Thèse n° 9159

Space-Efficient Representations of Graphs

Présentée le 13 septembre 2022

Faculté informatique et communications
Laboratoire de théorie du calcul 4
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

Jakab TARDOS

École
polytechnique
fédérale
de Lausanne

2022

Computer science is no more about computers
than astronomy is about telescopes.
— Edsger Dijkstra

To my sister, Tamara…

Acknowledgements

Completing a Ph.D. is a long and arduous journey, and would have been neither possible nor worth doing without all the people that helped me along the way. I am immensely thankful to all those who have contributed either to my work or to my life during these past five years.

First and foremost, I would like to express my gratitude to my advisor, Michael Kapralov, for his continued guidance, encouragement, and motivation throughout my time at EPFL. One could not ask for a better mentor. His apparent love of research, and his limitless energy made him a truly inspiring person to work with. It never ceased to amaze me how many projects he could juggle simultaneously while still being able to meet regularly on each of them, and come up with brilliant insights and helpful suggestions. I'm also grateful to the culture of collaboration he cultivated, always encouraging us to share, discuss, and work together on projects; this I think is one of the biggest reasons I found my work at EPFL so enjoyable.

I would also like to thank my other jury members, Mika Göös, Sanjeev Khanna, Ola Svensson, and Luca Trevisan for their time and effort in reading this thesis, as well as their insightful questions and comments during the defense.

Research is not a solo endeavor, and I could never have produced the quality and quantity of work that this thesis represents without my many collaborators. It is no accident that this is the only page of my thesis written in the singular first person. Both for directly contributing to the works contained in the following chapters, and perhaps more importantly, for shaping me as a researcher through our many discussions, I am indebted to my collaborators. For this reason, I would like to express my immense gratitude to Marwa El Halabi, Robert Krauthgamer, Silvio Lattanzi, Mikhail Makarov, Gilbert Maystre, Slobodan Mitrović, Navid Nouri, Ashkan Norouzi-Fard, Kshiteej Sheth, Aaron Sidford, Jakub Tarnawski, and Yuichi Yoshida.

In addition to my collaborators, I would like also to thank everyone in the extended theory of computer science laboratory of EPFL, who created a wonderful work environment, sharing their research through the weekly coffee talks, as well as through informal discussions. I am grateful also to the IC administrative staff, and especially to Pauline Raffestin, for helping me navigate all the bureaucratic aspects of life at EPFL, and for always being the most cheerful and friendly person at the office.

Beyond those who have made this work possible, I must not neglect to acknowledge those who have made these past five years some of the most enjoyable years of my life. I will not attempt the impossible task of listing here all of my flatmates, colleagues, and friends from both Lausanne and Budapest. Suffice it to say that throughout all the lunch breaks, evenings, weekends, and vacations, I was never short of things to do and people to hang out with. With their help, I think I was able to achieve the elusive goal of maintaining a work life balance during a Ph.D. Special thanks to Karen, who had the misfortune of graduating a few months before me, and since then she has been valiantly answering all of my stupid questions on theses, committees, defenses, and all the associated deadlines.

I am especially happy and grateful to have my loving, amazing, and hilarious girlfriend Elene in my life. She has been there to share in the stress and excitement of Ph.D. life and to provide moral (and often practical) support in hard times of deadlines. But more importantly, she was there after work, climbing or swimming or cooking with me – doing all the things that make life worth living.

Finally, I would like to thank my family for all their love and support. My sisters, Vera and Tamara (to whom this thesis is dedicated), have been great fun and company in the little time we were able to spend together these last five years. To my parents, Gábor and Zsuzsa, I'm grateful that they nurtured and encouraged my interest in mathematics from an early age – it is undoubtedly thanks to their work that I even attempted this challenge. I am also grateful to my grandmother, Anna, who showed great interest in the technical details of my research, every time I visited her.

Lausanne, August 22, 2022 Jakab Tardos

Abstract

With the increasing prevalence of massive datasets, it becomes important to design algorithmic techniques for dealing with scenarios where the input to be processed does not fit in the memory of a single machine. Many highly successful approaches have emerged in recent decades, such as processing the data in a stream, parallel processing, and data compression. The aim of this thesis is to apply these techniques to various important graph theoretical problems. Our contributions can be broadly classified into two categories: spectral graph theory, and maximum matching.

Spectral Graph Theory. Spectral sparsification is a technique of rendering an arbitrary graph sparse, while approximately preserving the quadratic form of the Laplacian matrix. In this thesis, we extend the result of Kapralov et al. (2017a), and propose a sketch and corresponding decoding algorithm for constructing a spectral sparsifier from a dynamic stream of edge insertions and deletions. The size of the resulting sparsifier, the size of the sketch, and the decoding time are all nearly linear in the number of vertices, and consequently nearly optimal.

The concept of spectral sparsification has recently been generalized to hypergraphs (Soma and Yoshida (2019)) – an analogue of graphs for modeling higher order relationships. As one of the main contributions of the thesis, we prove for the first time the existence of nearly-linear sized spectral sparsifiers for arbitrary hypergraphs, and provide a corresponding nearly-linear time algorithm for constructing them. Through a lower bound construction, we show that our sparsifiers achieve nearly-optimal compression of the hypergraph spectral structure.

On the more applied side of spectral graph theory, we present a fully scalable MPC (massively parallel computation) algorithm which is capable of simulating a large number of independent random walks of length ℓ from an arbitrary starting distribution in $O(\log \ell)$ rounds.

Maximum Matching. We propose a novel randomized composable coreset for the problem of maximum matching, called the *matching skeleton*. The coreset achieves an $\approx 1/2$ approximation, while having fewer than n edges.

We also propose a new, highly space-efficient variant of a peeling algorithm for maximum matching. With this, we are able to approximate the maximum matching size of a graph to within a constant factor, using a stream of m uniformly random edges (where m is the total number of edges), in as little as $O(\log^2 n)$ space. Conversely, we show that significantly fewer (that is $m^{1-\Omega(1)}$) samples do not suffice, even with unlimited space. Finally, we design a Local Computation Algorithm, which implicitly construct a constant-approximate maximum matching in time and space that is nearly linear in the maximum degree.

Keywords: Sparsification, Streaming Algorithms, Sublinear Algorithms, Parallel Computation, Local Computation Algorithms, Graph Sketching, Hypergraphs, Maximum Matching, Random Walks

Résumé

Avec la prévalence croissante des ensembles de données massives, il devient important de concevoir des techniques algorithmiques pour traiter les scénarios où l'entrée à traiter ne tient pas dans la mémoire d'une seule machine. De nombreuses approches très efficaces ont vu le jour au cours des dernières décennies, comme le traitement des données dans un flot, le traitement parallèle et la compression des données. L'objectif de cette thèse est d'appliquer ces techniques à divers problèmes importants de la théorie des graphes. Nos contributions peuvent être classées en deux grandes catégories : la théorie spectrale des graphes et la couplage maximum.

Théorie Spectrale des Graphes. La sparsification spectrale est une technique permettant de rendre un graphe arbitraire creux, tout en préservant approximativement la forme quadratique de la matrice Laplacienne. Dans cette thèse, nous étendons le résultat de Kapralov et al. (2017a), et proposons une *sketch* et un algorithme de décodage correspondant pour construire un sparsifieur spectral à partir d'un flot dynamique d'insertions et de délétions d'arêtes. La taille du sparificateur résultant, la taille du *sketch* et le temps de décodage sont tous presque linéaires dans le nombre de sommets, et par conséquent presque optimaux.

Le concept de sparsification spectrale a récemment été généralisé aux hypergraphes (Soma and Yoshida (2019)) - un analogue des graphes pour la modélisation des relations d'ordre supérieur. L'une des principales contributions de cette thèse est de prouver pour la première fois l'existence de sparificateurs spectraux de taille quasi-linéaire pour des hypergraphes arbitraires, et de fournir un algorithme correspondant en temps quasi-linéaire pour les construire. Grâce à la construction d'une borne inférieure, nous montrons que nos sparificateurs permettent une compression presque optimale de la structure spectrale de l'hypergraphe.

Du côté plus appliqué de la théorie spectrale des graphes, nous présentons un algorithme *MPC* (calcul massivement parallèle) *fully scalable*, capable de simuler un grand nombre de marches aléatoires indépendantes de longueur ℓ à partir d'une distribution de départ arbitraire en $O(\log \ell)$ tours.

Couplage Maximum. Nous proposons un nouveau *composable coreset* randomisé pour le problème de la couplage maximum, appelé le *matching skeleton*. Le coreset atteint une approximation de $\approx 1/2$, tout en ayant moins de n arêtes.

Nous proposons également une nouvelle variante très efficace en termes d'espace de l'algorithme de pelage pour le couplage maximum. Avec cet algorithme, nous sommes capables d'approximer la taille maximum du couplage d'un graphe à un facteur constant près, en utilisant un flot de m arêtes uniformément aléatoires (où m est le nombre total d'arêtes), dans un espace aussi réduit que $O(\log^2 n)$. À l'inverse, nous montrons qu'un nombre significativement plus faible d'échantillons (c'est-à-dire $m^{1-\Omega(1)}$)

ne suffit pas, même avec un espace illimité. Enfin, nous concevons un algorithme de calcul local, qui construit implicitement un couplage maximale approximative constante en temps et en espace, qui est presque linéaire dans le degré maximal.

Mots clés : Sparsification, Algorithmes de Streaming, Algorithmes Sous-Linéaires, Calcul Parallèle, Algorithmes de Calcul Local, *Sketching* des Graphs, Hypergraphes, Couplage Maximum, Marches Aléatoires

Contents

Acknowledgements **i**

Abstract (English/Français) **iii**

1 Introduction **1**

1.1 Overview of Our Contributions . . . 3

1.1.1 Spectral Graph Theory . . . 3

1.1.2 Maximum Matching . . . 7

I Spectral Graph Theory **11**

2 Spectral Sparsification via Sketching **13**

2.1 Introduction . . . 13

2.2 Preliminaries . . . 16

2.3 Main result . . . 18

2.3.1 Overview of the approach . . . 18

2.3.2 Our algorithm and proof of main result . . . 22

2.3.3 Maintenance of sketches . . . 32

2.3.4 Proof of Theorem 2.1.1 . . . 32

3 Spectral Sparsification of Hypergraphs **35**

3.1 Introduction . . . 35

3.1.1 Additional Related Work . . . 38

3.2 Preliminaries . . . 39

3.2.1 Spectral Graph Theory . . . 39

3.2.2 Effective Resistance . . . 40

3.2.3 Chernoff Bound . . . 40

3.3 Technical Overview . . . 41

3.3.1 Analyzing Ordinary Graphs . . . 41

3.3.2 Extension to Hypergraphs . . . 43

3.3.3 Speed-Up . . . 44

3.3.4 Lower Bounds . . . 45

3.3.5 Spectral Sparsification of Directed Hypergraphs . . . 48
3.4 Warm-Up: Ordinary Graphs . . . 49
3.4.1 Proof of Lemma 3.4.2 . . . 54
3.5 γ-Balanced Weight Assignments . . . 56
3.5.1 Spanning Tree Potential . . . 56
3.5.2 Existence of γ-Balanced Weight Assignments . . . 58
3.5.3 Polynomial-Time Construction . . . 60
3.6 Hypergraph Sparsification . . . 62
3.7 Nearly Optimal Speed-Up . . . 69
3.7.1 Fast Algorithm for Constructing Polynomial-Sized Sparsifiers . . . 70
3.7.2 Even Faster Construction for γ-Balanced Weight Assignments . . . 72
3.7.3 Proof of the Second Part of Theorem 1 . . . 77
3.8 Lower Bounds . . . 78
3.8.1 String Compression . . . 79
3.8.2 Construction . . . 80
3.9 Spectral Sparsification of Directed Hypergraphs . . . 85
3.9.1 Preliminaries . . . 85
3.9.2 Nearly Equally Overlapping Directed Hypergraphs . . . 87
3.9.3 Proofs of Claims 3.9.9 to 3.9.12 . . . 90
3.9.4 Proof of Theorem 3.1.2 . . . 93

4 Massively Parallel Simulation of Random Walks **95**
4.1 Introduction . . . 95
4.2 MPC random walks . . . 98
4.2.1 Overview of our Algorithm . . . 98
4.3 From random walks to local clustering . . . 104
4.4 Empirical Evaluation . . . 105
4.4.1 Scalability . . . 106
4.4.2 Comparison . . . 107

II Maximum Matching **109**

5 Randomized Composable Coresets for Maximum Matching **111**
5.1 Introduction . . . 111
5.2 Randomized Composable Coresets . . . 113
5.3 Preliminaries and Notation . . . 113
5.4 Our coreset: the matching skeleton . . . 114
5.5 Main Result . . . 117
5.6 Limitations of the server flow coreset . . . 120

6 Highly Space-Efficient Maximum Matching **123**
6.1 Introduction . . . 123
6.1.1 Related Work . . . 126
6.2 Preliminaries . . . 127

6.3 Our Algorithm and Technical Overview . . . 128
6.3.1 Offline Algorithm . . . 128
6.3.2 IID Edge Stream Version of Algorithm 14 . . . 130
6.4 Sample Complexity of Algorithm 15 . . . 135
6.4.1 Sample Complexity of Level Tests . . . 135
6.4.2 Sample Complexity of IID-PEELING . . . 136
6.5 Correctness of Algorithm 15 (proof of Theorem 6.4.3) . . . 139
6.5.1 Definitions and Preliminaries . . . 139
6.5.2 Lower Bound: Constructing a Near-Optimal Matching from $\widehat{M}$ . . . 143
6.5.3 Upper bound: Constructing a Near-Optimal Vertex Cover . . . 145
6.6 LCA . . . 149
6.6.1 Overview of Our Approach . . . 150
6.6.2 Related Work . . . 151
6.6.3 Algorithms . . . 152
6.6.4 Query Complexity . . . 154
6.6.5 Approximation Guarantee . . . 156
6.6.6 Memory Complexity and Consistent Oracles . . . 157
6.6.7 Proof of Theorem 6.1.2 . . . 158
6.7 Lower Bound of $\widetilde{\Omega}(d^2)$ for Simulation of Randomized Greedy . . . 159
6.7.1 Lower Bound for Infinite Graphs . . . 160
6.7.2 Depth and Variance Analysis for Infinite Graphs . . . 165
6.7.3 The Lower Bound Instance (Truncated $H^{d,\epsilon}$) . . . 167
6.8 Lower-bound on the Number of Sampled Edges . . . 168
6.8.1 Overview . . . 168
6.8.2 Preliminaries . . . 171
6.8.3 Constructing Graphs with Identical k-Level Degrees . . . 172
6.8.4 Increasing Girth via Graph Lifting . . . 177
6.8.5 k-Edge Subgraph Statistics in G and H . . . 180
6.9 Analysis of the algorithm on a random permutation stream . . . 182
6.9.1 Introduction and Technical Overview . . . 182
6.9.2 Preliminaries . . . 183
6.9.3 Padding and total variation distance . . . 184
6.9.4 Bounding KL-divergence . . . 188
6.9.5 The full algorithm . . . 194

7 Conclusion **199**

A Supplementary Material for Chapter 2 **201**

B Supplementary Material for Chapter 3 **203**

C Supplementary Material for Chapter 4 **205**
C.1 Proof of Theorem 4.1.2 . . . 205
C.2 Preliminaries of Section 4.3 . . . 206

C.3 Omitted claims and proofs 207
C.4 Additional Experiments 214

D Supplementary Material for Chapter 6 **217**
D.1 Proof of Lemma 6.5.8 217
D.2 Oversampling Lemma 221
D.3 Proofs omitted from Section 6.7 223
D.4 Details omitted from Section 6.8 225
D.4.1 Proof of Theorem 6.8.2 225
D.4.2 Proof of Lemma 6.8.23 227
D.5 Proofs omitted from Section 6.9 227

Bibliography **248**

1 Introduction

One of the main challenges facing modern computer science is the astonishing growth of datasets. The amount of available data is growing exponentially, outpacing even Moore's law. We increasingly find that our old algorithms are no longer efficient enough to handle the ever-larger inputs. Classical algorithms implicitly rely on having random access to the input – that is, being able to read any part of it at any time during the execution. However, this is no longer a valid assumption when the size of the input outgrows the memory of any single computer. Instead we must turn to new algorithmic techniques to handle such cases: Various forms of data compression can be used to drastically reduce the size of the input, while preserving its essence – allowing subsequent classical algorithms to run more efficiently in both space and time complexity. Alternately, the input may be scanned in a data stream, which allows an algorithm to meaningfully process the input while using significantly (often exponentially) less space than it would take to store it in full. Finally, an input that cannot fit in the memory of a single machine, may be distributed across many, and processed in parallel with only limited communication between the machines.

