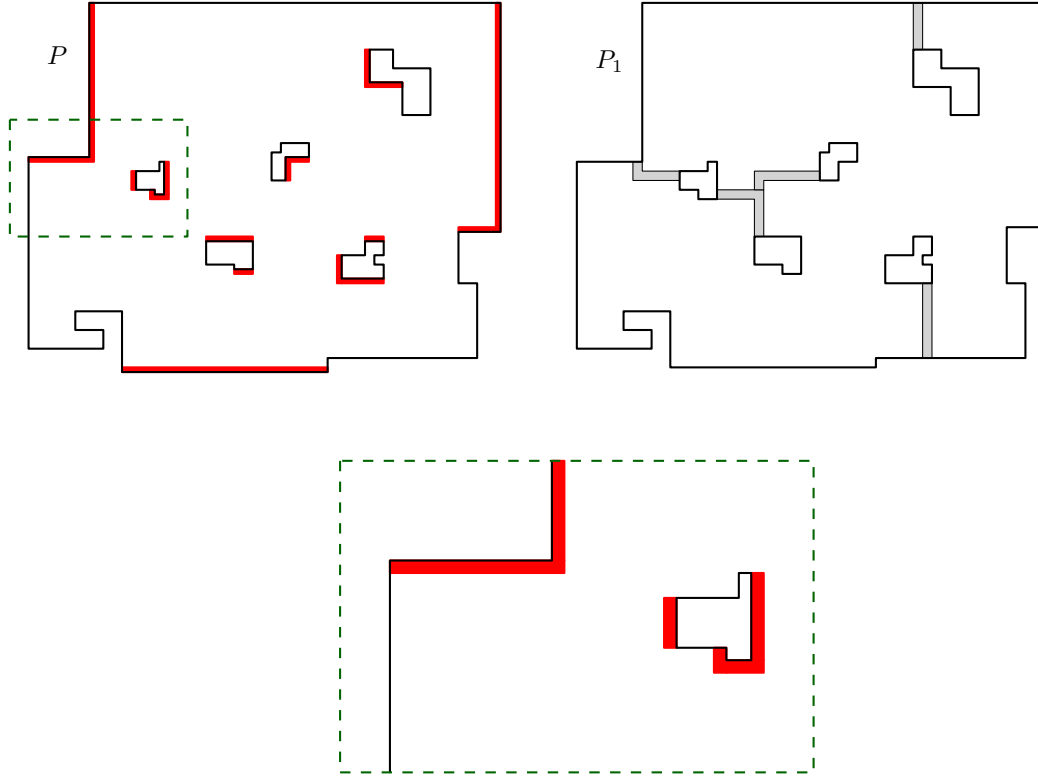


Figure 6: Steps 1 and 2 of the algorithm. Top left: Step 1, where the part $P \setminus P_1$, that is excluded from P_1 in order to make all coordinates even, is shown in red. Top right: Step 2, where the holes of a polyomino P_1 are connected to the outer boundary by the grey channels. Bottom: Closeup of the region in the dashed rectangle.

Step 4: Find the long pipes of P_3 . Find all maximal long pipes T_1, \dots, T_r in P_3 (recall that a pipe is long if its length is at least 3 times its width). See the right part of Figure 7. As we will see, there are at most $O(n)$ such pipes, they are disjoint, and they each have width $O(n)$. Later we will show how the pipes can be found in time $O(n \log n)$.

Step 5: Shorten the pipes and compute the associated graph G^* . Define $G_3 := G(P_3)$. We modify G_3 by performing the following shortening step for each $1 \leq i \leq r$; see Figure 8. Assume with no loss of generality that the pipe T_i is of the form $T_i = [0, \ell] \times [0, k]$ where ℓ is the length and $k \leq \ell/3$ is the width. If $\ell \leq 6$, we do nothing. Otherwise, for each $j \in \{0, \dots, k-1\}$, we let $S_j = [k+2, r] \times [j, j+1]$, where $r := 2\lceil \ell/2 \rceil - k - 2$, so that $G(S_j)$ is a horizontal path in G_3 consisting of an even number of vertices. For each $j \in \{1, \dots, k-1\}$, we proceed by deleting the vertices of S_j and their incident edges from G_3 , and instead, we add an edge from the cell $[k+1, k+2] \times [j, j+1]$ to the cell $[r, r+1] \times [j, j+1]$ (i.e., we connect the cells to the left and right of S_j with each other).

We denote the graph obtained after iterating over all i by G^* . Note that in G^* , there are only $O(k^2) = O(n^2)$ vertices corresponding to cells in each pipe T_i , since each pipe has width $k = O(n)$. We show below that G^* has $O(n^3)$ vertices and can be computed in time $O(n^3)$.

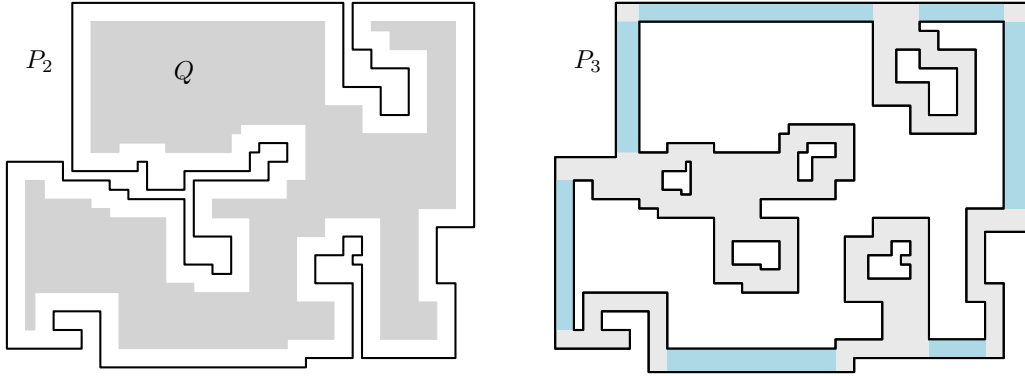


Figure 7: Steps 3 and 4 of the algorithm, performed on the instance from Figure 6. Left: Step 3, where the grey region Q is an offset of the hole-free polyomino P_2 . In this example, Q is connected, but that is in general not the case. For pedagogical reasons, we offset by a smaller value than the algorithm would actually use. Right: Step 4, where the grey and blue areas are $P_3 := P \setminus Q$. The blue rectangles show the seven long pipes.

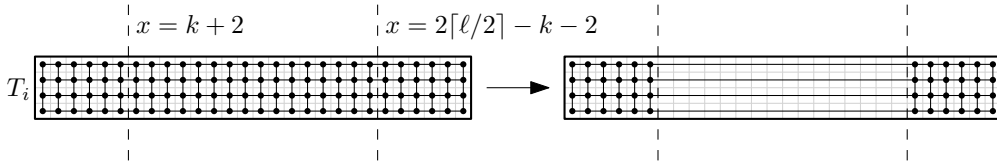


Figure 8: Step 5 of the algorithm. The part of the graph $G(T_i)$ in between the dashed vertical lines is substituted for long horizontal edges.

Step 6: Find the size of a maximum domino packing of P . We finally run a maximum matching algorithm on G^* . Let M be the resulting maximum matching, N_0 be the area of P , and N_2 be the number of vertices of G^* . The algorithm outputs $|M| + (N_0 - N_2)/2$ as the value of a maximum domino packing of P . We show below that this step can be performed in time $O(n^3 \text{ polylog } n)$.

This completes the description of the algorithm. In Section 4.2, we will provide some structural results on domino packings and polyominoes. In Section 4.3, we will use these results to argue that the algorithm works correctly. In Section 4.4, we will show that the reduced graph G^* has $O(n^3)$ vertices and edges. Finally, in Section 4.5, we will use this to argue how the steps of the algorithm can be implemented with the claimed running times.

4.2 Structural results on polyominoes and domino packings

Building up to our structural results on domino packings, we require a definition and a few simple lemmas. Variations of the following lemma is well-known. We present a proof for completeness.