Graphs are one of the most common forms of structured data. They occur in all fields of science and can model such varied concepts as online social networks, the structure of complex molecules, or the network of protein-protein interactions in living organisms. Graphs can help us find the shortest path to take through the road network of a large city, or track the spread of an epidemic in a population. It is unsurprising then that graphs are no exception to the the trend of ever-growing datasets, and the associated challenges: The internet has billions of websites (online advertisement allocations requires sophisticated analysis of this graph); the human brain contains billions of neurons with as many as 100 trillion connections between them (the field of network neuroscience seeks to understand the structural properties of this massive graph). In recent years, many space-efficient algorithmic techniques have found success in solving graph optimization problems, in both theory and practice.

Graph Compression

An example of data compression in the context of graphs is *sparsification.* This is the process of reducing a dense graph to a sparse, reweighted subgraph of itself (called a sparsifier) which preserves some crucial properties of the original graph. For instance, a *cut* sparsifier approximately preserves the size of all cuts. The more powerful *spectral* sparsifier preserves the quadratic form of the graph Laplacian matrix, which allows the recovery of cut sizes and more. A series of seminal results Karger (1994); Spielman and Teng

(2011); Batson et al. (2012) lead to the discovery that both cut and spectral sparsifiers of linear size exist for any input graph. This makes sparsification a powerful tool for compressing graph data: It can reduce the space required to store a massive graph from quadratic to linear in the number of vertices. A wide range of algorithms can be run on a sparsifier instead of the original graph, reducing both their time and space complexity. These are algorithms whose output depends only on the spectral structure of the input graph, for example a variety of clustering algorithms Andersen et al. (2006); Spielman and Teng (2013), as well as calculation of the personalized page rank vector.

This useful method has recently been generalized to the natural higher order analogues of graphs – hypergraphs. Hypergraphs are a powerful generalization of graphs, where edges can connect any number of vertices, which allows one to encode multi-way relationships. For instance, while a social network based on "friendships" can be aptly modeled as a graph, a communication network based on email exchanges is better modeled as a hypergraph, since emails often go out to multiple recipients. Since the size of a "dense" hypergraph can be as large as exponential in the number of vertices (in contrast to ordinary graphs), data compression in the form of sparsification is especially pertinent in this setting.

Parallel graph processing

One of the most successful methods for the space-efficient processing of large datasets is the so-called *massively parallel computation* (or MPC) model – this formalizes such real-world frameworks as Hadoop White (2012), Spark Zaharia et al. (2010), or Dryad Isard et al. (2007). Here, one has access to the input distributed across a large number of machines, allowing the processing of datasets far larger than the memory of each individual machine. Algorithms working in this setting must minimize the amount of communication required between machines, as this is typically the bottleneck in practical applications. This technique has seen much success in solving large scale graph analysis problems, both in theory and practice. One example of a data compression technique for graphs that is specifically designed for the massively parallel setting is the use of composable coresets (for example, see Bateni et al. (2014); Indyk et al. (2014); Mirrokni and Zadimoghaddam (2015)). Here, each machine calculates in parallel a small summary of the subgraph located on it – this is called the coreset. These summaries are then combined on a single machine, which computes an approximate solution to the original problem. Composable coresets are a highly versatile tool, and sufficiently small coresets imply space-efficient algorithms not just in MPC, but also in the streaming setting.

Streaming graph processing

Sparsifiers and composable coresets must store all vertices of their graphs and thus require at least linear space in the number of vertices. MPC algorithms for graph analysis usually use less space than this per machine; however, they almost always require space-per-machines that is at least a small polynomial of the input size. The most drastically space-efficient algorithms come from the *streaming* model, where it often suffices to use exponentially less space than it would take to store the input in its entirety. In the streaming setting, algorithms have access to their inputs through a sequence of bits – called the data stream – which can be parsed in order, but which cannot be stored in the limited memory available. Streaming was first considered in Munro and Paterson (1980), and such polylogarithmic-space algorithms have since been found for many fundamental problems (for instance Alon et al. (1999); Charikar et al. (2004)). When the input is a graph, a streaming algorithm may read the edges of the graph sequentially (insertion-only streams), or the algorithm may receive a sequence of edge insertions and deletions, which result in the underlying graph (dynamic streams). For many fundamental graph problems – for instance matching, clustering, vertex cover – the required space to solve the problem in the streaming setting is lower bounded by the number of vertices simply due to the large size of the output. However, when this is not an issue, it is often possible to process the input graph in polylogarithmic space – as in the case of approximate maximum matching *size* Kapralov et al. (2014); Paz and Schwartzman (2017).

1.1 Overview of Our Contributions

In this thesis, we explore space-efficient representations of graphs and hypergraphs (such as sparsifiers and composable coresets), as well as space-efficient solutions for various problems in algorithmic graph theory (in the streaming and MPC settings). In the following section we outline our contributions to the field.

1.1.1 Spectral Graph Theory

Spectral graph theory is the study of graphs through the linear algebraic properties of the adjacency matrix, and the closely related Laplacian matrix, L. From fundamental graph theoretical concepts such as bipartiteness or connectedness, to highly applicable algorithmic problems such as clustering or the calculation of the personalized page rank vector, much of graph theory can be better understood through this lens. It is also true that many relevant problems are robust to spectral approximations – that is, they give similar results on graphs with similar Laplacian matrices. This motivates the following definition of spectral sparsification.

Definition 1.1.1. *Given a graph $G = (V, E)$ a reweighted subgraph $\widetilde{G} = (V, \widetilde{E}, w)$ is an ϵ-spectral sparsifier, if for all $x \in \mathbb{R}^V$*

$$(1-\epsilon)\cdot x^\top L_G x \le x^\top L_{\widetilde{G}} x \le (1+\epsilon)\cdot x^\top L_G x.$$

Spectral sparsification was introduced by Spielman and Teng (2011) as a stronger variant of the earlier cut sparsification of Karger (1994). Since then, there has been much research on the topic, producing many different approaches to constructing spectral sparsifiers in diverse settings Spielman and Srivastava (2011); Kapralov et al. (2014); Zhu et al. (2015); Lee and Sun (2015, 2017). In particular, Batson et al. (2012) showed the existence of remarkably small ϵ-spectral sparsifiers, consisting of only $4n/\epsilon^2$ edges, for any input graph.

One of the most commonly used techniques for constructing spectral sparsifiers (and the one most relevant to this thesis) is *importance sampling* using effective resistance. Importance sampling is a deceptively simple algorithm that proceeds as follows: Assign some importance to each edge of the input graph. Then sample each edge independently with probability proportional to its importance. Finally, scale up the weight of those sampled, inverse proportionally to their importance, such that each original edge ends up in the sparsifier with an expected weight of 1. Spielman and Srivastava (2011) showed that sampling in such a way, proportionally to *effective resistance* results in a nearly optimal ϵ-spectral sparsifier of size $O(n\log n/\epsilon^2)$. Effective resistance of a pair of vertices can be understood by imaging the graph as a circuit where each edge represents a unit resistor.

Spectral Sparsification via Sketching

Sketching is the powerful technique of compressing data via a random linear transformation. In the context of graphs, this means summarizing the m by n edge-vertex incidence matrix B, as ΠB using some random d by m sketching matrix Π. (Typically, d is $\log^{O(1)} n$, making the total size of the sketch nearly linear.) The linearity of the transformation makes this a versatile tool in both dynamic streams and the massively parallel regime. Dynamic streams include both edge insertions and deletions, and are often used to model rapidly changing massive graphs, such as online social networks. In this setting, sketching handles edge deletions seamlessly, since when an edge e is deleted, $\Pi \cdot e$ can simply be subtracted from the current sketch. Sketching also readily admits to parallelization, since the sketches of arbitrary subgraphs add up to the sketch of the whole graph. Graph sketching was first introduced by Ahn et al. (2012a) and efficient sketches have since been found for a great number of fundamental graph theoretical problems Ahn et al. (2012b); Kapralov et al. (2017a); Ahn et al. (2012b); Kapralov and Woodruff (2014); Assadi et al. (2016); Andoni et al. (2016).

Since sparsification is intended for massive graphs which are difficult or infeasible to store by classical means, it makes sense to study the construction of sparsifiers in the streaming and massively parallel settings. Correspondingly, there have been multiple results in the past decade on producing graph sketches, from which a spectral sparsifier can be constructed Ahn et al. (2013); Kapralov et al. (2017a, 2019a), with various trade-offs between sketch size and runtime. In particular, Kapralov et al. (2017a) achieves the nearly optimal $\widetilde{O}(n)$ sketch size, using an adaptation of the spectral sparsifier of Spielman and Srivastava (2011) via importance sampling with effective resistances. However, they require quadratic time to decode the sparsifier from the sketch. They show that the core difficulty of the problem is finding all edges that have "high" effective resistance in the input graph (this is clearly necessary, but due to the work of Kapralov et al. (2017a) it also becomes essentially sufficient). To do this, Kapralov et al. (2017a) uses the observation that a vertex-pair (u, v) is an edge of high effective resistance exactly if it contributes a large fraction of the electrical flow induced from u to v. The highest contributors to the flow can be decoded using the seminal result of Charikar et al. (2004) for ℓ_2 heavy hitter sketching; however, testing each pair (u, v) for this property results in the quadratic runtime of the algorithm.

In this thesis, we propose a nearly linear time decoding algorithm for the sketching of spectral sparsifiers, achieving near-optimality in every metric of the problem: sparsifier size, sketch size, and decoding time. Through a more involved examination of the properties of the effective resistance metric, we are able to rely on certain high effective resistance edges showing up in the electrical flows induced on *other*, nearby vertex-pairs. By observing the electrical flows of a carefully chosen, $\widetilde{O}(n)$ vertex-pairs (as opposed to the total $\Omega(n^2)$), we are able to guarantee that we find every high effective resistance edge, thus achieving a nearly linear runtime overall.

Spectral Sparsification of Hypergraphs

Hypergraphs are increasingly used to represent complex structured data in various fields of science. The study of motif graphs – hypergraphs representing higher order structures such as triangles in some underlying network – is a widespread technique for the analysis of social Wasserman and Faust (1994), physical Benson et al. (2016), and biological networks Wong et al. (2012). Furthermore, hypergraphs have also found applications in problems related to computer vision Huang et al. (2009) and information retrieval Gibson et al. (2000). Consequently, there has been a recent line of work in applying the technique of spectral sparsification to hypergraphs. In order to generalize the definition of spectral sparsification Definition 1.1.1, one only needs to propose a hypergraph analogue to the quadratic form of the graph Laplacian. The definition below was introduced by Louis (2015) and Yoshida (2019), and has since become standard in the field.

Definition 1.1.2. *For a weighted hypergraph $H = (V, E, w)$ the* energy *function $Q_H : \mathbb{R}^V \to \mathbb{R}_+$ associated to H is defined as follows.*

$$Q_H(x) = \sum_{e \in E} w_e \cdot \max_{u,v \in e} (x_u - x_v)^2.$$

The energy function of a hypergraph is a highly natural and useful generalization of the ordinary graph Laplacian. It encodes the cut structure of H, since the energy of an indicator vector is equal to the size of the corresponding cut. It also has applications related to semi-supervised learning Zhang et al. (2020) and link prediction Yadati et al. (2020).

The definition of spectral sparsification Definition 1.1.1 then generalizes naturally: a reweighted sub-hypergraph $\widetilde{H}$ is an ϵ-spectral sparsifier of H if its energy function approximates that of H to within a multiplicative $1 \pm \epsilon$, on all inputs simultaneously. Soma and Yoshida (2019), who introduced this concept also proved the existence of ϵ-spectral hypergraph sparsifiers of size $\widetilde{O}(n^3/\epsilon^2)$; a subsequent work Bansal et al. (2019) achieves $\widetilde{O}(nr^3/\epsilon^2)$, where r is the maximum size of a hyperedge – the rank of the hypergraph.

For the first time, we show the existence of a nearly linear $\widetilde{O}(n/\epsilon^4)$ sized spectral sparsifier for all hypergraphs. We rely on two main technical tools: the balanced weight assignment of hypergraphs (introduced in Chen et al. (2020)), as well as a novel proof of correctness for the classical result of Spielman and Srivastava (2011). Both previous results (Soma and Yoshida (2019) and Bansal et al. (2019)) relied on a variant of importance sampling, where the hyperedge-importances were derived from the uniform clique expansion of the hypergraph (that is, the ordinary graph resulting from replacing each hyperedge with a uniformly weighted clique). While the clique expansion is a useful tool in handling hypergraphs, it doesn't quite capture the spectral structure, and any importance sampling defined in terms of it will inevitably be suboptimal. We instead use the balanced weight assignment of H. This is also constructed by replacing each hyperedge with a clique; however, the weights of the edges are carefully chosen such that the effective resistances of all edges in the same clique are roughly the same. We then use this quantity (the effective resistance of the edges of its clique) as the measure of importance for each hyperedge.

When proving correctness of spectral sparsification via importance sampling, one has to essentially prove the concentration of a random matrix (the Laplacian of the sparsifier) around its expectation. For this purpose, previous works concerning ordinary graphs have always relied on linear algebraic tools such as matrix Bernstein Tropp (2011). Unfortunately, this doesn't translate to the realm of hypergraphs, due to the non-linear nature of the hypergraph energy function. In this thesis, we reprove the original theorem of Spielman and Srivastava (2011) for ordinary graphs but only using simple tools: union bound and Chernoff bound – providing the first ever such "elementary" proof of spectral sparsification by effective resistance sampling. This proof then translates surprising easily to the hypergraph case, with the aid of balanced weight assignments. We use a similar proof-technique to construct the first ever non-trivial spectral sparsifiers for directed hypergraphs as well.

We complement our positive result with a corresponding lower bound, thus proving the near-optimality of our sparsifiers. Under the strict definition of Definition 1.1.1 it's not difficult to see that n hyperedges are required in some cases – indeed n hyperedges may be required simply to retain connectivity. However, n hyperedges in a hypergraph of rank r still represent nr bits of information, which leads us to the following question: Is there any way of compressing the approximate spectral structure of an r-rank hypergraph to $o(nr)$ bits? We answer this question in the negative, providing a construction based on Ruzsa-Szemerédi graphs that encodes uncompressible information into the spectral structure of hypergraphs. This shows that our sparsifiers are nearly optimal not only among sparsifiers, but represent the most efficient possible way of compressing the approximate hypergraph energy function. It also shows that this energy function is a significantly more general object than the ordinary graph Laplacian, which can be approximately encoded into $O(n)$ bits.

Massively Parallel Simulation of Random Walks

Random walks in graphs are exceedingly widely used in real-world application, for example, in the personalized page rank algorithm underlying the original implementation of the Google search engine Brin and Page (1998). Since such an algorithm would have to be executed over the graph representing the internet, it is unsurprising that a great amount of research goes into various sublinear space implementations of random walks. In one of the more practical results of the thesis, we present an MPC algorithm for simultaneously simulating a large number of independent random walks, as well as an empirical evaluation of its performance.

In the massively parallel computation model, the input (in our case the edges of a graph) is initially distributed randomly across a large number of machines, each possessing space $s \ll m$ – insufficient to store all but a small fraction of the edges of the input graph. The processing of the input graph then proceeds in synchronous rounds of computation followed by communication, where the machines exchange data through messages. The amount of computation performed is unlimited; neither is the size of the messages limited, except by the space s – machines cannot send or receive a greater volume of messages than would fit in their memory. Instead the aim is to minimize the space-per-machine, and the number of rounds of communication needed to solve the problem at hand. Ideally – in the case of so-called "fully scalable" MPC algorithms – the algorithm works as long as the space per machine is $s = n^{\epsilon}$ for an arbitrarily small constant ϵ.

Our task is to simulate a large number (B) of independent random walks of length ℓ in the input graph, from various starting vertices. A naive implementation of this could build each random walk step-by-step, using a round of communication between each step to aggregate the neighborhood of the current position of each walk onto a corresponding machine. Such simulations can be performed arbitrarily in parallel, and so a large number of independent random walks could be simulated in ℓ rounds of communication. However, one can do better: Łącki et al. (2020) succeeded in implementing a stitching algorithm for the parallel simulation of random walks in the MPC setting, in only $O(\log \ell)$ rounds. The idea of walk stitching is the following: instead of starting only the prerequisite B random walks, we initially start many more walks form every vertex of the graph. Then, round by round, we combine walks to double their length. Two length-1 walks combine to make a length-2 walk, two length-2 walks combine to make a length-4 walk, and so on, until in the final round of computation we pairwise combine length-$\ell/2$ walks to recover the desired length-ℓ walks. The weakness of this approach, however, is that in order to simulate a particular collection of random walks, the resulting distribution must be known ahead of time (at least approximately). Initializing a stitching algorithm without being aware of where the walks will end up can result in a miss-match between the number of walks ending in a particular region of the graph, and the number of walks starting there. In this case, some of the fractional walks cannot be stitched to anything and have to be discarded, resulting in an insufficient number of walks or a skewed distribution. For this reason, Łącki et al. (2020) restrict themselves to simulating random walks that start initially in the stationary distribution; in this case, all subsequent distributions will also be stationary, and can therefore be known ahead of time.

Our variant of the MPC stitching algorithm, presented in this thesis, sidesteps this issue, and can simulate random walks from an arbitrary starting distribution, with similar guarantees on space and communication complexity. The core idea of our algorithm is very simple: We perform several consecutive cycles of the above stitching algorithm. The first cycle uses the stationary as its starting distribution, while each cycle thereafter gradually changes the distribution, until we reach the desired one. Each distribution is similar enough to the previous one, that the previous stitching algorithm's result can be used to initialize the next stitching algorithm. With this technique, we achieve $O(\log\ell\cdot\log_\lambda B)$ round complexity with approximately $O((m+B)\lambda)$ total space, with λ representing the trade-off between the two quantities. When choosing λ to be a small polynomial of B, the round complexity matches that of Łącki et al. (2020), which is known to be optimal conditionally on the two-cycle conjecture Beame et al. (2013).

We verify the algorithm's efficacy through empirical evaluation, and confirm that it is significantly more space-efficient in simulating random walks from a non-stationary starting distribution than a naive generalization of Łącki et al. (2020). As a corollary to our main result, we also show that one can simulate the personalized page rank vector via the same method. Furthermore, we can use our approximate personalized page rank vector to perform local clustering in the MPC model, following the renowned Nibble algorithm of Andersen et al. (2006).

1.1.2 Maximum Matching

Maximum matching is the problem of selecting the largest possible set of edges in a graph that share no vertices. It is a problem of great theoretical significance, with its study going back nearly a hundred years Hall (1935), as well as numerous practical applications, for instance in online advertisement allocation. Consequently, is has been studied in great depth in every imaginable low-space model of computation (for instance Luby (1986); McGregor and Vorotnikova (2018); Assadi et al. (2019a) among many others), including streaming, and the MPC model.

Randomized Composable Coresets for Maximum Matching

Maximum matching is among those graph optimization problems which are not robust to spectral approximation, and so solving maximum matching is not aided by spectral sparsification. Instead, another method of graph compression – randomized composable coresets – has found some success in this area. While Konrad (2015) shows the limitations of deterministic composable coresets for the purposes of approximate maximum matching, these limitations don't extend to the version where the initial partitioning of the graph is done randomly:

Definition 1.1.3. *Let $\mathscr{A}$ be some algorithm compressing any graph H to its subgraph $\mathscr{A}(H)$. $\mathscr{A}$ is said to produce an α-**approximate randomized composable coreset** (or RCCS) for some maximization problem* P *if for any graph G, and its uniformly random k-partition $G_1,\ldots,G_k$ the following holds:*

$$\alpha\cdot \mathrm{P}[G]\leq \mathbb{E}\Big(\mathrm{P}[\mathscr{A}(G_1)\cup\ldots\cup\mathscr{A}(G_k)]\Big).$$

That is, we *randomly* partition the edges of the graph into k parts, summarize each part via our compression algorithm $\mathscr{A}$, and recombine everything to produced a smaller graph, that still contains an approximately optimal solution (in our case to maximum matching). The aim in constructing randomized composable coresets is to minimize the size (the upper bound on the number of edges in $\mathscr{A}(H)$), while also maximizing the approximation quality α.

The first RCCS for maximum matching, proposed by Assadi and Khanna (2017), was maximum matching itself. That is, one calculates an arbitrary maximum matching in each part, then combines them and recalculates a maximum matching in the union. This is naturally a very small RCCS; it was initially shown to have $\alpha = 1/9$ approximation quality, later improved to $1/3$ by Assadi et al. (2019a), and shown to be no greater than $1/2$. A more powerful coreset was also proposed by Assadi et al. (2019a), the so-called edge-degree constrained subgraph or EDCS, which boasted an $\alpha = 2/3$ approximation ratio, but at a cost of a larger size of $n\log^{O(1)} n$ edges.

As our contribution to the area, we propose a new RCCS for maximum matching – the *matching skeleton* (from Goel et al. (2012) and Bernstein et al. (2018)) – with a provable $1/2$ approximation ratio, and a maximum size of $n-1$. The matching skeleton is essentially the support of a sparse fractional matching which is in some sense optimally spread out through the graph. Its small size is guaranteed by the fact that it is always cycle-free. The improved approximation quality compared to the maximum matching RCCS can be attributed to the fact that in generating the matching skeleton we carefully choose some canonical fraction matching that reflects the structure of the original graph. By contrast, the maximum matching RCCS is simply an arbitrary maximum matching.

Highly Space-Efficient Maximum Matching

A prevalent technique for computing approximate maximum matchings in truly sublinear space is local exploration of the input graph. Given some algorithm for producing an approximate maximum matching one can simulate it locally. That is, instead of running the whole algorithm, we can single out a vertex or edge and ask: What would have happened to this vertex/edge in the course of the algorithm if we had run it? This question can often be answered by looking at only a small region of the graph, allowing for a space-efficient implementation. Yoshida et al. (2009) and Onak et al. (2012) simulated the randomized greedy algorithm locally, to estimate the matching size of a graph through query access; Kapralov et al. (2014) used local simulation of a peeling algorithm for maximum matching to achieve a similar result in the random order streaming setting; Levi et al. (2017) used a local simulation of Luby's algorithm (Luby (1986)) to implicitly construct an approximate maximum matching using a Local Computation Algorithm.

In this thesis, we propose a highly optimized local simulation of a peeling algorithm. Here we give intuition on what such a peeling algorithm does, and which modifications are needed to make its local simulation more efficient.

It is known that graphs that are close to regular also have large ($\Omega(n)$ sized) matchings. The difficulty in finding large matchings comes when the range of degrees in the graph is very broad; in this case it is easy to match high degree vertices, but difficult to match low degree ones (think, for instance, of the start graph). A peeling algorithm works by matching all of the highest degree vertices (say, the ones within a factor 2 of the maximum degree), and removing them, as well as their matched pairs. The algorithm then repeats this step in the resulting graph, which now has a factor 2 lower maximum degree. After $O(\log n)$ such steps, no more edges remain, and we have a constant-approximate maximum matching. To simulate this algorithm locally we have to ask: What would happen to vertex v throughout this process? If v has high degree for instance, we can say that it would be removed in the first round. If not, however, what happens to v in the second round depends on what happened to its neighbors in the first round. This creates a recursive chain of dependencies and results in an $O(\log n)$-depth exploration tree. The $O(\log n)$-radius neighborhood of v can potentially be as large as the whole graph, so we make two modifications to the algorithm in order to prune this exploration tree to a manageable size: First, we replace degree-counting with crude empirical estimates of the degree based on subsampling; second we cut the local simulation short if it takes more time than expected. This second modification makes proving the efficiency of the algorithm rather straight-forward. Correctness, however, is far less obvious since each such "cutting short" of the algorithm is prone to producing errors which propagate up the chain of recursive calls. Nevertheless, through careful analysis, we manage to bound the total effect of these errors, and show that even this heavily modified version of peeling produces a constant-factor approximate maximum matching in the end.

As one of the implementations of our technique, we design a state-of-the art Local Computation Algorithm (LCA) for implicitly constructing a constant-approximate maximum matching. The LCA model is specifically designed for those problems where the output itself is larger than the available memory: Instead of outputting the solution explicitly, the algorithm must be able to provide *consistent* query access to it through local exploration of the graph. In our case, we provide query access to a constant-approximate maximum matching using $d\log^3 n$ total space, and $d\log n$ time per query *in the worse case*, where d is the maximum degree of the graph. This improves upon the previous best known LCA of Levi et al. (2017) which required $\Omega(d^4 \log n)$ time.

Our technique is also applicable in the setting where we have access to the graph through a stream of uniformly random edge samples. Here, we cannot output a matching in truly sublinear space, due to the large possible size of the matching; however, we can determine the maximum matching *size* of the graph to within a constant factor in only $O(\log^2 n)$ space, while processing fewer than m edge samples.

Furthermore, we show that significantly fewer ($m^{1-\Omega(1)}$) edge samples do not suffice even in the absence of a space constraint. That is, one cannot deduce the size of the maximum matching of a graph from only looking at the edges subsampled at a polynomial rate. Our lower bound construction is a pair of graphs with greatly different matching sizes; both graphs are sparse, and we exploit the fact that once subsampled, only small constant-sized connected components of each graph remain, which cannot be distinguished from each other.

Finally, we extend our above positive result to the related setting of "random permutation streaming", where we have access to the input graph through a randomly ordered stream in which each edge appears exactly once. We are able to prove that the same algorithm achieves an $O(\log^2 n)$-approximation to the matching size – slightly worse than when using uniform edge samples, due to the technical difficulties in analyzing the dependencies inherent in a random permutation stream. Both results improve over the work of Kapralov et al. (2014), which achieved only a large $\log^{O(1)} n$ approximation ratio in both settings.

Spectral Graph Theory Part I

2 Spectral Sparsification via Sketching

This chapter is based on joint work with Michael Kapralov, Navid Nouri, and Aaron Sidford Kapralov et al. (2019b). A merged version with Kapralov et al. (2020b) has been accepted to the 31st ACM-SIAM Symposium on Discrete Algorithms (**SODA**) 2020 under the title

Dynamic Streaming Spectral Sparsification in Nearly Linear Time and Space.

The result also appears in Navid Nouri's thesis.

2.1 Introduction

Graph sketching, i.e. constructing small space summaries for graphs using linear measurements, has received much attention since the work of Ahn, Guha and McGregor Ahn et al. (2012a) gave a linear sketching primitive for graph connectivity with optimal $O(n \log^3 n)$ space complexity Nelson and Yu (2019). A key application of linear sketching has been to design small space algorithms for processing *dynamic graph streams*, where edges can be both inserted and deleted, although the graph sketching paradigm has been shown very powerful in many other areas such as distributed algorithms and dynamic algorithms (we refer the reader to the survey McGregor (2017) for more on applications of graph sketching). Furthermore, it is known that linear sketching is essentially a universal approach to designing dynamic streaming algorithms Li et al. (2014), and yields distributed protocols for graph processing with low communication. Sketching solutions have been recently constructed for many graph problems, including spanning forest computation Ahn et al. (2012a), cut and spectral sparsifiers Ahn et al. (2012b); Kapralov et al. (2017a), spanner construction Ahn et al. (2012b); Kapralov and Woodruff (2014), matching and matching size approximation Assadi et al. (2016, 2017), sketching the Laplacian Andoni et al. (2016); Jambulapati and Sidford (2018) and many other problems. The focus of our work is on *oblivious* sketches for approximating spectral structure of graphs with optimally fast recovery. A sketch is called *oblivious* if its distribution is independent of the input – such sketches yield efficient *single pass* dynamic streaming algorithms for sparsification. We now outline the main ideas involved in previous works on this and related problems, and highlight the main challenges in designing a solution that achieves both linear space and time.

Oblivious linear sketches with nearly optimal $n \log^{O(1)} n$ have been obtained for the related problems of constructing a spanning forest of the input graph Ahn et al. (2012a), the problem of constructing *cut sparsifiers* of graphs Ahn et al. (2012b) and for the spectral sparsification problem itself Kapralov et al. (2017a). In the former two cases the core of the problem is to design a sketch that allows recovery of edges that cross *small cuts* in the input graph, and the problem is resolved by applying ℓ_0-sampling(see, e.g., Jowhari et al. (2011); Cormode and Firmani (2014); Kapralov et al. (2017b)), and more generally exact (i.e., ℓ_0) sparse recovery techniques on the edge incidence matrix $B \in \mathscr{R}^{\binom{n}{2} \times n}$ of the input graph: one designs a sketching matrix $S \in \mathscr{R}^{\log^{O(1)} n \times \binom{n}{2}}$ and maintains $S \cdot B \in \mathscr{R}^{\log^{O(1)} n \times n}$ throughout the stream. A natural recovery primitive that follows Boruvka's algorithm for the MST problem then yields a nearly linear time recovery scheme. Specifically, to recover a spanning tree one repeatedly samples outgoing edges out of every vertex of the graph and contracts resulting connected components into supernodes, halving the number of connected components in every round. Surprisingly, a sketch of the original graph suffices for sampling edges that go across connected components in graphs that arise through the contraction process, yielding a spanning forest in $O(\log n)$ rounds and using $n \log^{O(1)} n$ bits of space.

The situation with spectral sparsifiers is very different: edges critical to obtaining a spectral approximation do not necessarily cross small cuts in the graph. Instead, 'important edges' are those that have large effective resistance, i.e can be made 'heavy' in the ℓ_2 sense in an appropriate linear combination of the columns of the edge incidence matrix B. This observation was used in Kapralov et al. (2017a) to design a sketch with nearly optimal $n \log^{O(1)} n$ space complexity, but the recovery of the sparsifier was brute-force and ran in $\Omega(n^2)$ time: one had to iterate over all potential edges and test whether they are in the graph and have 'high' effective resistance. Approaches based on relating effective resistances to inverse connectivity have been proposed Ahn et al. (2013), but these result in suboptimal $\Omega(n^{5/3})$ space complexity. In a very recent work Kapralov et al. (2019a) a subset of the authors proposed an algorithm with $n^{1.4+o(1)}$ space and runtime complexity, but no approach that yields optimal space and runtime was known previously.

A key reason why previously known sketching techniques for reconstructing spectral approximations to graphs failed to achieve nearly linear runtime is exactly the lack of simple 'local' (akin to Boruvka's algorithm) technique for recovering heavy edges. The main contribution of this chapter is such a technique: we propose a bucketing technique based on ball carving in (an approximation to) the effective resistance metric that recovers appropriately heavy effective resistance edges by routing flows between source-sink pairs that belong to the same bucket. This ensures that the recovery process is more 'localized', and results in a nearly linear time algorithm.

Our result. Formally, we consider the problem of constructing *spectral sparsifiers* Spielman and Teng (2011); Spielman and Srivastava (2011) of graphs presented as a dynamic stream of edges: given a graph $G = (V, E)$ presented as a dynamic stream of edge insertions and deletions and a precision parameter $\epsilon \in (0, 1)$, our algorithm outputs a graph G' such that

$$(1-\epsilon)L \preceq L' \preceq (1+\epsilon)L,$$

where L is the Laplacian of G, L' is the Laplacian of G' and $\prec$ stands for the positive semidefinite ordering of matrices.

Our main result is a linear sketching algorithm that compresses a graph with n vertices to a $n\log^{O(1)} n$-bit representation that allows $\log^{O(1)} n$-time updates, and from which a spectral approximation can be recovered in $n\log^{O(1)} n$ time. Thus, our result achieve both optimal space and time complexity simultaneously.

Theorem 2.1.1 (Near Optimal Streaming Spectral Sparsification)**.** *There exists an algorithm such that for any $\epsilon \in (0,1)$, processes a list of edge insertions and deletions for an unweighted graph G in a single pass and maintains a set of linear sketches of this input in $O(\epsilon^{-2} n\log^{O(1)} n)$ space. From these sketches, it recovers in $O(\epsilon^{-2} n\log^{O(1)} n)$ time, with high probability, a weighted subgraph H with $O(\epsilon^{-2} n\log n)$ edges, such that H is a $(1\pm\epsilon)$-spectral sparsifier of G.*

Our result in Theorem 2.1.1 can be thought of as the first efficient 'ℓ_2-graph sketching' result, using an analogy to compressed sensing recovery guarantees. It is interesting to note that compressed sensing primitives that allow recovery in time nearly linear in sketch size (which is exactly what our algorithm achieves for the sparsification problem) usually operate by hashing the input vector into buckets so as to isolate dominant entries, which can then be recovered efficiently. The main contribution of our work is giving a 'bucketing scheme' for graphs that allows for nearly linear time recovery. As we show, the right 'bucketing scheme' for the spectral sparsification problem is a space partitioning scheme in the effective resistance metric.