Lemma 2. *Let P be an orthogonal polygon with n corners and h holes. P can be divided into at most $n/2 + h - 1$ rectangular pieces by adding only vertical line segments to the interior of P . If P is a polyomino, the rectangular pieces can be chosen to be polyominoes too.*

Proof. For each concave corner of the polygon we add a vertical line segment in the interior of the polygon starting from that corner and going upwards or downwards (depending on the rotation of

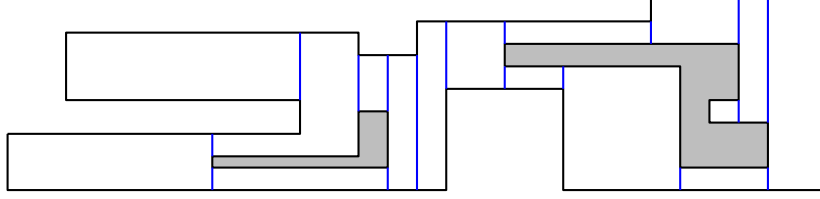


Figure 9: A partition of a polyomino with two holes into rectangles using vertical line segments (blue).

the given corner). This is illustrated in Figure 9. Let s be the number of line segments added. It is easy to check that this gives a partition of P into exactly $s - h + 1$ rectangles. With h holes, the number of concave corners is $n/2 + 2(h - 1)$, so also $s \leq n/2 + 2(h - 1)$ and the result follows. \square

Note that for a polygon with n corners, $h \leq (n - 4)/4$, so we have the following trivial corollary.

Corollary 3. *The number of rectangular pieces in Lemma 2 is at most $\frac{3}{4}n - 2$.*

We next show that the property of consistent parity is preserved under integral offsets.

Lemma 4. *Let P be a polyomino. If P has consistent parity, then $B(P, 1)$ and $B(P, -1)$ have consistent parity*

Proof. Suppose P has consistent parity. Let S be a 2×2 square as in Definition 2. Define $S_1 = S + (1, 1)$. It is easy to check that for all choices of integers i, j and $S'_1 := S_1 + (2i, 2j)$, either $S'_1 \subseteq B(P, 1)$ or $S'_1 \cap B(P, 1) = \emptyset$. Thus $B(P, 1)$ has consistent parity. The argument that $B(P, -1)$ has consistent parity is similar. \square

Lemma 5. *Let P be a connected polyomino of consistent parity and without holes. Define $L_1 = B(P, 1) \setminus P$ and $L_{-1} = P \setminus B(P, -1)$. Then $G(L_1)$ and $G(L_{-1})$ both have a Hamiltonian cycle of even length.*

Proof. To obtain a Hamiltonian cycle of $G(L_1)$, we can simply trace P around the outside of its boundary, visiting all cells of L_1 in a cyclic order. The corresponding closed trail of $G(L_1)$ visits each vertex at least once. The assumption of consistent parity is easily seen to imply that we in fact visit each vertex exactly once, so the obtained trail is a Hamiltonian cycle. The graph $G(L_1)$ is bipartite, so the cycle has even length. The argument that $G(L_{-1})$ has a Hamiltonian cycle of even length is similar. \square

With the above in hand, we are ready to state and prove our main structural results on domino packings. They are presented in Lemma 6 and Lemma 7.

Lemma 6. *Let P and P_0 be polyominoes such that $P_0 \subseteq P$, P_0 has no wholes, and P_0 has consistent parity. Let the total number of corners of P and P_0 be n . Define $r = \lfloor \frac{3}{8}n \rfloor$ and $Q = B(P_0, -r)$. There exists a maximum packing of P with 1×2 dominos which restricts to a tiling of Q .*

Let us briefly pause to explain the importance of Lemma 6. Suppose that P contains a region Q as described. Then Lemma 6 tells us that *any* domino tiling of Q can be extended to a maximum domino packing of P . We can thus disregard Q and focus on finding a maximum packing of $P \setminus Q$, thus reducing the problem to a smaller instance. This is one of our key tools for reducing the size of the original polyomino P to a matching problem of polynomial size. Another tool, namely to

contract long pipes, will be described in Lemma 7 below, and in Section 4.4, we will conclude that these two tools used carefully together reduce the packing problem to that of finding a maximum matching in a graph G^* of size $O(n^3)$.

Proof. It follows from Lemma 4 that Q has consistent parity, and it can thus be tiled with 2×2 squares and hence with dominos. Let \mathcal{Q} be a tiling of Q .

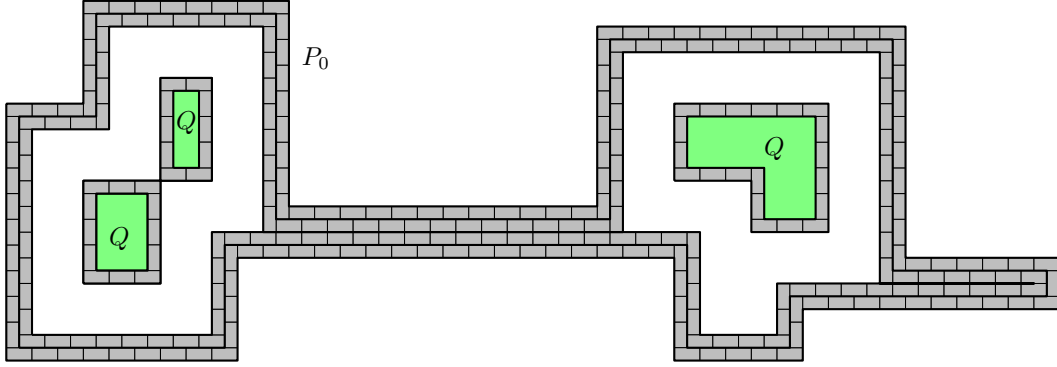


Figure 10: The polyomino P_0 and the offset Q (shown in green). The figure also illustrates the 'layers' A_i and their domino tilings, \mathcal{A}_i .

Define $R = P \setminus P_0$ and note that R has at most n corners. It follows from Corollary 3 that R can be partitioned into less than $\frac{3}{4}n$ rectangular polyominos. Each of these rectangles has a domino packing with at most one uncovered cell (which happens when the total number of cells in the rectangle is odd). Fix such a packing \mathcal{R} of the rectangles of R with dominos.

We next describe a tiling of $P_0 \setminus Q$ as follows. For integers $1 \leq i \leq r$ we define, $A_i = B(P_0, -i + 1) \setminus B(P_0, -i)$. Intuitively, we can construct Q from P_0 by peeling off the 'layers' A_i of P_0 one at a time. Let $i \in \{1, \dots, r\}$ be fixed. As P_0 has consistent parity, it follows from Lemma 4 that $B(P_0, -i + 1)$ has consistent parity. It is also easy to check that $B(P_0, -i + 1)$ has no holes either, and it then follows from Lemma 5 that each connected component of $G(A_i)$ has a Hamiltonian cycle of even length. These cycles give rise to a natural tiling of A_i ; if (v_1, \dots, v_{2k}) is the sequence of cells corresponding to such a cycle, then $\{v_1 \cup v_2, v_3 \cup v_4, \dots, v_{2k-1} \cup v_{2k}\}$ is a tiling of the cells of the cycle, and the union of such tilings over all connected components in $G(A_i)$ gives a tiling of A_i with dominos. Denote this tiling by \mathcal{A}_i . See Figure 10 for an illustration of this construction.

Combining the tilings $\mathcal{A}_1, \dots, \mathcal{A}_r$ and \mathcal{Q} with the packing \mathcal{R} , we obtain a domino packing, \mathcal{P} , of P where at most $\frac{3}{4}n$ cells of P are uncovered. We now wish to extend this packing to a maximum packing in a way where we do not alter the tiling \mathcal{Q} of Q . If we can do this, the result will follow. Let M be the matching corresponding to \mathcal{P} in $G(P)$. We make the following claim.