Effective resistance, spectral sparsification, and random spanning trees. The *effective resistance metric* or *effective resistance distances* induced by an undirected graph plays a central role in spectral graph theory and has been at the heart of numerous algorithmic breakthroughs over the past decade. They are central to the to obtaining fast algorithms for constructing spectral sparsifiers Spielman and Srivastava (2011); Koutis et al. (2016), spectral vertex sparsifiers Kyng et al. (2016), sparsifiers of the random walk Laplacian Cheng et al. (2015); Jindal et al. (2017), and subspace sparsifiers Li and Schild (2018). They have played a key role in many advances in solving Laplacian systems Spielman and Teng (2004); Koutis et al. (2010, 2011); Peng and Spielman (2014); Cohen et al. (2014); Koutis et al. (2016); Kyng et al. (2016); Kyng and Sachdeva (2016) and are critical to the current fastest (weakly)-polynomial time algorithms for maximum flow and minimum cost flow in certain parameter regimes Lee and Sidford (2014). Given their utility, the computation of effective resistances has itself become an area of active research Jambulapati and Sidford (2018); Chu et al. (2018).

In a line of work particularly relevant to this chapter, the effective resistance metric has played an important role in obtaining faster algorithms for generating random spanning trees Kelner and Madry (2009); Madry et al. (2015); Schild (2018). The result of Madry et al. (2015) partitions the graph into clusters with bounded diameter in the effective resistance metric in order to speed up simulation of a random walk, whereas Schild (2018) proposed a more advanced version of this approach to achieve a nearly linear time simulation. While these results seem superficially related to ours, there does not seem to be any way of using spanning tree generation techniques for our purpose. The main reason is that the objective in spanning tree generation results is quite different from ours: there one would like to find a partition of the graph that in a sense minimizes the number times a random walk crosses cluster boundaries, which does not correspond to a way of recovering 'heavy' effective resistance edges in the graph. In particular, while in spanning tree generation algorithms the important parameter is the number of edges crossing the cuts generated by the partitioning, whereas it is easily seen that heavy effective resistance edges cannot be recovered from small cuts. Finally, the problem of partitioning graphs into low effective resistance diameter clusters has been studied recently in Alev et al. (2017). The focus of the latter work is on partitioning into *induced* expanders, and the results of Alev et al. (2017) were an important tool in the work of Kapralov et al. (2019a) that achieved the previous best $n^{1.4+o(1)}$ space and runtime complexity for our problem. Our techniques in this chapter take a different route and achieve optimal results.

Prior work. Streaming algorithms are well-studied with too many results to list and we refer the reader to McGregor (2014, 2017) for a survey of streaming algorithms. The idea of linear graph sketching was introduced in a seminal paper of Ahn, Guha, and McGregror Ahn et al. (2012a), where a $O(\log n)$-pass sparsification algorithm for dynamic streams was presented (this result is for the weaker notion of cut sparsification due to Karger (1994); Benczúr and Karger (1996)). A single-pass algorithm for cut sparsification with nearly optimal $\widetilde{O}(\epsilon^{-2}n)$ space was given in Ahn et al. (2012b), and extensions of the sketching approach of Ahn et al. (2012a) to hypergraphs were presented in Guha et al. (2015). The more challenging problem of computing a spectral sparsifier from a linear sketch was addressed in Ahn et al. (2013), who gives an $\tilde{O}(\epsilon^{-2}n^{5/3})$ space solution. An $\tilde{O}(\epsilon^{-2}n)$ space solution was obtained in Kapralov et al. (2017a) by more explicitly exploiting the connection between graph sketching and vector sparse recovery, at the expense of $\widetilde{O}(\epsilon^{-2}n^2)$ runtime. In a recent work Kapralov et al. (2019a) the authors gave a single pass algorithm with $\epsilon^{-2}n^{1.4+o(1)}$ space and runtime complexity.

We also mention that spectral sparsifiers have been studied in the insertion-only streaming model, where edges can only be added to G Kelner and Levin (2013); Cohen et al. (2016); Kyng et al. (2017), and in a dynamic data structure model Abraham et al. (2016); Andoni et al. (2016); Jambulapati and Sidford (2018), where more space is allowed, but the algorithm must quickly output a sparsifier at every step of the stream. While these models are superficially similar to the dynamic streaming model, they seem to allow for different techniques, and in particular do not require linear sketching since they do not constrain the space used by the algorithm. The spectral sparsification problem on its own has received a lot of attention in the literature (e.g., Spielman and Srivastava (2011); Spielman and Teng (2011); Batson et al. (2012); Zhu et al. (2015); Lee and Sun (2015, 2017). We refer the reader to the survey Batson et al. (2013) for a more complete set of references.

2.2 Preliminaries

General Notation. Let $G=(V,E)$ be an unweighted undirected graph with n vertices and m edges. For any vertex $v\in V$, let $\chi_v \in \mathscr{R}^n$ be the indicator vector of v, with a one at position v and zeros elsewhere. Let $B_n \in \mathscr{R}^{\binom{n}{2}\times n}$ denote the vertex edge incidence matrix of an unweighted and undirected complete graph, where for any edge $e=(u,v)\in\binom{V}{2}$, $u\neq v$, its e'th row is equal to $\mathbf{b}_e := \mathbf{b}_{uv} := \chi_u - \chi_v$. Let $B\in\mathscr{R}^{\binom{n}{2}\times n}$ denote the vertex edge incidence matrix of $G=(V,E)$. B is obtained by zeroing out any rows of B_n corresponding to $(u,v)\notin E$.[1]

For weighted graph $G=(V,E,w)$, where $w: E\to\mathscr{R}_+$ denotes the edge weights, let $W\in\mathscr{R}_+^{\binom{n}{2}\times\binom{n}{2}}$

[1]Note this is different then the possibly more standard definition of B as the $E\times V$ matrix with the rows not in the graph removed altogether.

be the diagonal matrix of weights where $W(e,e) = w(e)$ for $e \in E$ and $W(e,e) = 0$ otherwise. Note that $L = B^{\mathrm{T}} W B = B_n^T W B_n$, is the Laplacian matrix of G. Let L^+ denote the Moore-Penrose pseudoinverse of L. Also, for a real valued variable s, we define $s^+ := \max\{0, s\}$. We also use the following folklore:

Fact 2.2.1. *For any Laplacian matrix L of an unweighted and undirected graph, its minimum nonzero eigenvalue is bounded from below by $\lambda_\ell = \frac{1}{8n^2}$ and its maximum eigenvalue is bounded from above by $\lambda_u = 2n$.*

Definition 2.2.2. *For any unweighted graph $G = (V,E)$ and any $\gamma \geq 0$, we define L_{G^γ}, as follows:*

$$L_{G^\gamma} = L_G + \gamma I.$$

This can be seen in the following way. One can think of G^γ as graph G plus some regularization term. In order to distinguish between edges of G and regularization term in G^γ, we let $B_{G^\gamma} = B \oplus \sqrt{\gamma} I$, where $B \oplus \sqrt{\gamma} I$ is the operation of appending rows of $\sqrt{\gamma} I$ to matrix B. One should note that $B_{G^\gamma}^{\mathrm{T}} B_{G^\gamma} = L_{G^\gamma}$. Also for simplicity we define L_ℓ for any integer $\ell \in [0, d+1]$ as follows:

$$L_\ell = \begin{cases} L_G + \frac{\lambda_u}{2^\ell} I & \text{if } 0 \leq \ell \leq d \\ L_G & \text{if } \ell = d+1. \end{cases}$$

where d and λ_u are defined as in Lemma A.0.1.

We often denote the matrix $L_{G^\gamma} = L_G + \gamma I$ by K, and in particular use the notation L and K interchangeably.

Effective Resistance. Given a weighted graph $G = (V,E,w)$ we associate it with an electric circuit where the vertices are junctions and each edge e is a resistor of resistance $1/w(e)$. Now suppose in this circuit we inject one unit current at vertex u, extract one from vertex v, and let $\mathbf{f}_{uv} \in \mathscr{R}^m$ denote the the currents induced on the edges. By Kirchhoff's current law, except for the source u and the sink v, the sum of the currents entering and exiting any vertex is zero. Hence, we have $\mathbf{b}_{uv} = B^{\mathrm{T}} \mathbf{f}_{uv}$. Let $\varphi \in \mathscr{R}^n$ denote the voltage potentials induced at the vertices in the above setting. By Ohm's law we have $\mathbf{f} = W B \varphi$. Putting these facts together:

$$\chi_u - \chi_v = B^{\mathrm{T}} W B \varphi = L\varphi.$$

Observe that $(\chi_u - \chi_v) \perp \ker(L)$, and hence $\varphi = L^+(\chi_u - \chi_v)$.

The *effective resistance* between vertices u and v in graph G, denoted by R_{uv} is defined as the voltage difference between vertices u and v, when a unit of current is injected into u and is extracted from v. Thus we have:

$$R_{uv} = \mathbf{b}_{uv}^{\mathrm{T}} L^+ \mathbf{b}_{uv}. \tag{2.1}$$

We also let $R_{uu} := 0$ for any $u \in V$, for convenience. For any matrix $K \in \mathscr{R}^{n \times n}$, we let $R_{uv}^K := \mathbf{b}_{uv}^{\mathrm{T}} K^+ \mathbf{b}_{uv}$.

Also, for any pair of vertices (w_1, w_2), the potential difference induced on this pair when sending a unit of flow from u to v can be calculated as:

$$\varphi(w_1) - \varphi(w_2) = \mathbf{b}_{w_1 w_2}^{\top} L^+ \mathbf{b}_{uv}. \tag{2.2}$$

Furthermore, if the graph is unweighted, the flow on edge (w_1, w_2) is

$$\mathbf{f}_{uv}(w_1 w_2) = \mathbf{b}_{w_1 w_2}^\top L^+ \mathbf{b}_{uv}. \tag{2.3}$$

We frequently use the following simple fact.

Fact 2.2.3 (See e.g. Kapralov et al. (2017a), Lemma 3)**.** *For any graph $G = (V, E, w)$, $\gamma \geq 0$ and any Laplacian matrix $L \in \mathscr{R}^V$, let $K = L + \gamma I$. Then, for any pair of vertices $(u, v), (u', v') \in V \times V$,*

$$|\mathbf{b}_{u'v'}^\top K^+ \mathbf{b}_{uv}| \leq \mathbf{b}_{uv}^\top K^+ \mathbf{b}_{uv}.$$

Proof. Let $\varphi = K^+ \mathbf{b}_{uv}$. Suppose that for some $x \in V \setminus \{u\}$, $\varphi(x) > \varphi(u)$. Then, since $K = L + \gamma I$ is a full rank and diagonally dominant matrix, then one can easily see that we should have $b_{uv}(x) > 0$, which is a contradiction. So, $\varphi(u) \geq \varphi(x)$ for any $x \in V \setminus \{u\}$. In a similar way, we can argue that $\varphi(v) \leq \varphi(y)$ for any $y \in V \setminus \{v\}$. So, the claim holds. □

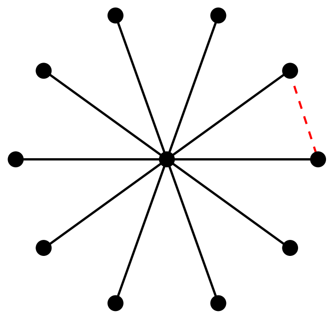
Spectral Approximation. For matrices $C, D \in \mathscr{R}^{p \times p}$, we write $C \preceq D$, if $\forall x \in \mathscr{R}^p$, $x^\mathrm{T} C x \leq x^\mathrm{T} D x$. We say that $\widetilde{C}$ is $(1 \pm \epsilon)$-spectral sparsifier of C, and we write it as $\widetilde{C} \approx_\epsilon C$, if $(1-\epsilon)C \preceq \widetilde{C} \preceq (1+\epsilon)C$. Graph $\widetilde{G}$ is $(1 \pm \epsilon)$–spectral sparsifier of graph G if, $L_{\widetilde{G}} \approx_\epsilon L_G$. We also sometimes use a slightly weaker notation $(1-\epsilon)C \preceq_r \widetilde{C} \preceq_r (1+\epsilon)C$, to indicate that $(1-\epsilon)x^\top C x \leq x^\top \widetilde{C} x \leq (1+\epsilon) x^\top C x$, for any x in the row span of C.

2.3 Main result

We start by giving some intuition and presenting the high level idea of our algorithm in Section 2.3.1 below. In Section 2.3.2 we formally state the algorithm and provide correctness analysis. In Section 2.3.3 we describe how the required sketches can be implemented using the efficient pseudorandom number generator from Kapralov et al. (2019a). Finally in Section 2.3.4 we give the proof of Theorem 2.1.1.

2.3.1 Overview of the approach

To illustrate our approach, suppose for now that our goal is to find edges with effective resistance at least $\frac{1}{\log n}$ in a graph $G = (V, E)$, which we denote by "heavy edges". This task has been studied in prior work on spectral sparsification Kapralov et al. (2017a) and was essentially shown in Kapralov et al. (2019a) to be sufficient to yield a spectral sparsification with only almost constant overhead. Each of Kapralov et al. (2017a) and Kapralov et al. (2019a) solve this problem by running ℓ_2-heavy hitters on approximate flow vectors, obtained by coarse sparsifier of the graph. The number of test flow vectors used in Kapralov et al. (2017a) is quadratic in the number of vertices, i.e., they brute force on all pair of vertices to find the heavy edges, and this was improved to $n^{1.4+o(1)}$ in Kapralov et al. (2019a). Consequently, a natural question that one could attack to further improve the running times of these methods is the following:


(a) Graph of Example 2.3.1

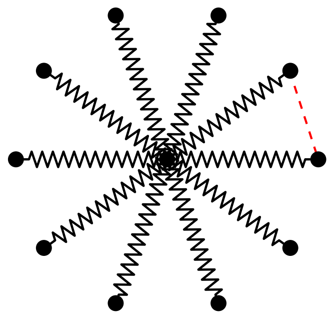

(b) Graph of Example 2.3.2

Figure 2.1 – (a) Graph of Example 2.3.1. A star with n petals along with one additional edge. (b) Graph of Example 2.3.2. A star graph with $\Theta(n^{0.7})$ petals, along with one additional edge. Each zigzag represents a path of connected cliques with effective resistance diameter $O(1)$.

Can we efficiently find a nearly linear number of test vectors that enable us to recover all heavy edges?

In this work, we answer this question in the affirmative and formally show that there exist a linear number of test vectors, which suffice to find all heavy edges. This is essentially the key technical contribution of this chapter and generalizing this solution yields our main algorithmic results.

To illustrate our approach, suppose that one can compute the flow vector using the following formula [2]

$$BL^{+}\mathbf{b}_{uv} = \mathbf{f}_{uv} \tag{2.4}$$

for any pair of vertices in polylogarithmic time (in our actual algorithms we will be unable to compute these flow vectors exactly). Note that

$$||\mathbf{f}_{uv}||_2^2 = \mathbf{b}_{uv}^{\top} L^{+} B^{\top} B L^{+} \mathbf{b}_{uv} = \mathbf{b}_{uv}^{\top} L^{+} \mathbf{b}_{uv} = R_{uv} \tag{2.5}$$

and

$$\mathbf{f}_{uv}(uv) = \mathbf{b}_{uv}^{\top} L^{+} \mathbf{b}_{uv} = R_{uv}. \tag{2.6}$$

This implies that, when $R_{uv} > \frac{1}{\log n}$, the contribution of uv coordinate of this vector to the ℓ_2 norm is substantial, and known ℓ_2-heavy hitters can recover this edge using corresponding sketches, efficiently. One should note that ℓ_2-heavy hitter returns a set of edges with $\Omega(\frac{1}{\operatorname{polylog} n})$ contribution to the ℓ_2^2 of the flow vector. A natural question that arises is whether it is possible to recover a heavy edge without using its flow vector, but rather using other flow vectors. Consider the following example.

Example 2.3.1 (Star Graph Plus Edge)**.** *Suppose that graph $G = (V, E)$ is a "star" with a center and n petals along with one additional edge that connects a pair of petals, i.e., $V = \{v_1, v_2, \ldots, v_n\}$ and $E =$*

[2]Note that in the actual algorithm we use K^{+} as opposed to L^{+}, since we work with regularized versions of the Laplacian of G, denoted by K. We use L in this overview of our techniques to simplify notation.