Claim. Let $k \leq r$. Suppose that the matching M can be extended to a matching of size $|M| + k$. Then this extension can be made using a sequence C_1, \dots, C_k of k augmenting paths one after the other (that is, C_i is an augmenting path *after* the matching has been extended using C_1, \dots, C_{i-1}) such that for each $i \in \{1, \dots, k\}$, we have that C_i only uses vertices of $G(R \cup \bigcup_{j=1}^i A_j)$.

Before proving this claim, we first argue how the result follows. Since there are less than $\frac{3}{4}n$ unmatched vertices in M , we can extend M to a maximum matching using at most $r = \lfloor \frac{3}{8}n \rfloor$ augmenting paths. By the claim, these paths can be chosen so that they avoid the vertices of $G(Q)$.

In particular, we never alter the matching of $G(Q)$, so the final maximum matching restricted to $G(Q)$ is just the tiling \mathcal{Q} .

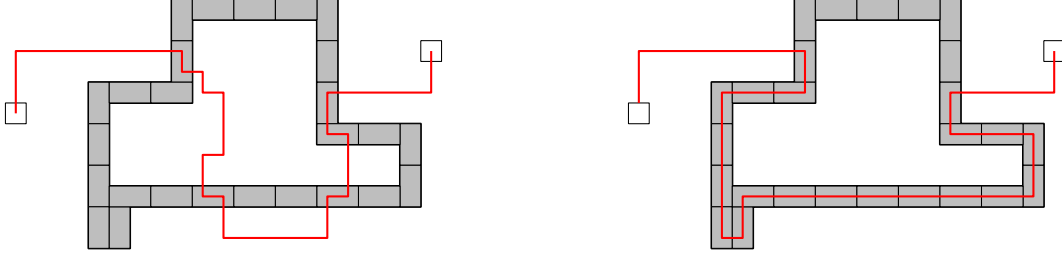


Figure 11: Left: An alternating path between two unmatched vertices which enters a connected component of $G(A_k)$. Right: Modifying the alternating path using the the Hamiltonian cycle of the connected component.

We proceed to prove the claim by induction on k . The statement is trivial for $k = 0$, so let $1 \leq k \leq r$ satisfy the assumptions of the claim and suppose inductively that C_1, \dots, C_{k-1} can be chosen such that for each $i \in \{1, \dots, k-1\}$, we have that C_i only uses vertices of $G(R \cup \bigcup_{j=1}^i A_j)$. After augmenting the matching using C_1, \dots, C_{k-1} , we have only modified the matching restricted to $G(R \cup \bigcup_{j=1}^k A_j)$. By Lemma 1, we can find an augmenting path C'_k connecting two unmatched vertices u, v of $G(P)$. We will modify C'_k to a path C_k with $C_k \subset R \cup \bigcup_{j=1}^k A_j$. Write $C'_k : u = u_1, u_2, \dots, u_{2\ell} = v$. Let D be a Hamiltonian cycle of one of the connected components of $G(A_k)$; see Figure 11. If the path C'_k ever enters the vertices of D , we let i be minimal such that $u_i \in D$ and j be maximal such that $u_j \in D$. We can now replace the subpath u_i, u_{i+1}, \dots, u_j of C'_k with part of the Hamiltonian cycle D . Whether we go clockwise or counterclockwise along D depends on whether u_i is matched with u_{i+1} in a clockwise or counterclockwise fashion in D . We do the same modification for every Hamiltonian cycle D corresponding to a connected component of $G(A_k)$ that that C'_k intersects. Note that each cycle D partitions the vertices $G(P) \setminus D$ into an interior and an exterior part. Since P_0 has no holes and $u, v \in R$, the original path C'_k enters D from the exterior at u_i and likewise leaves D into the exterior at u_j . Also note that Q is contained in the interior parts of the cycles of $G(A_k)$. It then follows that the final resulting path C_k avoids Q and A_j for $j > k$, so it is contained in $R \cup \bigcup_{j=1}^k A_j$. □

As it turns out, Lemma 6 is not in itself sufficient to yield a polyomino with area $n^{O(1)}$. For example, P may contain exponentially long and narrow pipes (see Definition 1), say of width $n/10$, which will remain even when Q is removed. Surprisingly, it turns out that such narrow pipes are the only obstacles that prevent us from reducing to an instance of size polynomial in n . This is what motivates the following lemma which intuitively yields a reduction for shortening long narrow pipes.

Lemma 7. *Let $k, \ell \in \mathbb{N}$ with ℓ even. Let $L \subseteq [-1, 0] \times [0, k]$, $R \subseteq [\ell, \ell + 1] \times [0, k]$ be polyominoes and define $P = L \cup R \cup ([0, \ell] \times [0, k])$. Color the cells of the plane in a chessboard like fashion and let b and w be respectively the number of black and white cells contained in P . Assume without loss of generality that $b \geq w$. If $\ell \geq 2k$, then the number of uncovered cells in a maximum domino packing of P is exactly $b - w$. Moreover, there exists a maximum domino packing such that the rectangle $[k + 1, \ell - k - 1] \times [0, k]$ is completely covered and all dominos intersecting the rectangle are horizontal.*

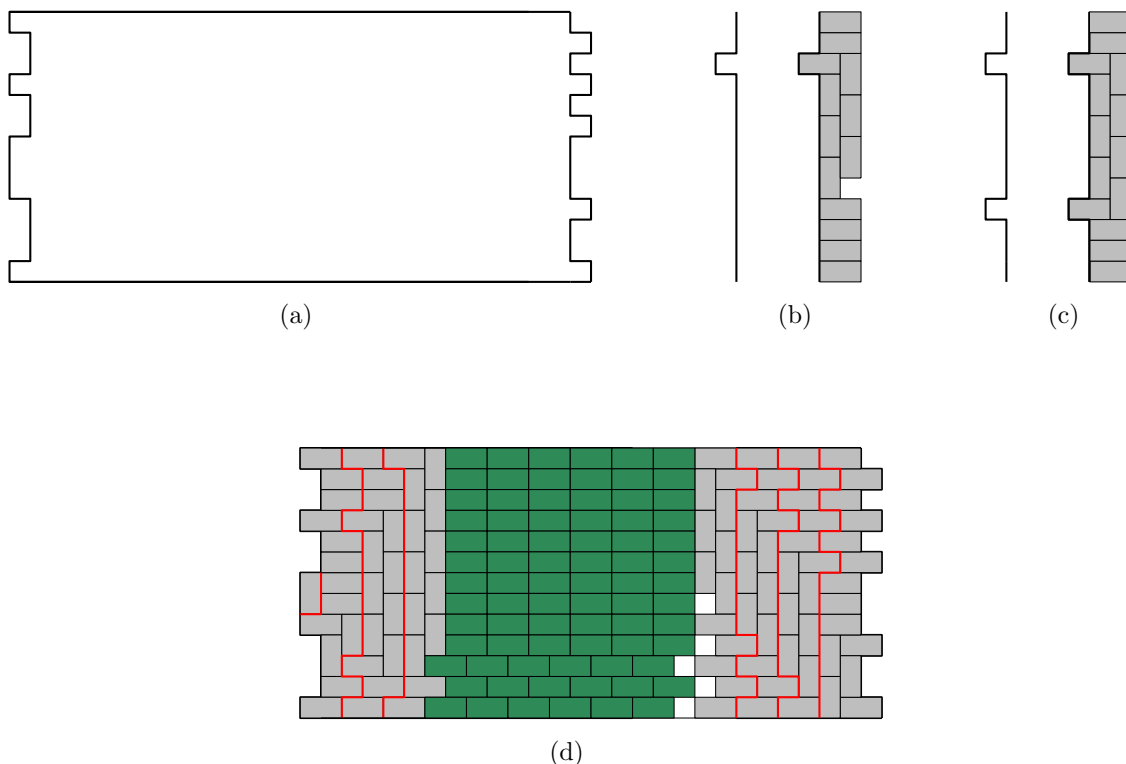


Figure 12: (a) The polyomino P . (b) Shifting a notch. (c) Cancelling two notches. (d) The partial packing obtained after shifting notches downwards and cancelling notches when possible (gray) and the horizontal dominos completing the packing (green).

Proof. As each domino covers one black and one white cell, any packing will leave at least $b - w$ cells uncovered. We thus need to demonstrate the existence of a packing with exactly $b - w$ uncovered cells. To see that such a packing exists, it is very illustrative to consider Fig. 12. An example of a polyomino, P , is illustrated in Fig. 12(a). We first tile as many cells of L and R as possible, such that no two uncovered cells of L and R share an edge. We will call these uncovered cells *notches*. We next show how we can alter the configuration of notches by only adding a layer of width 2. First, we note that a notch can be shifted an even number of cells downwards or upwards using the construction in Fig. 12 (b). In case we have two notches of different colours in the chessboard coloring and with no other notches between them, we can use the construction in Fig. 12 (c) to cancel these two notches from the configuration of notches. Our goal is to use the constructions of (b) and (c) to shift the notches of L and R downwards, cancelling notches if possible. Going through the notches of L from bottom to top, we shift them down as far as possible using the construction in (b). In case a notch has a different color than the nearest notch below it, we use construction (c) to cancel them. We further add horizontal dominos such that the configuration of notches is preserved at all other positions than where the shifting or cancelling occurs. We do a similar thing for R . The process from start to end is illustrated in Fig. 12(d) which also shows the resulting partial tiling. The red lines in the figure separate the steps of the process.

Note that each added layer of the process has thickness 2. Initially, each of L and R consists of at most $\lceil k/2 \rceil$ notches (after the first step which isolates the notches). Moreover, if k is odd and

there is $\lceil k/2 \rceil$ notches in L or R , then no shifting/cancelling is needed on that side. It then follows from the assumption $l \geq 2k$ that we are able to finish this partial packing. Let b' and w' be the number of black and white cells uncovered by this partial packing. Then $b' - w' = b - w$. It is also easy to check that we can complete the packing of P using only horizontal dominos and leaving exactly $b' - w' = b - w$ cells uncovered. This completes the proof. \square

Lemma 7 allows us to 'shorten' long and narrow pipes when searching for the maximum domino packing. It follows from the Lemma that going from G_3 to G^* in the shortening step 5 of our algorithm does not alter the number of unmatched vertices in a maximum matching. We will return to this in Section 4.3. Note that, unlike in Lemma 6, we cannot simply remove (part of) the pipe from the polyomino. The following lemma allows us to upper bound the size of a set of overlapping pipes no two of which are contained in the same larger pipe.

Lemma 8. *Let $G = (V, E)$ be a graph of order $n \geq 2$ with no self-loops but potential multiple edges. Suppose that G has a planar embedding such that for any pair of multiple edges (e_1, e_2) , the Jordan curve formed by e_1 and e_2 in the planar embedding of G contains a vertex of G in its interior. Then the number of edges of G is upper bounded by $3n - 5$.*

Proof. In what follows, we will use the classic result that the number of edges of a simple planar graph of order n is upper bounded by $3n - 6$. We prove the result by strong induction on n . For $n = 2$ the result is trivial so let $n > 2$ be given and suppose the bound holds for smaller values of n . Let \mathcal{E} be a planar embedding of G . Let (e_1, e_2) be a pair of distinct multiple edges that is minimal in the sense that no other such pair (e'_1, e'_2) exists with the following property: If γ and γ' are the Jordan curves formed by (e_1, e_2) and (e'_1, e'_2) in \mathcal{E} , then $\text{Int } \gamma' \subset \text{Int } \gamma$. Assume that e_1 and e_2 connect vertices u and v . Let V' be the set of vertices of G that are contained in $\text{Int } \gamma$ under \mathcal{E} and let $k = |V'|$. Then, $1 \leq k \leq n - 2$. Let $G_1 = (V_1, E_1)$ where $V_1 = V' \cup \{u, v\}$ and E_1 is formed by e_1 together with all edges of G that are incident to a vertex in V' . Let $G_2 = (V_2, E_2)$ where $V_2 = V \setminus V'$ and $E_2 = E \setminus E_1$. Clearly G_1 is a simple planar graph on $k + 2$ vertices. Moreover, it is readily checked that G_2 is a planar graph on $n - k$ vertices which satisfies the assumptions of the lemma. Note that $2 \leq n - k < n$. It thus follows from the inductive hypothesis that the number of edges of G is upper bounded by

$$3(n - k) - 5 + 3(k + 2) - 6 = 3n - 5.$$

This completes the proof. \square

Lemma 9. *Let P be a polyomino with n corners. Let Q_1, \dots, Q_r be pairwise disjoint pipes of P , no two of which are contained in a larger pipe of P , i.e., for no two distinct i, j does there exist a pipe Q with $Q_i \cup Q_j \subseteq Q$. Then $r \leq 3n - 5$.*

Proof. We construct a graph $G = (V, E)$ as follows. V is the set of (geometric) edges of P . For each $i \in \{1, \dots, r\}$ we let $u_i, v_i \in V$ be the two parallel edges of P which contain two opposite sides of Q_i and we add the edge (u_i, v_i) to E . G is thus a graph of order n with exactly r edges. We note that G may have multiple edges but it has a natural planar embedding, \mathcal{E} , such that for each pair of multiple edges (e_1, e_2) the Jordan curve formed by e_1 and e_2 under \mathcal{E} contains a vertex of G in its interior (see Figure 13). Here we used that for any two i, j with $1 \leq i < j \leq r$, the two pipes Q_i and Q_j are not contained in a larger pipe of P . It thus follows from Lemma 8 that $r \leq 3n - 5$. \square

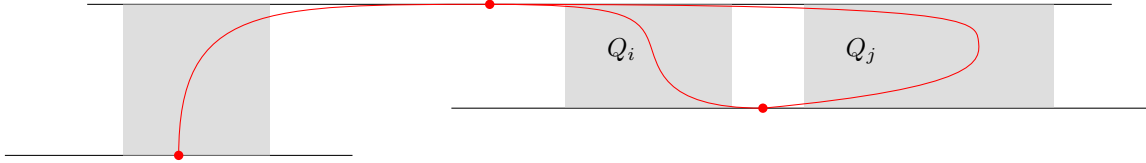


Figure 13: The construction of the planar graph G . Pipes are shown in grey. The white rectangle between, Q_i and Q_j must contain some nonempty subset of ∂P in its interior — otherwise this rectangle could be joined with Q_i and Q_j forming a larger pipe of P containing both Q_i and Q_j .

We need to argue that the algorithm outputs the correct value and that the different steps can be implemented to obtain the stated running times.

4.3 Correctness of the domino packing algorithm

We now refer the reader back to the description of our domino packing algorithm from Section 4.1 and show that it correctly finds the size of a maximum domino packing. To show this, it suffices to show that maximum matchings of G_0 and G^* , leave the same number of unmatched vertices. First note that P_1 has at most n corners. Further, a polyomino with n corners can have at most $(n - 4)/4$ holes, and since we remove a hole and add at most 6 new corners in going from P'_i to P'_{i+1} , P_2 has at most $5n/2 < 3n$ corners. It follows that also $Q = B(P_2, \lfloor 3n/2 \rfloor)$ has at most $3n$ corners. Letting n_1 denote the number of corners of $P_3 = P \setminus Q$ it finally follows that $n_1 \leq 4n$. Now $\text{dist}(\partial P, Q) \geq \lfloor 3n/2 \rfloor \geq \lfloor \frac{3}{8}n_1 \rfloor$ and moreover, Q has consistent parity and no holes, so Lemma 6 applies, giving that G_0 has a maximum matching, which restricts to a perfect matching of $G(Q)$ and to a maximum matching of G_3 . In particular, maximum matchings of G_0 and G_1 leave the same number of vertices unmatched.