$\{(v_1,v_2),(v_1,v_3),\cdot,(v_1,v_n)\}\cup\{(v_2,v_3)\}$ (see Fig. 2.1a).

Clearly, for edge (v_2,v_3), $R_{v_2v_3}=\frac{2}{3}$. Suppose that we want to recover this edge by examining an electrical flow vector other than $\mathbf{f}_{v_2v_3}$. We can in fact pick an arbitrary vertex $x\in V\setminus v_2$ and send one unit of flow to v_2. Regardless of the choice of x, edge (v_2,v_3) contributes an $\Omega(1)$ fraction of the energy of the flow, and thus can be recovered by applying heavy hitters to $\mathbf{f}_{xv_2}$. Similarly, for any v_i, when one unit of flow is sent from x to v_i, at least a constant fraction of the energy is contributed by edge (x,v_i). So, all high effective resistance edges in this graph (all edges) can be recovered using $n-1$ simple flow vectors, i.e., $\{\mathbf{f}_{xv_1},\ldots,\mathbf{f}_{xv_n}\}$.

Of course, the graph in Example 2.3.1 has only n edges, and so could be stored explicitly in the streaming setting, without needing to recover edges from heavy hitter queries. However, we can give a similar example which is in fact dense.

Example 2.3.2 (Thick Star Plus Edge)**.** *Suppose that graph G is a dense version of the previous example as follows: it has a center and $\Theta(n^{0.7})$ petals. Each petal consists of a chain of $\Theta(n^{0.2})$ cliques of size $n^{0.1}$, where each pair of consecutive cliques is connected with a complete bipartite graph. One can verify that the effective resistance diameter of each petal is $\Theta(1)$. Now, we add an additional edge, e, that connects an arbitrary node in the leaf of one petal to a node in the leaf of another petal (see Fig. 2.1b).*

As in Example 2.3.1, e is heavy, with $R_e=\Theta(1)$. In fact, it is the only heavy edge in the graph. One can verify that, similar to Example 2.3.1, if we let C_2 and C_3 denote the cliques that e connects, choosing an arbitrary vertex x and sending flow to any node in C_2 and then to any node in C_3, will give an electrical flow vector where e contributes an $\Omega(1)$ fraction of the energy. Thus, e can be recovered by applying heavy hitters to these vectors. Consequently, using n test vectors (sending flow from x to each other node in the graph) one can recover all heavy edges of this example.

Unfortunately, it is possible to give an example where the above simple procedure of checking the flow from an arbitrary vertex to all others fails.

Example 2.3.3 (Thick Line Plus Edge)**.** *Suppose that graph $G=(V,E)$ is a thick line, consisting of $n^{0.9}$ set of points (clusters) where any two consecutive clusters form a complete bipartite graph. Formally, $V=\{v_1,v_2,\ldots,v_n\}=C_1\cup C_2\cup\cdots\cup C_{n^{0.9}}$, where C_i's are disjoint sets of size $n^{0.1}$ and*

$$E=\bigcup_{i=1}^{n^{0.9}-1}C_i\times C_{i+1}.$$

Also, add an edge $e=(u,v)$ such that $u\in C_1$ and $v\in C_{n^{0.2}}$ (see Fig. 2.2).

One can verify that $R_e=\Omega(1)$. However, if one picks an arbitrary vertex $x\in V$ and sends one unit of flow each other vertex, running ℓ_2-heavy hitters on each of these flows will not recover edge e if x is far from u and v in the thick path. Any flow that must cross (u,v) will have very large energy due to the fact that it must travel a long distance to the clusters containing these vertices, so e will not contribute non-trivial fraction.

Fortunately, the failure of our recovery method in Example 2.3.3 is due to a simple fact: the effective resistance diameter of the graph is large. When the effective resistance diameter is small (as in Examples 2.3.1 and 2.3.2) the strategy always suffices. This follows from the following simple observation:

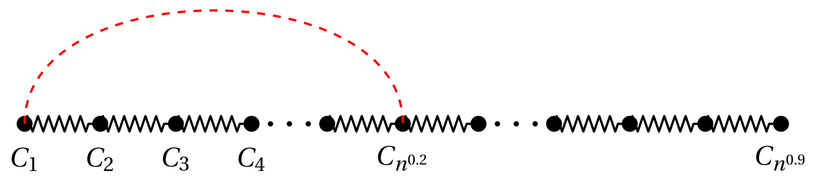


Figure 2.2 – Graph of Example 2.3.3. Each C_i represents a cluster with $n^{0.1}$ vertices (with no internal edges) and each zigzag represents the edges of a complete bipartite graph between consecutive C_i's.

Observation 2.3.4. *For a graph $G = (V, E)$, suppose that for an edge $e = (u, v) \in E$, one has*

$$R_e \geq \beta.$$

Then, for any $x \in V$, in at least one of these settings, edge e carries at least $\beta/2$ units of flow:

1. *One unit of flow is sent from x to u.*
2. *One unit of flow is sent from x to v.*

This observation follows formally from the following simple lemma.

Lemma 2.3.5. *For a graph $G = (V, E)$, suppose that $D \in \mathscr{R}^{V\times V}$ is a PSD matrix. Then, for any pair of vertices $(u, v) \in \binom{V}{2}$ and for any vertex $x \in V \setminus \{u, v\}$,*

$$\max\{|\mathbf{b}_{xu}^\top D\mathbf{b}_{uv}|, |\mathbf{b}_{xv}^\top D\mathbf{b}_{uv}|\} \geq \frac{\mathbf{b}_{uv}^\top D\mathbf{b}_{uv}}{2}.$$

Proof. Note that

$$\mathbf{b}_{uv}^\top D\mathbf{b}_{uv} = \left(\mathbf{b}_{ux}^\top + \mathbf{b}_{xv}^\top\right) D\mathbf{b}_{uv} = \mathbf{b}_{ux}^\top D\mathbf{b}_{uv} + \mathbf{b}_{xv}^\top D\mathbf{b}_{uv},$$

and hence, the claim holds. □

Consider the setting where $\beta = \frac{1}{\log n}$. The observation guarantees that edge e contributes at least $\frac{1}{4\log^2 n}$ energy to either flow $\mathbf{f}_{xv}$ or $\mathbf{f}_{xu}$. Thus, we can recover this edge via ℓ_2 heavy hitters, as long as the total energy $\|\mathbf{f}_{xv}\|_2^2$ or $\|\mathbf{f}_{xu}\|_2^2$ is not too large. Note that this energy is just equal to the effective resistance R_{xv} between x and v (respectively x and u). Thus it is bounded if the effective resistance diameter is small, demonstrating that our simple recovery procedure always succeeds in this setting. For example, if the diameter is $O(1)$, both $\|\mathbf{f}_{xv}\|_2^2 = O(1)$ and $\|\mathbf{f}_{xu}\|_2^2 = O(1)$, and so by Observation 2.3.4, edge $e = (u, v)$ contributes at least a $\Theta\left(\frac{1}{\log^2 n}\right)$ fraction of the energy of at least one these flows.

We next explain how to extend this procedure to handle general graphs, like that of Example 2.3.3.

Ball carving in effective resistance metric: When the effective resistance diameter of G is large, if we attempt to recover e using ℓ_2-heavy hitters on the flow vectors $\mathbf{f}_{xu}$ and $\mathbf{f}_{xv}$, for an arbitrary chosen $x \in V$, we may fail if the effective resistance distance between x and v or u ($\|\mathbf{f}_{xv}\|_2^2$ or $\|\mathbf{f}_{xu}\|_2^2$) is large. This is exactly what we saw in Example 2.3.3.

However, using the fact that $||\mathbf{f}_{xv}||_2^2 = R_{xv}$, our test will succeed if we find a vertex x, which is close to u and v in the effective resistance metric. This suggests that we should partition the vertices into cells of fairly small effective resistance diameter, ensuring that both endpoints of an edge (u, v) that we would like to recover fall in the same cell with nontrivially large probability. This is exactly what standard metric decomposition techniques achieve through a ball-carving approach, which we use, as described next.

Partitioning the graph into low effective resistance diameter sets: It is well-known that using Johnson-Lindenstrauss (JL) dimension reduction (see Lemma A.0.4), one can embed vertices of a graph in $\mathscr{R}^q$, for $q = O(\log n)$, such that the Euclidean distance squares correspond to a constant factor multiplicative approximation to effective resistance of corresponding vertices. We then partition $\mathscr{R}^q$ intro ℓ_∞ balls centered at points of a randomly shifted infinite q-dimensional grid with side length $w > 0$, essentially defining a hash function that maps every point in $\mathscr{R}^q$ to the nearest point on the randomly shifted grid. We then bound the maximum effective resistance of pair of vertices in the same bucket (see Claim 2.3.8), and show how an appropriate choice of the width w ensures that u and v belong to the same cell, with a probability no less than a universal constant (see Claim 2.3.7). This ensures that in at least one of $O(\log n)$ independent repetitions of this process with high probability, u and v fall into the same cell. We note that the parameters of our partitioning scheme can be improved somewhat using Locality Sensitive Hashing techniques (e.g., Indyk and Motwani (1998); Datar et al. (2004); Andoni and Indyk (2006); Andoni et al. (2014); Andoni and Razenshteyn (2015)). More precisely, LSH techniques would improve the space complexity by polylogarithmic factors at the expense of slightly higher runtime (the best improvement in space complexity would result from Euclidean LSH Andoni and Indyk (2006); Andoni and Razenshteyn (2015), at the cost of an $n^{o(1)}$ additional factor in runtime). However, since the resulting space complexity does not quite match the lower bound of $\Omega(n \log^3 n)$ due to Nelson and Yu (2019), we leave the problem of fine-tuning the parameters of the space partitioning scheme as an exciting direction for further work.

Sampling edges with probability proportional to effective resistances: The above techniques can actually be extended to recover edges of any specific target effective resistance. Broadly speaking, if we aim to capture edges of effective resistance about R, we can afford to lower our grid cell size proportionally to R. Unfortunately, these edges don't contribute enough to the flow vector to be recoverable. Thus, we will also subsample the edges of the graph at rate approximately proportional to R to allow us to detect the target edges while also subsampling them.

2.3.2 Our algorithm and proof of main result

As mentioned in the introduction, our algorithm consists of two phases. In the first phase, our algorithm maintains sketches of the stream, updating the sketches at each edge addition or deletion. Then, in the second phase, when queried, it can recover a spectral sparsifier of the graph from the sketches that have been maintained in the first phase. In the following lines, we give a brief overview of each phase:

Updating sketches in the dynamic stream. Our algorithm maintains a set of sketches ΠB, of size $O(n \operatorname{polylog}(n) \cdot \epsilon^{-2})$, and updates them each time it receives an edge addition or deletion in the stream. ΠB consists of multiple sketches $(\Pi_s^\ell B_s^\ell)_{\ell,s}$ where B_s^ℓ is a subsampling of the edges in B at rate 2^{-s} and Π_s^ℓ is an ℓ_2 heavy hitters sketch. In Section 2.3.3 we discuss these sketches in more detail and we show that the update time for each edge addition or deletion is $O(\operatorname{polylog}(n) \cdot \epsilon^{-2})$.

Recursive sparsification: After receiving the updates in the form of a dynamic stream, as described above, our algorithm uses the maintained sketches to recover a spectral sparsifier of the graph. This is done recursively, and heavily relies on the idea of a chain of coarse sparsifiers described in Lemma A.0.1. For a regularization parameter ℓ between 0 and $d = O(\log n)$ the task of $\textsc{Sparsify}(\Pi^{\le \ell} B, \ell, \epsilon)$ is to output a spectral sparsifier to matrix L_ℓ, which is defined as follows:

$$L_\ell = \begin{cases} L_G + \frac{\lambda_u}{2^\ell} I & \text{if } 0 \le \ell \le d \\ L_G & \text{if } \ell = d+1. \end{cases}$$

where $d = \lceil \log_2 \frac{\lambda_u}{\lambda_\ell} \rceil$ (see Lemma A.0.1 for more details about chain of coarse sparsifiers). Note that the call receives a collection of sketches $\Pi^{\le \ell} B$ as input that suffices for all recursive calls with smaller values of ℓ. So, in order to get a sparsifier of the graph we invoke $\textsc{Sparsify}(\Pi^{\le d+1} B, d+1, \epsilon)$, which receives all the sketches maintained throughout the stream and passes the required sketches to the recursive calls in Line 6 of Algorithm 1. This recursive algorithm takes as input $\Pi^{\le \ell} B$ corresponding to the parts of the sketch used to recover a spectral approximation to L_k for all $k \le \ell$, ℓ corresponding to the current L_ℓ which we wish to recover a sparsifier of, and ϵ corresponding to the desired sparsification accuracy. The algorithm first invokes itself recursively to recover $\tilde{K}$, a spectral approximation for $L_{\ell-1}$ (or uses the trivial approximation $\lambda_u I$ when $\ell = 0$). The effective resistance metric induced by $\tilde{K}$ is then approximated using the Johnson-Lindenstrauss lemma (JL). Finally, the procedure $\textsc{RecoverEdges}$ (i.e. Algorithm 2) uses this metric and the heavy hitters sketches $(\Pi_s^\ell B_s^\ell)_s$. We formally state our algorithm, Algorithm 2 below.

Algorithm 1 $\textsc{Sparsify}(\Pi^{\leq \ell}B, \ell, \epsilon)$

```
procedure $\textsc{Sparsify}(\Pi^{\leq \ell}B, \ell, \epsilon)$
    $W \leftarrow 0^{n\times n}$
    if $\ell = 0$ then
        $\widetilde{K} \leftarrow \lambda_u I$
    else
        $\widetilde{K} \leftarrow \frac{1}{2(1+\epsilon)}\textsc{Sparsify}(\Pi^{\leq \ell-1}B, \ell-1, \epsilon)$
    $\widetilde{B} \leftarrow$ the edge vertex incident matrix of $\widetilde{K}$ (discarding the regularization)
    $\widetilde{W} \leftarrow$ the diagonal matrix of weights over the edges of $\widetilde{K}$ (discarding the regularization)
    $Q \leftarrow q \times \binom{n}{2}$ is a random $\pm 1$ matrix for $q \leftarrow 1000 \log n$
                                    ▷ $q$ above is chosen as it suffices to get a $(1 \pm \frac{1}{5})$ approximation from JL
    $M \leftarrow \frac{1}{\sqrt{q}} Q \widetilde{W}^{1/2} \widetilde{B} \widetilde{K}^{+}$          ▷ $M$ is such that $R^{\widetilde{K}}_{uv} \leq \frac{5}{4}||M(\chi_u - \chi_v)||_2^2 \leq \frac{3}{2} R^{\widetilde{K}}_{uv}$ w.h.p.
                                    ▷ $R^{\widetilde{K}}$ is the effective resistance metric in $\widetilde{K}$
    for $s \in [-\log(3 \cdot c_2 \cdot \log n \cdot \epsilon^{-2}), 10 \log n]$ do
        $E_s \leftarrow \textsc{RecoverEdges}(\Pi^{\ell}_{s^+} B^{\ell}_{s^+}, M, \widetilde{K}^{+}, s, q, \epsilon)$     ▷ We use the notation $s^+ = max(0, s)$
        for $e \in E_s$ do
            $W(e,e) \leftarrow 2^{-(s^+)}$
    if $\ell = \left\lceil \log_2 \frac{\lambda_u}{\lambda_\ell} \right\rceil + 1$ then
        $\gamma \leftarrow 0$
    else
        $\gamma \leftarrow \frac{\lambda_u}{2^{\ell}}$
    return $B_n^{\top} W B_n + \gamma I$.
```

Algorithm 2 (the $\textsc{RecoverEdges}$ primitive) is the core of Algorithm 1. It receives a parameter s as input, and its task is to recover edge of effective resistance $\approx \frac{\epsilon^2}{\log n} 2^{-s}$ from a sample at rate $\min(1, O(\frac{1}{\epsilon^2} \log n \cdot 2^{-s}))$ from an appropriate sketch. It is convenient to let s range from $-O(\log(\log n/\epsilon^2))$ to $O(\log n)$, so that the smallest value of s corresponds to edges of constant effective resistance. That way the sampling level corresponding to s is simply equal to $s^+ := \max(0, s)$. Therefore Algorithm 2 takes as input a heavy hitters sketch $\Pi^{\ell}_{s^+} B^{\ell}_{s^+}$ of $B^{\ell}_{s^+}$, the edge incidence matrix of L_ℓ sampled at rate 2^{-s^+}, an approximate effective resistance embedding M, the target sampling probability 2^{-s}, the dimension q of the embedding, and the target accuracy ϵ. This procedure then performs the previously described random grid hashing of the points using the effective resistance embedding and queries the heavy hitters sketch to find the edges sampled at the appropriate rate.