Next, we argue that maximum matchings of G_3 and G^* again leave the same number of vertices unmatched. It is easy to see that a maximum matching of G^* can be extended to a matching of G_3 with the same number of unmatched vertices by simply inserting more horizontal dominos in the horizontal pipes and vertical dominos in the vertical pipes (here we use that the S_j 's as defined in Step 3, each consists of an even number of cells). Conversely, let M_1 be a maximum matching of G_3 . We show that G^* has a matching, M_2 , with the same number of uncovered cells. For this we consider the pipes $(T_i)_{i=1}^r$ found in step 4 of the algorithm. For each $1 \leq i \leq r$, we let T'_i be the pipe obtained from T_i by shortening T_i by one layer of cells in each end. The length of T'_i is thus two shorter than that of T_i . Let further $L_i \supseteq T'_i$ consist of all cells of P which are covered by a domino which cover at least one cell of T'_i . The sets $(L_i)_{i=1}^r$ are pairwise disjoint and they are each of the form of the set L in Lemma 7 (up to a 90 degree rotation). Moreover, the maximum matching M_1 restricts to a maximum matchings of M'_1 of $G(P_3 \setminus \bigcup_{i=1}^r L_i)$ and a maximum matching $M_1^{(i)}$ of $G(L_i)$ for $1 \leq i \leq r$. For $1 \leq i \leq r$, we let $G_3^{(i)} = G(L_i)$ and $G_2^{(i)}$ be the corresponding subgraph of G_2 . We define M_2 to be M'_1 combined with any maximum matchings of the $G_2^{(i)}$, $1 \leq i \leq r$. By applying Lemma 7 to each L_i , it follows that the maximum matchings of $G_3^{(i)}$ and $G_2^{(i)}$ leave the same number of unmatched vertices. It thus follows that M_2 and M_1 leave the same number of unmatched vertices. This finishes the argument that the algorithm works correctly.

4.4 Bounding the size of the reduced instance

In determining the running time of our algorithm, it is crucial to bound the size of the reduced instance G^* . In this section we show that G^* has $O(n^3)$ vertices. As explained in the next section, we can then find a maximum matching of G^* in $O(n^3 \text{ polylog } n)$ time.

We start out by proving the following lemma.

Lemma 10. *The polyomino P_3 contains no $63n \times 63n$ square subpolyomino.*

Proof. Let $n' = \lfloor 3n/2 \rfloor$. We show that P_3 contains no $41n' \times 41n'$ square as a subpolyomino and the desired result will follow. Suppose for contradiction that $S \subseteq P_3$ is such a subpolyomino. Note that Q consists of exactly those points of P_1 of distance at least n' to all the channels of $C := P_1 \setminus P_2$ and to ∂P_1 . Thus any point $x \in P_3$ has distance at most n' to C or to ∂P_1 . In particular S contains a $39n' \times 39n'$ square subpolyomino, $S_1 \subseteq P_1$, all points of which are of distance at least n' to ∂P_1 and thus, of distance at most n' to C .

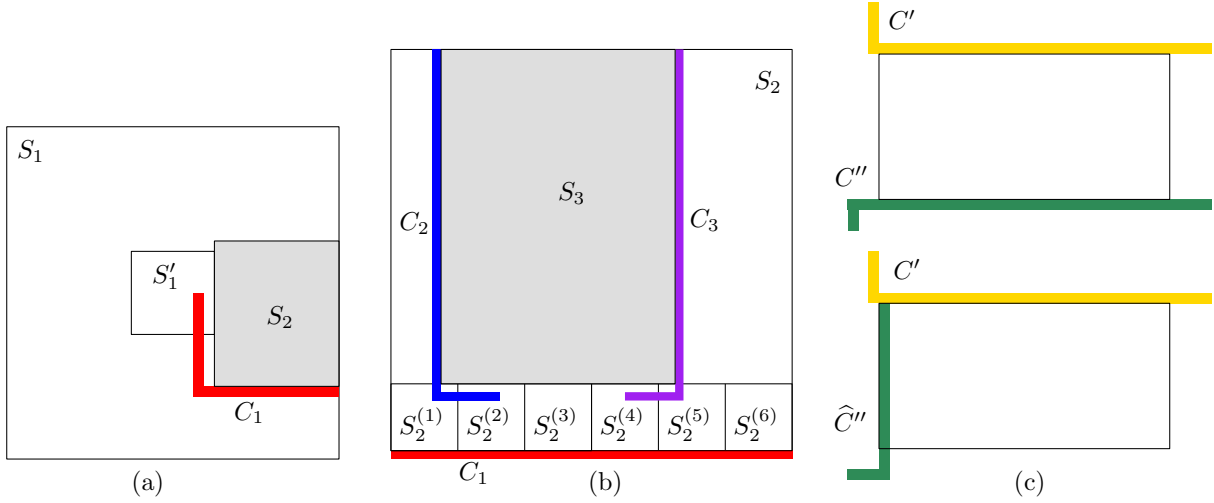


Figure 14: Situations in the proof of Lemma 10.

By the way we chose the channels, each channel connects a hole of P_1 with either the boundary of P_1 or with a channel already carved in an earlier iteration. Since $\partial P_1 \cap S_1 = \emptyset$, it follows that any channel intersecting S_1 has an end outside S_1 and thus leaves S_1 through an edge of S_1 .

Write S'_1 for the central $3n' \times 3n'$ square polyomino of S_1 ; see Figure 14 (a). Since any point of S_1 is of distance at most n' to C , S'_1 must intersect a channel $C_1 \subset C$ (depicted in red in the figure). We know that C_1 leaves S_1 . It is simple to check that this leads to the existence of an $18n' \times 19n'$ rectangular polyomino $S_2 \subseteq S_1$ having along one of its sides a straight part of the channel C_1 of length $18n'$; see Figure 14 (b). Assume with no loss of generality that $S_2 = [0, 18n'] \times [0, 19n']$ and that the channel C_1 runs along the base of the rectangle S_2 as in the figure. For $1 \leq i \leq 6$ we define $S_2^{(i)}$ to be the square polyomino $[3(i-1)n', 3in'] \times [0, 3n']$. By the same reasoning as above, each of these squares must intersect the set of channels C non-trivially. As each channel turns at most once by construction, the squares $S_2^{(i)}$ are disjoint from C_1 . To finish the proof, we require the following claim.

Claim. Let $B = [0, k] \times [0, \ell]$, $k, \ell \in \mathbb{N}$ be a $k \times \ell$ square polyomino. Suppose that $[0, k] \times [-1, 0]$ is contained in some channel, C' , and that $[0, k] \times [\ell, \ell + 1]$ is contained in some other channel, C'' . Then $k \leq \ell + 2$.

Proof of Claim. See Figure 14 (c). Suppose without loss of generality that C' was carved in iteration i and C'' was carved in iteration j in the process of generating P_2 in step 2 of the algorithm, and that $i < j$. The channel C'' was chosen to connect a yet unconnected hole H of P'_{j-1} with the outer boundary of P'_{j-1} along a shortest path in the L_∞ -norm. At the time C'' was carved, C' had already been carved and thus the edges of C' (except the two “ends” of length 2) is part of the outer face of P'_{j-1} . Under the assumption $k > \ell + 2$, we can find a shorter path connecting H to $\partial P'_{j-1}$; see the bottom part of Figure 14 (c). This shorter path shows that we could have picked a shorter channel \widehat{C}'' in place of C'' . This is a contradiction, so we conclude that $k \leq \ell + 2$. \square

Let us now finish the proof of the lemma. We know that the two squares $S_2^{(2)}$ and $S_2^{(5)}$ each intersect channels of C . Let us denote these not necessarily distinct channels respectively C_2 and C_3 . If these channels are the same, the channel C_2 passes straight through $S_2^{(3)}$. But then the two channels C_1 and C_2 run in parallel for a length of at least $3n' + 4$ and they have distance at most $3n'$ which gives a contradiction with the claim. Thus C_2 and C_3 are different channels. By the same reasoning as for C_1 , the channel C_2 must leave S_2 . If it does so in a direction parallel to C_1 , we similarly obtain a contradiction with the claim. Thus, it must leave S_2 in a direction perpendicular to C_1 . The same logic applies to C_3 ; see Figure 14 (b). Now the two channels C_2 and C_3 provide a contradiction to the claim. Indeed, their straight segments span a box, S_3 , of dimensions $\ell \times 18n'$ where $\ell \leq 18n' - 4$. With this contradiction, we conclude that P_3 contains no $41n' \times 41n'$ square as a subpolyomino and the proof is complete. \square

Remark. No serious effort has been made to optimize the constants in Lemma 10.

Corollary 11. *For any $x \in P_3$, we have $\text{dist}(x, \partial P_3) \leq 32n$.*

Proof. If not, P_3 contains a $63n \times 63n$ square, a contradiction. \square

Corollary 11 shows that each point of P_3 is of distance $O(n)$ to the boundary of P_3 . In particular, this shows that the long pipes, T_1, \dots, T_r , found in step 4 of the algorithm all have width at most $O(n)$. By Lemma 9, $r = O(n)$, so when performing the shortening reduction in step 5 of our algorithm, the part of G^* contained in contracted pipes gets size $O(n^3)$. We finish this section by showing that $P_3 \setminus \bigcup_{i=1}^r T_i$ also consists of $O(n^3)$ cells. From this it will follow that G^* is of order $O(n^3)$ which is what we require. We state the result as a lemma.

Lemma 12. *The reduced instance G^* found by our algorithm has $O(n^3)$ vertices and edges.*

Proof. For technical reasons to be made clear shortly we define T'_i to be the pipe obtained from T_i by shortening T_i by a layer of cells in each end. The length of T'_i is thus exactly the length of T_i minus 2. Let R be the polyomino $P_3 \setminus \bigcup_{i=1}^r T'_i$. Consider a (geometrical) edge, e , in the set $\partial R \setminus \partial P_3$ which is an edge forming an end of a shortened pipe T'_i . It then follows that the endpoints of e are corners of R and in particular that e is not contained in a longer edge of ∂R (this would not necessarily be the case if we had not shortened the pipes a layer in each end when defining R). We will use this observation shortly.

As discussed, it suffices to show that R has $O(n^3)$ cells. Note that R has $O(n)$ corners: Indeed, R is obtained from P_3 , which has $O(n)$ corners, by removing $O(n)$ pipes, each of which adds only 4 corners. We show that any point $x \in R$ is of distance $O(n)$ from a corner. Since each corner can have at most $O(n^2)$ cells within distance $O(n)$, this will show that R has $O(n^3)$ cells.

So let $x \in R$ be arbitrary. Also let $c := 32$. By Corollary 11, any point of P_3 is of distance at most cn to ∂P_3 . It follows that, similarly, any point of R is of distance at most cn to ∂R . Let S be the $6cn \times 6cn$ square centered at x and suppose that S contains no corner of R . Then $\partial R \cap S$ is a collection of horizontal and vertical straight line segments. Moreover, they are either all horizontal or all vertical as otherwise, they would intersect in a corner of R inside S . Assume without loss of generality that they are all horizontal. Using that any point of R is of distance at most cn to ∂R , it follows that there exists two such parallel segments of distance at most $2cn$, one being above x and one being below. Take a closest pair of such segments. Together they form a pipe, T , of R of length $6cn$ and width at most $2cn$, i.e., a pipe of a length at least three times its width. Now T is disjoint from the pipes T_1, \dots, T_r (since $T \subset R$ and each T'_i is disjoint from R). T is a pipe of R but we claim that it is in fact also a pipe of P_3 . To see this, we note that by Corollary 11, the pipes found in step 4 of our algorithm have width at most $2cn$. In particular, the edges of $\partial R \setminus \partial P_3$ have length at most $2cn$ and we saw that they are not contained in longer edges of ∂P_3 . However, the pipe T has length $6cn$ and so, the long edges of T are in fact edges of ∂P_3 , so T is a pipe of P_3 . This contradicts the maximality of the set of pipes T_1, \dots, T_r . We thus conclude that S must contain a corner of ∂R . Since $x \in R$ was arbitrary, this shows that any point in R is of distance at most $3cn = O(n)$ to a corner of R and the proof is complete. \square

4.5 Implementation of the individual steps

We next describe how the different step of our domino tiling algorithm can be implemented.

Step 1: Compute the unique maximal polyomino $P_1 \subset P$ with all coordinates even. We first compute the set $P_1 \subset P$ with consistent parity. To obtain P_1 , we move all corners of P to the interior of P to the closest points with even coordinates as shown in Figure 6.

Moving the corners may cause some corridors of P to collapse (namely the corridors of P of thickness 1), so that P_1 has overlapping edges corresponding to degenerate corridors. The degenerate vertical corridors can be filtered out in $O(n \log n)$ time as follows (and the degenerate horizontal ones are handled analogously). We sort the vertical edges after their x -coordinates and thus partition the edges into groups with identical x -coordinates. For each group of vertical edges with the same x -coordinate, we sort them according to the y -coordinates of their lower endpoints. Let e_1, \dots, e_k be one such sorted group. We run through the edges e_1, \dots, e_k in this order. For each edge e_i , we run through the succeeding edges until we get to an edge e_j which is completely above e_i . For each of the overlapping edges $e_k, k \in \{i+1, \dots, j-1\}$, we remove the corresponding degenerate corridor of P_1 created by e_i and e_k . Since no triple of edges can be pairwise overlapping, there are $O(n)$ overlapping pairs in total, so this process is dominated by sorting, which takes $O(n \log n)$ time.

Step 2: Compute a polyomino $P_2 \subset P_1$ with no holes and consistent parity by carving channels in P_1 . Remember that we need to find a set of minimum size of 2×2 squares S_1, \dots, S_k contained in P'_i and with even coordinates that connects an edge of a hole to an edge of the outer boundary of P'_i . For each pair of an edge of a hole of P'_i and an edge of the outer boundary of P'_i , we can compute the size of the smallest set of squares connecting those two edges in $O(1)$ time, so

by checking all pairs, we find the overall smallest set in $O(n^2)$ time. Note that no edge of a middle square S_j , $2 \leq j \leq k-1$, is contained in the boundary $\partial P'_i$, since otherwise, there would be a smaller set of squares with the desired properties. Therefore, constructing $P'_{i+1} := P'_i \setminus \bigcup_{j=1}^k S_j$ can then be done in $O(1)$ time once the squares S_j have been found. Since there are initially $O(n)$ holes to eliminate, the process takes $O(n^3)$ time in total.

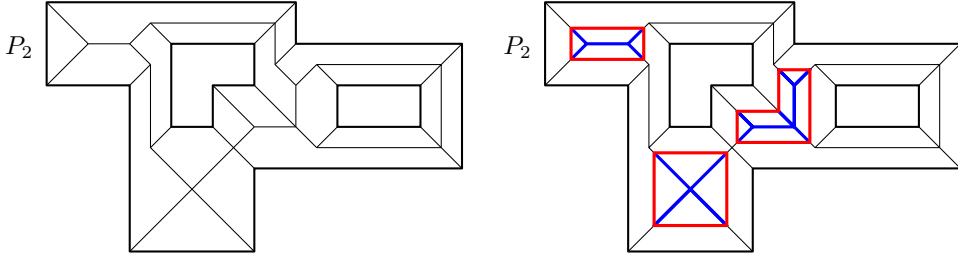


Figure 15: Left: The L_∞ Voronoi diagram $VD = VD(P_2)$ of the edges of a polyomino P_2 . Right: The blue parts of VD are the subgraphs that have some sufficient distance d to the boundary ∂P_2 . The red cycles enclose the regions of P_2 with at least distance d to ∂P_2 .