The development and analysis of $\textsc{RecoverEdges}$ (Algorithm 2) is the main technical contribution of our paper. In the rest of the section we prove correctness of Algorithm 2 (Lemma 2.3.6, our main technical lemma), and then provide a correctness proof for Algorithm 1, establishing Theorem 2.3.9. We then put these results together with runtime and space complexity bounds to obtain a proof of Theorem 2.1.1.

Lemma 2.3.6 below is our main technical lemma. Specifically, Lemma 2.3.6 proves that if Algorithm 1 successfully executes all lines before Line 13, then each edge is sampled and weighted properly (as required by Theorem A.0.2), in the remaining steps.

Algorithm 2 RecoverEdges$(\Pi^{\ell}_{s^+}B^{\ell}_{s^+}, M, \widetilde{K}^+, s, q, \epsilon)$

```
procedure RecoverEdges$(\Pi^{\ell}_{s^+}B^{\ell}_{s^+}, M, \widetilde{K}^+, s, q, \epsilon)$
    $E' \leftarrow \emptyset$.    ▷ $q$ is the dimension to perform hashing in, $s$ is the sampling level
    $C \leftarrow$ the constant in the proof of Lemma 2.3.6
    $c_2 \leftarrow$ the oversampling constant of Theorem A.0.2
    $w \leftarrow 2q \cdot \sqrt{\frac{\epsilon^2}{c_2 \cdot 2^s \cdot \log n}}$.
    for $j \in [10 \log n]$ do
        For each dimension $i \in [q]$, choose $s_i \sim \mathrm{Unif}([0, w])$.
        Initialize $H \leftarrow \emptyset$ to an empty hash table
        for $u \in V$ do    ▷ Hash vertices to points on randomly shifted grid
            For all $i \in [q]$, let $\mathscr{G}(u)_i := \left\lfloor \frac{(M\chi_u)_i - s_i}{w} \right\rfloor$.
            Insert $u$ into $H$ with key $\mathscr{G}(u) \in \mathbb{Z}^q$
                ▷ $\mathscr{G}(u) \in \mathbb{Z}^q$ indexes a point on a randomly shifted grid
        for $b \in \mathrm{keys}(H)$ do    ▷ $b \in \mathbb{Z}^q$ indexes a point on a randomly shifted grid
            $x \leftarrow$arbitrary vertex in $H^{-1}(b)$
            for $v \in H^{-1}(b) \setminus \{x\}$ do
                $F \leftarrow$ HeavyHitter$\left(\Pi^{\ell}_{s^+}B^{\ell}_{s^+}\widetilde{K}^+\mathbf{b}_{xv}, \frac{1}{2} \cdot \frac{1}{C \cdot q^3} \cdot \sqrt{\frac{\epsilon^2}{\log n}}\right)$.    ▷ As per Lemma A.0.3
                for $e \in F$ do
                    $p'_e \leftarrow \frac{5}{4} \cdot c_2 \cdot ||Mb_e||_2^2 \cdot \log n / \epsilon^2$
                    if $p'_e \in (2^{-s-1}, 2^{-s}]$ then
                        $E' \leftarrow E' \cup \{e\}$.
    return $E'$.
```

Lemma 2.3.6 (Edge Recovery)**.** *Consider an invocation of* RecoverEdges$(\Pi^{\ell}_{s^+}B^{\ell}_{s^+}, M, \widetilde{K}, s, q, \epsilon)$ *of Algorithm 2, where $\Pi^{\ell}_{s^+}B^{\ell}_{s^+}$ is a sketch of the edge incidence matrix B of the input graph G as described in Section 2.3.3, s is some integer, and $\epsilon \in (0, 1/5)$. Suppose further that $\widetilde{K}$ and M satisfy the following guarantees:*

(A) *$\widetilde{K}$ is such that $\frac{1}{3} \cdot L_{\ell} \preceq_r \widetilde{K} \preceq_r L_{\ell}$ (see Lines 4 and 6 of Algorithm 1)*

(B) *M is such that for any pair of vertices u and v, $R^{\widetilde{K}}_{uv} \le \frac{5}{4}||M(\chi_u - \chi_v)||_2^2 \le \frac{3}{2}R^{\widetilde{K}}_{uv}$ ($R^{\widetilde{K}}$ is the effective resistance metric in $\widetilde{K}$; see Line 11 of Algorithm 1)*

Then, with high probability, for every edge e, RecoverEdges$(\Pi^{\ell}_{s^+}B^{\ell}_{s^+}, M, \widetilde{K}, s, q, \epsilon)$ *will recover e if and only if:*

(1) *$\frac{5}{4} \cdot c_2 \cdot ||Mb_e||_2^2 \cdot \log(n)/\epsilon^2 \in (2^{-s-1}, 2^{-s}]$ where c_2 is the oversampling constant of Theorem A.0.2 (see Lines 18 and 19 of Algorithm 2), and*

(2) *edge e is sampled in $B^{\ell}_{s^+}$.*

The proof of Lemma 2.3.6 relies on the following two claims regarding the hashing scheme of Algorithm 2. First, Claim 2.3.7 shows that the endpoints of an edge of effective resistance bounded by a threshold most likely get mapped to the same grid point in the random hashing step in Line 10 of Algorithm 2.

Claim 2.3.7 (Hash Collision Probability)**.** *Let q be a positive integer and let the function $\mathscr{G} : \mathscr{R}^q \to \mathbb{Z}^q$ define a hashing with width $w > 0$ as follows:*

$$\forall i \in [q], \ \ \mathscr{G}(u)_i = \left\lfloor \frac{u_i - s_i}{w} \right\rfloor$$

where $s_i \sim \mathrm{Unif}[0, w]$, as per Line 10 of Algorithm 2. If for a pair of points $x, y \in \mathscr{R}^q$, $||x - y||_2 \le w_0$ and $w \ge 2w_0 q$, then $\mathscr{G}(x) = \mathscr{G}(y)$ with probability at least $1/2$.

Proof. First note that by union bound

$$\mathbb{P}(\mathscr{G}(x) \neq \mathscr{G}(y)) = \mathbb{P}(\exists i : \mathscr{G}(x)_i \neq \mathscr{G}(y)_i) \le \sum_{i=1}^{q} \mathbb{P}(G(x)_i \neq G(y)_i)\,. \tag{2.7}$$

Now let us bound each term of the sum.

$$\begin{aligned} \mathbb{P}(\mathscr{G}(x)_i \neq \mathscr{G}(y)_i) &= \mathbb{P}\left(\left\lfloor \frac{x_i - s_i}{w} \right\rfloor \neq \left\lfloor \frac{y_i - s_i}{w} \right\rfloor\right) \\ &= \frac{|x_i - y_i|}{w} \\ &\le \frac{||x - y||_2}{w} \\ &\le \frac{1}{2q} \end{aligned} \tag{2.8}$$

Combining Eq. (2.7) and Eq. (2.8), we get that $\mathbb{P}(\mathscr{G}(x) \neq \mathscr{G}(y)) \le 1/2$ as claimed. □

The next claim, Claim 2.3.8 bounds the effective resistance diameter of buckets in the hash table constructed in Line 11 of Algorithm 2.

Claim 2.3.8 (Hash Bucket Diameter)**.** *Let the function $\mathscr{G} : \mathscr{R}^q \to \mathbb{Z}^q$, for some integer q, define a hashing with width $w > 0$ as follows:*

$$\forall i \in [q], \ \ \mathscr{G}(u)_i = \left\lfloor \frac{u_i - s_i}{w} \right\rfloor$$

where $s_i \sim \mathrm{Unif}[0, w]$, as per Line 10 of Algorithm 2. For any pair of points $u, v \in \mathscr{R}^q$, such that $\mathscr{G}(u) = \mathscr{G}(v)$, one has

$$||u - v||_2 \le w \cdot \sqrt{q}.$$

Proof. Since $\mathscr{G}(u) = \mathscr{G}(v)$, then

$$\begin{aligned} ||u - v||_2 &= \sqrt{\sum_{i \in q} (u_i - v_i)^2} \\ &\le \sqrt{w^2 \cdot q} && \text{since } \forall i \in q, |u_i - v_i| \le w \\ &= w \cdot \sqrt{q}. \end{aligned}$$

□

Using Claim 2.3.7 and Claim 2.3.8 we now prove Lemma 2.3.6.

Proof of Lemma 2.3.6: Let $p'_e := \frac{5}{4} \cdot c_2 \cdot ||Mb_e||_2^2 \cdot \log(n)/\epsilon^2$. First note that both of conditions **(1)** and **(2)** are necessary. Indeed, if e is not sampled in B_s, it will never be returned by HEAVYHITTER in Line 16, and if $p'_e \notin (2^{-s-1}, 2^{-s}]$ then e will not be added to E' due to Line 19 of Algorithm 2. It remains to show that the two conditions are sufficient to recover e with high probability.

For an edge $(u, v) = e \in E$ satisfying conditions **(1)** and **(2)** we prove that the size of the grid (w as defined in Line 5 of Algorithm 2) is large enough to capture edge e, as described by Claim 2.3.7. Specifically, we invoke the claim with $w_0 = ||Mb_e||_2$. Note that we have $w \geq 2qw_0$ by the setting of w in Line 5 and the fact that

$$||Mb_e||_2 = \sqrt{\frac{4}{5} \cdot p'_e \cdot \frac{\epsilon^2}{c_2 \cdot \log n}} \leq \sqrt{\frac{\epsilon^2}{c_2 \cdot 2^s \cdot \log n}},$$

where we used the fact that $p'_e \leq 2^{-s}$. Thus, we have $w \geq 2qw_0$ as prescribed by Claim 2.3.7, so u and v fall into the same cell with probability at least $1/2$ in a single instance of hashing. Hashing is then repeated $10 \log n$ times to guarantee that they fall into the same cell at least once with high probability, see Line 6 of Algorithm 2.

Consider now an instance of hashing where u and v fall into the same cell, say $\mathscr{C}$ (which corresponds to a hash bucket in our hash table H). Let x be chosen arbitrarily from $\mathscr{C}$ as per Line 14 of Algorithm 2 . Our algorithm sends electrical flow from x to both u and v and by Observation 2.3.4 in at least one of these flows e will have weight $R_e^{\widetilde{K}}/2$. More precisely, by Lemma 2.3.5 invoked with $D = \widetilde{K}^+$ we have

$$\max\{|\mathbf{b}_{xu}^\top \widetilde{K}^+ \mathbf{b}_{uv}|, |\mathbf{b}_{xv}^\top \widetilde{K}^+ \mathbf{b}_{uv}|\} \geq \frac{\mathbf{b}_{uv}^\top \widetilde{K}^+ \mathbf{b}_{uv}}{2} = R_e^{\widetilde{K}}/2. \tag{2.9}$$

Without loss of generality assume that this is the flow from x to v.

It remains to show, that unlike in Example 2.3.3, the total energy of the xv flow does not overshadow the contribution of edge e. Intuitively this is because the effective resistence of e is proportional to $2^{-s} \cdot \epsilon^2$ and therefore its ℓ_2-contribution is proportional to $2^{-2s} \cdot \epsilon^4$. On the other hand, the effective resistence diameter of $\mathscr{C}$ is proportional to $2^{-s} \cdot \epsilon^2$, which bounds the energy of the xv flow before subsampling. Subsampling at rate 2^{-s} decreases the energy by a factor of 2^{-s} in expectation, and the energy concentrates sufficiently around its expectation with high probability. We prove everything in more detail below. It turns out that the actual ratio between contribution of e and the entire energy of the subsampled flow is polylogarithmic in n and quadratic in ϵ. Therefore, we can afford to store a heavy hitter sketch powerful enough to recover e.

Now let $\widetilde{\mathbf{f}}_{xv} = B\widetilde{K}^+(\chi_x - \chi_v)$, and $\widetilde{\mathbf{f}}_{xu} = B\widetilde{K}^+(\chi_x - \chi_u)$. Note that $\mathbf{f}_{xv} \in \mathscr{R}^{\binom{n}{2}}$ is a vector whose nonzero entries are exactly the voltage differences across edge in G when one unit of current is forced from x to v in $\widetilde{K}$. We have, writing L instead of L_ℓ to simplify notation,

$$\begin{aligned} ||\widetilde{\mathbf{f}}_{xv}||_2^2 &= (\chi_x - \chi_v)^\top \widetilde{K}^+ B^\top B \widetilde{K}^+ (\chi_x - \chi_v) \\ &\leq (\chi_x - \chi_v)^\top \widetilde{K}^+ L \widetilde{K}^+ (\chi_x - \chi_v) && \text{Since } B^\top B \preceq L \end{aligned}$$

$$\begin{aligned}
&\le 3\cdot(\chi_x-\chi_v)^\top \widetilde{K}^+ \widetilde{K}\widetilde{K}^+(\chi_x-\chi_v) && \text{Since } L \preceq 3\cdot\widetilde{K} \text{ by assumption } \textbf{(A)}\\
&= 3\cdot(\chi_x-\chi_v)^\top \widetilde{K}^+(\chi_x-\chi_v) && \text{of the lemma}\\
&= 3\cdot R_{xv}^{\widetilde{K}}
\end{aligned}$$

Moreover we have

$$\widetilde{\mathbf{f}}_{xv}(uv) = (\chi_u-\chi_v)^{\mathrm{T}}\widetilde{K}^+(\chi_x-\chi_v)$$

and

$$\widetilde{\mathbf{f}}_{xu}(uv) = (\chi_u-\chi_v)^{\mathrm{T}}\widetilde{K}^+(\chi_x-\chi_u).$$

For simplicity, let

$$\beta := \frac{\epsilon^2}{c_2\cdot\log n}.$$

By Eq. (2.9) we have

$$\begin{aligned}
|\widetilde{\mathbf{f}}_{xv}(uv)| &\ge \frac{b_{uv}^\top\widetilde{K}^+b_{uv}}{2}\\
&\ge \frac{5}{12}\cdot||M\mathbf{b}_e||_2^2 && \text{By assumption } \textbf{(B)} \text{ of the lemma}\\
&\ge \frac{1}{3}\cdot\frac{1}{2^{s+1}\cdot c_2}\cdot\frac{\epsilon^2}{\log n} && \text{Since } p'_e \ge \frac{1}{2^{s+1}}\\
&= \frac{1}{3}\cdot\frac{\beta}{2^{s+1}}
\end{aligned} \tag{2.10}$$

Since x, v belong to the same cell, by Claim 2.3.8, $||M(\chi_u-\chi_v)||_2 \le w\cdot\sqrt{q}$, thus,

$$\begin{aligned}
||\widetilde{\mathbf{f}}_{xv}||_2^2 &\le 3\cdot R_{xv}^{\widetilde{K}}\\
&\le \frac{5}{4}(w^2\cdot q) && \text{Since } R_{xv}^{\widetilde{K}} \le \frac{15}{4}||M(\chi_x-\chi_v)||_2^2 \text{ by } \textbf{(B)}\\
&= 15q^3\cdot\frac{\epsilon^2}{c_2\cdot 2^s\cdot\log n} && \text{By Line 5 of Algorithm 2}\\
&= 15q^3\cdot\frac{\beta}{2^s}
\end{aligned} \tag{2.11}$$

Now, let $\widetilde{\mathbf{f}}_{xv}^{(s)} := B_s\widetilde{K}\mathbf{b}_{xv}$ denote an independent sample of the entries of $\widetilde{\mathbf{f}}_{xv}$ with probability $\frac{1}{2^s}$. We now argue that, if the edge (u,v) is included in B_s, then it is recovered with high probability by the heavy hitter procedure HEAVYHITTER in Line 16. We let $\widetilde{\mathbf{f}}^{(s)} := \widetilde{\mathbf{f}}_{xv}^{(s)}$ and $\widetilde{\mathbf{f}} := \widetilde{\mathbf{f}}_{xv}$ (i.e., we omit the subscript xv) to simplify notation.