Step 3: Compute the offset $Q := B(P_2, -\lfloor 3n/2 \rfloor)$ and then $P_3 := P \setminus Q$. We now explain how to compute the set $Q \subset P_2$ defined by $Q := B(P_2, -\lfloor 3n/4 \rfloor)$. The boundary of Q can be computed from the L_∞ Voronoi diagram $VD := VD(P_2)$ of the edges of P_2 by a well-known technique described by Held, Lukács, & Andor [15], as follows. The Voronoi diagram VD is a plane graph contained in P_2 that partitions P_2 into one region $R(e_i)$ for each edge e_i of P_2 , such that if $x \in R(e_i)$ then $\text{dist}(x, e_i) = \text{dist}(x, \partial P_2)$, and VD consists of horizontal and vertical line segments and line segments that make 45° angles with the x -axis; see Figure 8 (left).

We find all maximal subgraphs of VD consisting of points with distance at least $\lfloor 3n/4 \rfloor$ to ∂P_2 . The leaves of each subgraph G lie on a cycle in P_2 where the distance to ∂P_2 is constantly $\lfloor 3n/4 \rfloor$, and the cycles can be found by traversing the leaves of G in clockwise order; see [15] for the details and Figure 8 (right) for a demonstration.

We can compute VD in $O(n \log n)$ time using the sweep-line algorithm of Papadopoulou & Lee [24]. In our special case where all edges are horizontal or vertical, the algorithm becomes particularly simple as described by Martínez, Vigo, Pla-García, & Ayala [20]. Once we have VD , it takes $O(n)$ time to compute ∂Q .

One can avoid the computation of VD by offsetting the boundary into the interior by distance 1 repeatedly $\lfloor 3n/4 \rfloor$ times. After each offset, we remove collapsed corridors as described in step 1. This would take in total $O(n^2 \log n)$ time.

The representation of $P_3 := P \setminus Q$ is obtained by simply adding the cycles representing the boundary of Q to the representation of P .

Step 4: Find the long pipes of P_3 . Recall that each long pipe $T_i \subset P_3$ is a maximal rectangle with a pair of edges contained in ∂P_3 which are at least 3 times as long as the other pair of edges.

To find the pipes, we compute the L_∞ Voronoi diagram $VD_1 := VD(P_3)$ of the edges of P_3 in $O(n \log n)$ time, as described in step 3. Consider a long pipe T_i . Assume without loss of generality that $T_i = [0, \ell] \times [0, k]$ where ℓ is the length and $k \leq \ell/3$ is the width. We now observe that the

segment $e := [k/2, \ell - k/2] \times k/2$ in the horizontal symmetry axis of T_i is contained in an edge of VD_1 , since for any point p in e , the edges of P_3 closest to p are the horizontal edges of T_i .

Each horizontal or vertical edge e of VD_1 separates the regions of points that are closest to a pair s_1, s_2 of horizontal or vertical edges of P_3 . It is easy to check whether s_1, s_2 define a long pipe containing (a part of) e . Hence, all pipes can be identified by traversing the edges of VD_1 . As VD_1 has complexity $O(n)$, this step takes $O(n \log n)$ time in total.

Step 5: Shorten the pipes and compute the associated graph G^* . Recall that we define $G_3 := G(P_3)$ and obtain the final graph G^* by replacing long horizontal (resp. vertical) paths in pipes with long horizontal (resp. vertical) edges. Once P_3 and the long pipes have been computed, it is straightforward to construct G^* in $O(n^3)$ time, since the size of G^* is $O(n^3)$ by Lemma 2.

Step 6: Find the size of a maximum domino packing of P . Recall that our algorithm outputs $|M| + (N_0 - N_2)/2$, where M is a maximum matching of G^* , N_0 is the area of P , and N_2 is the number of vertices of G^* . In order to compute M , we use the multiple-source multiple-sink maximum flow algorithm by Borradaile, Klein, Mozes, Nussbaum, & Wulff-Nilsen [6]. In a directed plane graph with m vertices and edge capacities, where a subset of the vertices are *sources* and another subset are *sinks*, the algorithm finds the maximum flow from the sources to the sinks respecting the capacities in time $O(m \log^3 m)$. Gawrychowski & Karczmarz [13] described an improved algorithm with running time $O(m \frac{\log^3 m}{\log^2 \log m})$. Our graph G^* is bipartite, so a maximum matching equals a maximum flow between the two vertex classes, when each edge has capacity 1. We can therefore find the maximum matching of G^* in time $O(n^3 \text{polylog } n)$.

Remark. We note that we can also find an (implicit) description of a maximum domino packing of P . We simply extend the matching of G^* by inserting horizontal dominos in the horizontal pipes and vertical dominos in the vertical pipes. We further decide to give Q any standard tiling, e.g., the one that uses only horizontal dominos. It follows from the correctness of the algorithm that this gives a maximum domino packing of P .

4.6 Simpler but slower algorithm

Using the structural results on domino packings, we are able to prove the correctness of the following much simpler algorithm, which works by truncating the long edges of P . We sort the corners of P by x -coordinates and consider the corners in this order c_1, \dots, c_n . When $x(c_{i+1}) - x(c_i) > 9n$, we move all the corners c_{i+1}, \dots, c_n to the left by a distance of $2 \lfloor \frac{x(c_{i+1}) - x(c_i)}{2} \rfloor - 6n$. We call this operation a *contraction*. The result after all of the contractions is a polyomino P' with the parities of the x -coordinates unchanged and with the difference between the x -coordinates of any two consecutive corners at most $6n$. We then consider the corners in order according to y -coordinates and do a similar truncation of the long vertical edges. We have now reduced the container P to an orthogonal polygon P'' of area at most $O(n^4)$, since the span of the x -coordinates is $O(n^2)$, as is the span of the y -coordinates. We then compute a maximum matching M in the graph $G(P'')$ and return $|M| + \frac{\text{area}(P) - \text{area}(P'')}{2}$ as the size of a maximum packing in P . Using the multiple-source multiple-sink maximum flow algorithm to compute the matching M , this leads to an algorithm with running time $O(n^4 \text{polylog } n)$.

We now verify that the number of uncovered cells in maximum packings is invariant under a single contraction, and the correctness of the algorithm hence follows. To this end, suppose that

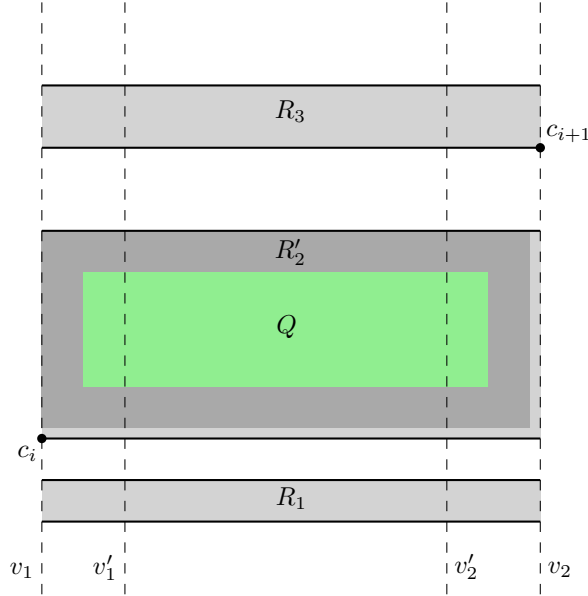


Figure 16: A contraction of the simple algorithm with one fat and two skinny rectangles. The algorithm moves all corners c_{i+1}, \dots, c_n to the left, essentially contracting the area between the vertical lines v'_1 and v'_2 to nothing.

$x(c_{i+1}) - x(c_i) > 9n$, so that we move the corners c_{i+1}, \dots, c_n to the left; see Figure 16. Let v_1 and v_2 be vertical lines with x -coordinates $x(c_i)$ and $x(c_{i+1})$, respectively, and let V be the vertical strip bounded by v_1 and v_2 . The intersection $P \cap V$ is a collection of disjoint rectangles R_1, \dots, R_k of width $x(c_{i+1}) - x(c_i)$ and various heights. We define a rectangle R_i to be *fat* if its height is more than $3n$, and otherwise R_i is *skinny*. We now define a polyomino P_0 in order to apply Lemma 6. For each fat rectangle R_i , we let $R'_i \subseteq R_i$ be the maximum rectangle with even coordinates and add R'_i to P_0 . As each rectangle R_i corresponds to exactly two horizontal edges, the number of rectangles k is upper bounded by $n/4$ and in particular, the number of corners of $P \setminus P_0$ is at most $2n$. Letting $Q := B(P_0, -\lfloor 3n/2 \rfloor)$, we get from Lemma 6 that there exists a maximum tiling of P that restricted to Q is a tiling.