We will prove a lower bound on $\frac{\widetilde{\mathbf{f}}^{(s)}(uv)^2}{||\widetilde{\mathbf{f}}^{(s)}||_2^2}$ that holds with high probability. Note that

$$||\widetilde{\mathbf{f}}^{(s)}||_2^2 = \sum_{e\in B_s\setminus\{(u,v)\}}\widetilde{\mathbf{f}}^{(s)}(e)^2 + \widetilde{\mathbf{f}}^{(s)}(uv)^2 \tag{2.12}$$

For ease of notation let $X := \sum_{e\in B_s\setminus\{(u,v)\}}\widetilde{\mathbf{f}}^{(s)}(e)^2$, and let $\tau := R_{xv}^{\widetilde{K}}$. Thus, we have for a sufficiently large constant $C > 1$

$$\begin{aligned}
&\Pr\left(\frac{\widetilde{\mathbf{f}}^{(s)}(uv)^2}{\|\widetilde{\mathbf{f}}^{(s)}\|_2^2} < \frac{1}{C^2\cdot q^6}\cdot\frac{\epsilon^2}{\log n}\middle|(u,v)\in B_s\right)\\
=&\Pr\left(X > \left(\frac{C^2\cdot q^6\cdot\log n}{\epsilon^2}-1\right)\cdot\widetilde{\mathbf{f}}^{(s)}(uv)^2\middle|(u,v)\in B_s\right)\\
\le&\Pr\left(\|\widetilde{\mathbf{f}}^{(s)}\|_2^2 > \frac{1}{2}\cdot C^2\cdot q^6\cdot\frac{\log n}{\epsilon^2}\cdot\widetilde{\mathbf{f}}(uv)^2\right)\\
\le&\Pr\left(\left\|\frac{\widetilde{\mathbf{f}}^{(s)}}{\tau}\right\|_2^2 > \frac{1}{\tau^2}\cdot\frac{1}{2}\cdot C^2\cdot q^6\cdot\frac{\log n}{\epsilon^2}\cdot\widetilde{\mathbf{f}}(uv)^2\right)\\
=&\Pr\left(\|\widetilde{\mathbf{y}}^{(s)}\|_2^2 > \frac{1}{\tau^2}\cdot\frac{1}{2}\cdot C^2\cdot q^6\cdot\frac{\log n}{\epsilon^2}\cdot\widetilde{\mathbf{f}}(uv)^2\right),
\end{aligned}\tag{2.13}$$

where we let $\widetilde{\mathbf{y}} := \frac{\widetilde{\mathbf{f}}}{\tau}$ and $\widetilde{\mathbf{y}}^{(s)} := \frac{\widetilde{\mathbf{f}}^{(s)}}{\tau}$ to simplify notation in the last line and used the fact that $\widetilde{\mathbf{f}}^{(s)}(uv)^2 = \widetilde{\mathbf{f}}(uv)^2$ conditioned on $(u,v)\in B_s$ in going from line 2 to line 3. Noting that $|\widetilde{\mathbf{f}}(uv)| \ge \frac{1}{3}\cdot\frac{\beta}{2^{s+1}}$ by Eq. (2.10) and $\tau \le 5q^3\cdot\frac{\beta}{2^s}$ by Eq. (2.11), we get that the last line in Eq. (2.13) is upper bounded by

$$\Pr\left(\frac{\widetilde{\mathbf{f}}^{(s)}(uv)^2}{\|\widetilde{\mathbf{f}}^{(s)}\|_2^2} < \frac{1}{C^2\cdot q^6}\cdot\frac{\epsilon^2}{\log n}\middle|(u,v)\in B_s\right) \le \Pr\left(\|\widetilde{\mathbf{y}}^{(s)}\|_2^2 > \frac{C'\cdot\log n}{\epsilon^2}\right),\tag{2.14}$$

where C' is a constant that can be made arbitrarily large by increasing C. On the other hand, we have the following

$$\begin{aligned}
\mathbf{E}\left(\|\widetilde{\mathbf{y}}^{(s)}\|_2^2\right) &= \frac{1}{2^s}\cdot\frac{\|\widetilde{\mathbf{f}}\|_2^2}{\tau^2} && \text{Since } \widetilde{\mathbf{f}}^{(s)} \text{ is obtained by sampling at rate } \frac{1}{2^s}\\
&\le \frac{1}{2^s}\cdot\frac{3}{\tau} && \text{By Eq. (2.11)}\\
&\le \frac{1}{2^s}\cdot\frac{6}{R_{uv}^{\widetilde{K}}} && \text{By Eq. (2.9) and Fact 2.2.3}\\
&\le \frac{1}{2^s}\cdot\frac{36}{5\cdot\|M\mathbf{b}_{uv}\|_2^2} && \text{By assumption } \textbf{(B)} \text{ of the lemma}\\
&\le \frac{1}{2^s}\cdot\frac{36}{5\cdot\frac{4}{5\cdot c_2}\cdot\frac{1}{2^{s+1}}\cdot\frac{\epsilon^2}{\log n}} && \text{By condition } \textbf{(1)} \text{ of the lemma}\\
&= \frac{18\cdot c_2\cdot\log n}{\epsilon^2},
\end{aligned}$$

where the transition from line 2 to line 3 is justified by noting that

$$\tau \ge \left|(\chi_u-\chi_v)^{\mathrm{T}}\widetilde{K}^+(\chi_x-\chi_v)\right| \ge \frac{1}{2}(\chi_u-\chi_v)^{\mathrm{T}}\widetilde{K}^+(\chi_u-\chi_v) = \frac{1}{2}R_{uv}^{\widetilde{K}}$$

by Fact 2.2.3 and choice of x.

We now upper bound the right hand side of Eq. (2.14). For every entry (a,b) in $\widetilde{\mathbf{y}}$, using Fact 2.2.3 one has

$$\left|\widetilde{\mathbf{y}}_{ab}\right| = \frac{|(\chi_a-\chi_b)^{\mathrm{T}}\widetilde{K}^+(\chi_x-\chi_v)|}{\tau} \le \frac{|(\chi_x-\chi_v)^{\mathrm{T}}\widetilde{K}^+(\chi_x-\chi_v)|}{\tau} = 1$$

Thus, every entry is in $[-1,1]$, and since every entry is sampled independently, so we can use standard Chernoff/Hoeffding Hoeffding (1963) bound and we get

$$\Pr\left(||\widetilde{\mathbf{y}}^{(s)}||_2^2 > \frac{C' \cdot \log n}{\epsilon^2}\right) \le n^{-10}$$

as long as C' is a sufficiently large absolute constant (which can be achieved by making the constant C sufficiently large). Hence, we get from Eq. (2.13) that with high probability over the sampling of entries in B_s

$$\frac{|\widetilde{\mathbf{f}}^{(s)}(uv)|}{||\widetilde{\mathbf{f}}^{(s)}||_2} \ge \frac{1}{C \cdot q^3} \cdot \sqrt{\frac{\epsilon^2}{\log n}}.$$

We set $\eta = \frac{1}{2} \cdot \frac{1}{C \cdot q^3} \cdot \sqrt{\frac{\epsilon^2}{\log n}}$, thus if $|\widetilde{\mathbf{f}}^{(s)}(uv)| \ge 2\eta ||\widetilde{\mathbf{f}}^{(s)}||_2$ our sparse recovery sketch must return uv with high probability, by Lemma A.0.3. □

Theorem 2.3.9. *(Correctness of Algorithm 1) Algorithm* $\textsc{Sparsify}(\Pi^{\le \ell} B, \ell, \epsilon)$*, for* $\ell = d+1 = \lceil \log_2 \frac{\lambda_u}{\lambda_\ell} \rceil + 1$ *(see Lemma A.0.1), any* $\epsilon \in (0, 1/5)$ *and sketches* $\Pi^{\le \ell} B$ *of graph* G *as described in Section 2.3.3, returns a graph* H *with* $O(n \cdot \text{polylog}\, n \cdot \epsilon^{-2})$ *weighted edges, with Laplacian matrix* L_H*, such that*

$$L_H \approx_\epsilon L_G,$$

with high probability.

Proof. Let $\gamma = \lambda_u / 2^\ell$. As the algorithm only makes recursive calls with lower values of ℓ, we proceed by induction on ℓ.

Inductive hypothesis: A call of $\textsc{Sparsify}(\Pi^{\le \ell} B, \ell, \epsilon)$ returns a graph H^ℓ with $O(n \cdot \text{polylog}\, n \cdot \epsilon^{-2})$ weighted edges, with Laplacian matrix L_{H^ℓ}, such that

$$L_{H^\ell} \approx_\epsilon L_\ell$$

with high probability, where

$$L_\ell = \begin{cases} L_G + \frac{\lambda_u}{2^\ell} I & \text{if } 0 \le \ell \le d \\ L_G & \text{if } \ell = d+1. \end{cases}$$

and for all $\ell \ge 0$ the matrix $\widetilde{K}$ defined at the beginning of Algorithm 1 is a 3-spectral sparsifier of $G^{\gamma(\ell)}$.

Base case: $\ell = 0$.In this case we set $\widetilde{K} = \lambda_u I$ (see Line 4 of Algorithm 1). By Lemma A.0.1 we have

$$\frac{1}{2} \cdot L_\ell \preceq \widetilde{K} \preceq L_\ell, \tag{2.15}$$

i.e. $\widetilde{K}$ is a factor 3 spectral approximation of L_ℓ for $\ell = 0$. We argue that the graph output by Algorithm 1 satisfies $L_{H^\ell} \approx_\epsilon L_\ell$ below, together with the same argument for the inductive step.

Inductive step: $\ell-1 \to \ell$**.**As per Line 6 of Algorithm 1 we set $\widetilde{K} = \frac{1}{2(1+\epsilon)}\textsc{Sparsify}(\Pi^{\leq \ell-1}B, \ell-1, \epsilon)$, therefore the corresponding Laplacian for this call is $L_{\ell-1}$. By the inductive hypothesis $\textsc{Sparsify}(\Pi^{\leq \ell-1}B, \ell-1, \epsilon)$ returns an ϵ-sparsifier of $L_{\ell-1}$, so we have

$$(1-\epsilon)\cdot L_{\ell-1} \preceq_r 2(1+\epsilon)\widetilde{K} \preceq_r (1+\epsilon)\cdot L_{\ell-1}. \tag{2.16}$$

Moreover, by Lemma A.0.1, we have

$$\frac{1}{2}\cdot L_\ell \preceq \frac{1}{2}\cdot L_{\ell-1} \preceq L_\ell. \tag{2.17}$$

Putting Eq. (2.16) and Eq. (2.17) together we get

$$\frac{1-\epsilon}{2(1+\epsilon)}\cdot L_\ell \preceq_r \widetilde{K} \preceq_r L_\ell, \tag{2.18}$$

which implies for $\epsilon \leq 1/5$ that

$$\frac{1}{3}\cdot L_\ell \preceq_r \widetilde{K} \preceq_r L_\ell. \tag{2.19}$$

We thus have that for all values of ℓ the matrix $\widetilde{K}$ defined at the beginning of Algorithm 1 is a 3-spectral sparsifier of $G^{\gamma(\ell)}$, assuming the inductive hypothesis for $\ell-1$ (except for the base case case, where no inductive hypothesis is needed). Consequently, for any pair of vertices (u,v) in the same connected component in L,

$$b_{uv}^\top L_\ell^+ b_{uv} \leq b_{uv}^\top \widetilde{K}^+ b_{uv} \leq 3\cdot b_{uv}^\top L_\ell^+ b_{uv} \tag{2.20}$$

For the rest of the proof, we let $L := L_\ell$ for simplicity. We now show that the rest of the algorithm constructs an ϵ-sparsifier for L by sampling each edge e with some probability at least $\min\{1, R_{uv}^L \log(n)/\epsilon^2\}$ and giving it weight inverse proportional to the probability. This will indeed give us an ϵ-sparsifier due to Theorem A.0.2. In particular, this probability will be the following: For edges e in the appended complete graph γI the probability is 1. For an edge e in the original graph G we define the variable p'_e, as in Line 18 of Algorithm 2, to be $\frac{5}{4}\cdot c_2\cdot \|Mb_e\|_2^2 \cdot \log(n)/\epsilon^2$, and we define p_e to be $\min\{1, p'_e\}$. Let s_e be the integer such that $p'_e \in (2^{-s_e-1}, 2^{-s_e}]$. Note that then $p_e \in (2^{-s_e^+-1}, 2^{-s_e^+}]$. Our probability for sampling an edge e of the original graph will be $2^{-s_e^+}$, which is less than $\min\{1, c_2\cdot R_e^L \log(n)/\epsilon^2\}$, as required by Theorem A.0.2.

Consider the conditions of Lemma 2.3.6.

1. $\frac{1}{3}\cdot L_\ell \preceq_r \widetilde{K} \preceq_r L_\ell$ is satisfied as shown above.
2. Note that $R_{uv}^{\widetilde{K}} = \|\widetilde{W}^{1/2}\widetilde{B}\widetilde{K}^+ b_u - \widetilde{W}^{1/2}\widetilde{B}\widetilde{K}^+ b_v\|_2^2$ so we can use the Johnson-Lindenstrauss lemma to approximate $R_{uv}^{\widetilde{K}}$ using a smaller matrix. In Lines 9 and 10 we use the exact construction of Lemma A.0.4 with q being large enough for parameters $\epsilon = 1/5$ and $\beta = 6$. Therefore, $R_{uv}^{\widetilde{K}} \leq \frac{5}{4}\|M(\chi_u - \chi_v)\|_2^2 \leq \frac{3}{2}R_{uv}^{\widetilde{K}}$ is satisfied with high probability, by Lemma A.0.4.

Thus by Lemma 2.3.6 if edge e is sampled in $B_{s_e^+}^\ell$ then $\textsc{RecoverEdges}(\Pi_{s_e^+}^\ell B_{s_e^+}^\ell, M, \widetilde{K}^+, s_e, q, \epsilon)$ will recover e with high probability in Line 14 of Algorithm 1. It will then be given the required weight $(2^{s_e^+})$. Note that e will not be recovered in any other call of RECOVEREDGES, that is when $s \neq s_e$. Note also, that $2^{-s_e^+}$ is indeed an upper bound on $\min\{1, c_2 \cdot R_e^L \cdot \log(n)/\epsilon^2\}$, and within constant factor of it. Therefore, by Theorem A.0.2, the resulting graph will be an ϵ-spectral sparsifier of G^γ, and it will be $O(n \cdot \text{polylog}(n)/\epsilon^2)$-sparse (disregarding the regularization). □

2.3.3 Maintenance of sketches

Note that Algorithm 2 takes sketch ΠB as input. More precisely, Π is a concatenation of HEAVYHITTER sketch matrices composed with sampling matrices, indexed by sampling rate s and regularization level ℓ. In particular, for all s and ℓ let B_s^ℓ be a row-sampled version of B at rate 2^{-s}. Then Π_s^ℓ is a HEAVYHITTER sketch drawn from the distribution from Lemma A.0.3 with parameter $\eta = \frac{1}{2} \cdot \frac{1}{C \cdot q^3} \cdot \sqrt{\frac{\epsilon^2}{\log n}}$. Note that the matrices $\left(\Pi_s^\ell\right)_{s,\ell}$ are independent and identically distributed. We then maintain $\Pi_s^\ell B_s^\ell$ for all s and ℓ. We define

$$\Pi^\ell B = \Pi_0^\ell B_0^\ell \oplus \ldots \oplus \Pi_{10\log n}^\ell B_{10\log n}^\ell,$$

where $\oplus$ denotes concatenation of rows. We let $\Pi^{\leq \ell}$ denote $\Pi^0 \oplus \ldots \oplus \Pi^\ell$, and let Π denote $\Pi^{\leq d+1}$ to simplify notation. Thus, the algorithm maintains ΠB throughout the stream. We maintain ΠB by maintaining each $\Pi_s^\ell B_s^\ell$ individually. To this end we have for each s and ℓ an independent hash function h_s^ℓ mapping $\binom{V}{2}$ to $\{0,1\}$ independently such that $\mathbb{P}(h_s^\ell(u,v) = 1) = 2^{-s}$. Then when an edge insertion or deletion, $\pm(u,v)$, arrives in the stream, we update $\Pi_s^\ell B_s^\ell$ by $\pm \Pi_s^\ell \cdot b_{uv} \cdot h_s^\ell(u,v)$.