We define $P_1 := P \setminus Q$ and observe that the contraction corresponds to contracting a set of long pipes in P_1 . These pipes are the skinny rectangles R_i and the parts of the fat rectangles vertically above and below the removed part Q . We therefore get from Lemma 7 that the number of uncovered cells is invariant under the contraction.

For some containers P , the graph $G(P'')$ really has $\Omega(n^4)$ vertices, so the algorithm is slower than the complicated algorithm. For instance when the boundary of P consists of four “staircases”, each consisting of $n/4$ vertices, where each step has width and height n ; see Figure 17 (left). Here the complicated algorithm will remove most of the interior, leaving a layer of cells of thickness $O(n)$ around the boundary, but the simple algorithm will not make any contractions.

One might be tempted to think that we can even truncate the edges so that the difference between consecutive x - and y -coordinates is either 1 or 2, keeping the parity of all coordinates. However, this does not work, as seen in Figure 17 (right). Two dominos can be packed in the reduced container P' , and the reduction decreases the area by eighth cells, so the formula would give

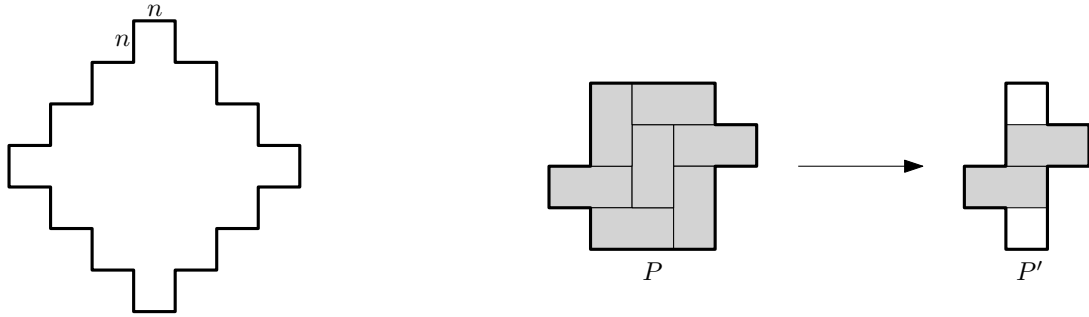


Figure 17: Left: A polyomino with area $\Omega(n^4)$ that the simple algorithm will not reduce. Right: If we truncate edges so that consecutive x -coordinates have difference either 1 or 2 (keeping the parities invariant), then there may be more uncovered cells in a maximum packing of the reduced instance than in the original.

that the original container P has room for six dominos, but there is actually room for seven.

References

- [1] Danièle Beauquier and Maurice Nivat. On translating one polyomino to tile the plane. *Discrete & Computational Geometry*, 6:575–592, 1991.
- [2] Danièle Beauquier, Maurice Nivat, Eric Remila, and Mike Robson. Tiling figures of the plane with two bars. *Computational Geometry*, 5(1):1–25, 1995.
- [3] Robert Berger. The undecidability of the domino problem. *Memoirs of the American Mathematical Society*, 1(66), 1966.
- [4] Fran Berman, David Johnson, Tom Leighton, Peter W. Shor, and Larry Snyder. Generalized planar matching. *Journal of Algorithms*, 11(2):153–184, 1990.
- [5] Francine Berman, Frank Thomson Leighton, and Lawrence Snyder. Optimal tile salvage, 1982. Technical report, Purdue University, Department of Computer Sciences, <https://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1321&context=cstech>.
- [6] Glencora Borradaile, Philip N. Klein, Shay Mozes, Yahav Nussbaum, and Christian Wulff-Nilsen. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. *SIAM Journal on Computing*, 46(4):1280–1303, 2017.
- [7] Chen-Fu Chien, Shao-Chung Hsu, and Jing-Feng Deng. A cutting algorithm for optimizing the wafer exposure pattern. *IEEE Transactions on Semiconductor Manufacturing*, 14(2):157–162, 2001.
- [8] J.H Conway and J.C Lagarias. Tiling with polyominoes and combinatorial group theory. *Journal of Combinatorial Theory, Series A*, 53(2):183 – 208, 1990.
- [9] Dirk K. de Vries. Investigation of gross die per wafer formulas. *IEEE Transactions on Semiconductor Manufacturing*, 18(1):136–139, 2005.

- [10] Dania El-Khechen, Muriel Dulieu, John Iacono, and Nikolaj Van Omme. Packing 2×2 unit squares into grid polygons is NP-complete. In *Proceedings of the 21st Canadian Conference on Computational Geometry (CCCG 2009)*, pages 33–36, 2009.
- [11] Robert J. Fowler, Michael S. Paterson, and Steven L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Information processing letters*, 12(3):133–137, 1981.
- [12] George Gamow and Marvin Stern. *Puzzle-math*. Macmillan, 1958.
- [13] Pawel Gawrychowski and Adam Karczmarz. Improved bounds for shortest paths in dense distance graphs. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, pages 61:1–61:15, 2018.
- [14] S. W. Golomb. Checker boards and polyominoes. *The American Mathematical Monthly*, 61(10):675–682, 1954.
- [15] Martin Held, Gábor Lukács, and László Andor. Pocket machining based on contour-parallel tool paths generated by means of proximity maps. *Computer-Aided Design*, 26(3):189–203, 1994.
- [16] Dorit S. Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM (JACM)*, 32(1):130–136, 1985.
- [17] Takashi Horiyama, Takehiro Ito, Keita Nakatsuka, Akira Suzuki, and Ryuhei Uehara. Packing trominoes is NP-complete, #P-complete and ASP-complete. In *24th Canadian Conference on Computational Geometry (CCCG 2012)*, pages 211–216, 2012.
- [18] S. Jang, J. Kim, T. Kim, H. Lee, and S. Ko. A wafer map yield prediction based on machine learning for productivity enhancement. *IEEE Transactions on Semiconductor Manufacturing*, 32(4):400–407, 2019.
- [19] C. Kenyon and R. Kenyon. Tiling a polygon with rectangles. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science (FOCS 1992)*, pages 610–619, 1992.
- [20] J. Martínez, M. Vigo, N. Pla-García, and D. Ayala. Skeleton computation of an image using a geometric approach. In *31st Eurographics (EG 2010)*, 2010.
- [21] Hanno Melzner and Alexander Olbrich. Maximization of good chips per wafer by optimization of memory redundancy. *IEEE Transactions on Semiconductor Manufacturing*, 20(2):68–76, 2007.
- [22] Igor Pak, Adam Sheffer, and Martin Tassy. Fast domino tileability. *Discrete & Computational Geometry*, 56(2):377–394, 2016.
- [23] Igor Pak and Jed Yang. Tiling simply connected regions with rectangles. *Journal of Combinatorial Theory, Series A*, 120(7):1804 – 1816, 2013.
- [24] Evanthia Papadopoulou and D.T. Lee. The L_∞ Voronoi diagram of segments and VLSI applications. *International Journal of Computational Geometry & Applications*, 11(05):503–528, 2001.

- [25] Eric Rémila. Tiling a polygon with two kinds of rectangles. *Discrete Comput. Geom.*, 34(2):313–330, 2005.
- [26] William P. Thurston. Conway’s tiling groups. *The American Mathematical Monthly*, 97(8):757–773, 1990.
- [27] H.A.G. Wijshoff and J. van Leeuwen. Arbitrary versus periodic storage schemes and tessellations of the plane using one type of polyomino. *Information and Control*, 62(1):1–25, 1984.