Overall, the number of random bits needed for all the matrices in an invocation of Algorithm 2 is at most $R = \widetilde{O}(n^2)$, in addition to the random bits needed for the recursive calls. To generate matrix Π we use the fast pseudo random numbers generator from Theorem 2.3.10 below:

Theorem 2.3.10. Kapralov et al. (2019a) *For any constants $q, c > 0$, there is an explicit pseudo-random generator (PRG) that draws on a seed of $O(S\,\text{polylog}(S))$ random bits and can simulate any randomized algorithm running in space S and using $R = O(S^q)$ random bits. This PRG can output any pseudorandom bit in $O(\log^{O(q)} S)$ time and the simulated algorithm fails with probability at most S^{-c} higher than the original.*

Observe that the space used by Algorithm 2 is $s = \widetilde{O}(n)$ in addition to the space used by the recursive calls. Since $R = O(n^2)$, we have $R = O(s^2)$. Therefore, by Theorem 2.3.10 we can generate seed of $O(s \cdot \text{poly}(()\log s))$ random bits in $O(s \cdot \text{poly}(()\log s))$ time that can simulate our randomized algorithm.

Also, note that the random matrix $Q \in \mathbb{R}^{\Theta(\log n) \times \binom{n}{2}}$ for JL (Line 9 of Algorithm 2) can be generated using $O(\log n)$-wise independent hash functions.

2.3.4 Proof of Theorem 2.1.1

Proof of Theorem 2.1.1:

Correctness of Algorithm 2 is proved in Theorem 2.3.9. It remains to prove space and runtime bounds.

Run-time and space analysis. We will prove that one call of SPARSIFY in Algorithm 1 requires $\widetilde{O}(n)$ time and space, discounting the recursive call, where n is the size of the vertex set of the input graph. Consider first Lines 9 and 11, and note that the random matrix $Q \in \mathbb{R}^{\Theta(\log n)\times\binom{n}{2}}$ for JL (Line 9 of Algorithm 2) can be generated using $O(\log n)$-wise independent hash functions, resulting in poly $(()\log n)$ time to generate an entry of Q and $O(\log n)$ space. We then multiply $Q\widetilde{W}^{1/2}\widetilde{B}$ by $\widetilde{K}^{+}$ which amounts to solving $\Theta(\log n)$ Laplacian systems and can be done in $O(n \operatorname{polylog} n \cdot \epsilon^{-2})$ time, since $\widetilde{K}$ is $O(n \operatorname{polylog}(n) \cdot \epsilon^{-2})$ sparse, using any of a variety of algorithms in the long line of improvements in solving Laplacian systems Spielman and Teng (2004); Koutis et al. (2010, 2011); Kelner et al. (2013); Lee and Sidford (2013); Peng and Spielman (2014); Cohen et al. (2014); Koutis et al. (2016); Kyng et al. (2016); Kyng and Sachdeva (2016). The resulting matrix, M, is again $\Theta(\log n \times n)$ and can be stored in $n \operatorname{polylog} n$ space. We note that the aforementioned Laplacian solvers provide approximate solutions with inverse polynomial precision, which is sufficient for application of the HEAVYHITTER sketch.

The for loops in both Line 13 and Line 6 iterate over only $\Theta(\log n)$ values. For all non-empty cells we iterate over all vertices in that cell, so overall, we iterate n times. The HEAVYHITTERS subroutine called with parameter $\eta = \epsilon / \operatorname{polylog} n$ returns by definition at most $\operatorname{polylog} n/\epsilon^2$ elements, so the for loop in Line 17 is over $\operatorname{polylog} n/\epsilon^2$ iterations. In total this is $O(n \operatorname{polylog} n \cdot \epsilon^{-2})$ time and space as claimed.

To get an ϵ-sparsifier of the input graph G, we need only to run SPARSIFY$(\Pi^{\leq d+1}B, d+1, \epsilon)$. Therefore chain of recursive calls will be $\Theta(\log(n))$ long, and the total run time will still be $\widetilde{O}(n\epsilon^{-2})$. □

3 Spectral Sparsification of Hypergraphs

This chapter is based on joint work with Michael Kapralov, Robert Krauthgamer, and Yuichi Yoshida. It appreared in two parts, which have been accepted to the 53rd ACM Symposium on Theory of Computation (**STOC**) 2021 Isard et al. (2007) under the title

Towards Tight Bounds for Spectral Sparsification of Hypergraphs,

and to the 62nd IEEE Symposium on Foundations of Computer Science (**FOCS**) 2021 Kapralov et al. (2021a) under the title

Spectral Hypergraph Sparsifiers of Nearly Linear Size.

3.1 Introduction

We study spectral sparsification of hypergraphs, where the goal is to reduce the size of a hypergraph while preserving its energy. Given a hypergraph $H = (V, E, w)$ with a weight function $w : E \to \mathbb{R}_+$ over its hyperedges, the *energy* of $x \in \mathbb{R}^V$ (called a potential vector) is defined as

$$Q_H(x) := \sum_{e \in E} w(e) \cdot \max_{u,v \in e} (x_u - x_v)^2.$$

The problem of minimizing $Q_H(x)$ over $x \in \mathbb{R}^V$ subject to certain constraints appears in many problems involving hypergraphs, including clustering Takai et al. (2020), semi-supervised learning Hein et al. (2013); Yadati et al. (2019); Zhang et al. (2020) and link prediction Yadati et al. (2020), from which we can see the relevance of $Q_H(x)$ in application domains. Note that when $x \in \mathbb{R}^V$ is a characteristic vector $\mathbb{1}_S \in \{0,1\}^V$ of a vertex subset $S \subset V$, the energy $Q_H(\mathbb{1}_S)$ coincides with the total weight of hyperedges cut by S, where we say that a hyperedge $e \in E$ is *cut* by S if $e \cap S \neq \emptyset$ and $e \cap (V \setminus S) \neq \emptyset$.

Since the number of hyperedges in a hypergraph of n vertices can be $\Omega(2^n)$, it is desirable to reduce the number of hyperedges in the hypergraph while (approximately) preserving the value of $Q_H(x)$ for every $x \in \mathbb{R}^V$, because this lets us speed up any algorithm involving Q_H and reduce its memory usage by running it on the smaller hypergraph instead of H itself. Soma and Yoshida Soma and Yoshida (2019) formalized this concept as spectral sparsification for hypergraphs – a natural generalization of the corresponding concept introduced by the celebrated work of Spielman and Teng (2011) for graphs. Specifically, for $0 < \epsilon < 1$, we say that a hypergraph $\widetilde{H}$ is an *ϵ-spectral-sparsifier* of a hypergraph H if $\widetilde{H}$ is a reweighted subgraph of H such that

$$\forall x \in \mathbb{R}^V, \qquad Q_{\widetilde{H}}(x) \in (1 \pm \epsilon) Q_H(x).^{1} \tag{3.1}$$

We note that when H is an ordinary graph, this definition matches that for graphs Spielman and Teng (2011). Soma and Yoshida Soma and Yoshida (2019) showed that every hypergraph H admits an ϵ-spectral-sparsifier with $\widetilde{O}(n^3/\epsilon^2)$ hyperedges,[2] and gave a polynomial-time algorithm for constructing such sparsifiers. Since then the number of hyperedges needed has been reduced to $\widetilde{O}(nr^3/\epsilon^2)$ Bansal et al. (2019), and recently to $\widetilde{O}(nr/\epsilon^{O(1)})$ Kapralov et al. (2021b), where r is the maximum size of a hyperedge in the input hypergraph H (called the *rank* of H).

The natural question whether every hypergraph admits a spectral sparsifier with $\widetilde{O}(n)$ hyperedges (for fixed ϵ) has proved to be challenging. On the one hand, it is well-known that a hypergraph is a strictly richer object than an ordinary graph (hyperedges cannot be "simulated" by edges, even approximately), and in all previous results and techniques, this extra complication introduced an extra factor of at least r. On the other hand, an exciting recent result Chen et al. (2020) has achieved sparsifiers with $\widetilde{O}(n)$ hyperedges, if one is only interested in preserving the *hypergraph cut function*, i.e., satisfying Eq. (3.1) only for all characteristic vectors $x = \mathbb{1}_S$ where $S \subseteq V$. Nevertheless, the spectral version of this question has remained open, primarily due to the non-linearity of the hypergraph Laplacian and the lack of linear-algebraic tools that have been so effective for graphs.

We settle this question by showing that a nearly linear number of hyperedges suffices.

Theorem 1. *For every hypergraph with n vertices and every $1/n \le \epsilon \le 1/2$, there exists an ϵ-spectral-sparsifier with $O(n\epsilon^{-4}\log^3 n)$ hyperedges. Moreover, one can construct such a sparsifier in time $\widetilde{O}(mr + \mathrm{poly}(n))$, where m is the number of hyperedges and r is the maximum size of a hyperedge in H.*

Bit-complexity lower bound. To complement Theorem 1, we consider lower bounds on the bit complexity of sparsifiers. Here, we consider *ϵ-cut sparsifiers*, which require that Eq. (3.1) holds only for vectors of the form $x = 1_S$. This notion actually predates spectral sparsification and was first defined by Benczúr and Karger Benczúr and Karger (2015) for graphs, and by Kogan and Krauthgamer Kogan and Krauthgamer (2015) for hypergraphs. Obviously, lower bounds for cut sparsifiers directly imply the same lower bounds also for spectral sparsifiers.

The second contribution of this work is a surprising connection between *a Ruzsa-Szemerédi (RS) graph* Ruzsa and Szemerédi (1978), which is a well-studied notion in extremal graph theory, and a lower bound on the bit complexity of a hypergraph cut sparsifier. Here, an (ordinary) graph is called a *(t, a)-RS graph* if its edge set is the union of t induced matchings of size a. Then, we show the following.

Theorem 3.1.1. *Suppose that there exists a (t, a)-Ruzsa-Szemerédi graph on n vertices with $a \ge 6000\sqrt{n \log n}$. Assume also one can compress unweighted $(t+1)$-uniform hypergraphs $G = (V, E)$ on $2n$ vertices into k*

[1] $a \in (1 \pm \epsilon) b$ is a shorthand for $(1-\epsilon) b \le a \le (1+\epsilon) b$.

[2] Throughout, $\widetilde{O}(\cdot)$ suppresses a factor of $\log^{O(1)} n$.

bits, from which $Q_G(1_S)$ can be approximated for every $S \subseteq V$ within factor $1 \pm \epsilon$, where $\epsilon = O(a/n)$. Then, $k = \Omega(at)$.

For example, by instantiating Theorem 3.1.1 with the $(n^{\Omega(1/\log\log n)}, n/3 - o(n))$-Ruzsa-Szemerédi graphs known due to Fischer et al. Fischer et al. (2002), we deduce that $\Omega(nr)$ bits are necessary to encode all the cut values of an arbitrary r-uniform hypergraph with $r = n^{O(1/\log\log n)}$, even within a fixed constant ratio $1+\epsilon$.

This lower bound near-tight due to Theorem 1. Applying this construction with fixed ϵ and $r = n^{O(1/\log\log n)}$ yields a sparsifier of G with $O(n \log n)$ hyperedges; encoding a hyperedge (including its weight, which is bounded by n^r) takes at most $O(r \log n)$ bits, and thus one can encode all the cuts of G using $O(nr \log^2 n)$ bits. It follows that our lower bound is optimal up to a lower order factor $O(\log^2 n)$. Instantiating our lower bound with the original construction of Ruzsa and Szemerédi Ruzsa and Szemerédi (1978), we can rule out the possibility of compressing the cut structure of a hypergraph with n vertices and maximum hyperedge size r with significantly less than nr space, and a polynomial scaling in the error (that is with $nr^{1-\Omega(1)}\epsilon^{-O(1)}$ space), *for any r*. See Corollaries 3.8.11 to 3.8.13 in Section 3.8 for more details.

In fact, our space lower bound for hypergraphs far exceeds the $O(n \log n/\epsilon^2)$ bits that suffices to approximately represent all the cuts of an (ordinary) graph by simply storing a cut sparsifier. We thus obtain the first provable separation between the bit complexity of approximating all the cuts of a graph vs. of a hypergraph.

Spectral sparsification of directed hypergraphs.

We also consider spectral sparsification of directed hypergraphs. Here, a hyperarc e consists of two disjoint sets, called the *head* $h(e) \subseteq V$ and the *tail* $t(e) \subseteq V$, and the *size* of the hyperarc is $|t(e)| + |h(e)|$. A directed hypergraph $G = (V, E)$ then consists of a vertex set V and a hyperarc set E. For an edge-weighted directed hypergraph $G = (V, E, w)$ and a vector $x \in \mathbb{R}^V$, the *energy* of x in G is defined as

$$Q_G(x) = \sum_{e \in E} w_e \max_{u \in t(e), v \in h(e)} (x_u - x_v)_+^2, \tag{3.2}$$

where $(a)_+ = \max\{a, 0\}$. Again, it is defined so that $Q_G(1_S)$ is the total weight of hyperarcs that are cut by S, where a hyperarc e is *cut* if $t(e) \cap S \neq \emptyset$ and $h(e) \cap (V \setminus S) \neq \emptyset$.

It is not difficult to see that a spectral sparsifier might require (in the worst-case) at least $\Omega(n^2)$ hyperarcs, even for an ordinary directed graph. Indeed, consider a balanced bipartite clique directed from one side of the bipartition towards the other. Here, every arc is the unique arc crossing some particular directed cut, and hence a sparsifier must keep all the $\Omega(n^2)$ arcs (see also Ikeda and Tanigawa (2018); Cheng et al. (2020)). However, Soma and Yoshida Soma and Yoshida (2019) showed that every directed hypergraph admits an ϵ-spectral sparsifier with $\widetilde{O}(n^3/\epsilon^2)$ hyperarcs. We tighten this gap by showing that $\widetilde{O}(n^2/\epsilon^2)$ hyperarcs are sufficient when every hyperarc is of constant size.

Theorem 3.1.2. *Given a directed hypergraph $G = (V, E)$ with maximum hyperarc size at most r such that $11r \le \sqrt{\epsilon n}$, and a value $\epsilon \le 1/2$, one can compute in polynomial time with probability $1 - o(1)$ an ϵ-spectral sparsifier of G with $O(n^2 r^3 \log^2 n/\epsilon^2)$ hyperarcs.*

We note that Theorem 3.1.2 is stated under the assumption $11r \leq \sqrt{\epsilon n}$, which is useful for our analysis for technical reasons. For larger values of r the result of Soma and Yoshida (2019) gives a better bound on the number of hyperedges in the sparsifier, and therefore this assumption is not restrictive.

3.1.1 Additional Related Work

Recall that we call $\widetilde{H} = (V, \widetilde{E}, \widetilde{w})$ an ϵ*-cut sparsifier* of $H = (V, E, w)$ if every cut weight is preserved to within a factor of $1 \pm \epsilon$. This definition matches the one for ordinary graphs introduced by Benczúr and Karger Benczúr and Karger (2015), who showed that every graph has an ϵ-cut-sparsifier with $O(n \log n/\epsilon^2)$ edges, where n is the number of vertices. For hypergraphs, Kogan and Krauthgamer Kogan and Krauthgamer (2015) gave the first construction of non-trivial cut sparsifiers, which uses $O(n(r + \log n)/\epsilon^2)$ hyperedges, where r is the maximum size of a hyperedge. They also mentioned that the results of Newman and Rabinovich Newman and Rabinovich (2013) implicitly give an ϵ-cut sparsifier with $O(n^2/\epsilon^2)$ hyperedges. Chen, Khanna, and Nagda Chen et al. (2020) improved this bound to $O(n \log n/\epsilon^2)$, which is almost tight because one needs $\Omega(n/\epsilon^2)$ edges even for ordinary graphs Andoni et al. (2016); Carlson et al. (2019).

Spielman and Teng Spielman and Teng (2011) introduced the notion of a spectral sparsifier for ordinary graphs and showed that every graph on n vertices admits an ϵ-spectral sparsifier with $O(n \log^{O(1)} n/\epsilon^2)$ edges. This bound was later improved to $O(n/\epsilon^2)$ Batson et al. (2012), which is tight Andoni et al. (2016); Carlson et al. (2019). The literature on graph sparsification is too vast to cover here, including Spielman and Teng (2011); Spielman and Srivastava (2011); Batson et al. (2012); Zhu et al. (2015); Lee and Sun (2015, 2017) and many other constructions, and we refer the reader to the surveys Vishnoi (2013); Teng (2016).

For an ordinary graph $G = (V, E, w)$, the *Laplacian* of G is the matrix $L_G = D_G - A_G$, where $D_G \in \mathbb{R}^{V \times V}$ is the diagonal (weighted) degree matrix and $A_G \in \mathbb{R}^{V \times V}$ is the adjacency matrix of G. Then, the energy Q_G, defined in Eq. (3.1), can be written also as

$$Q_G(x) = x^\top L_G x.$$

For a hypergraph $H = (V, E, w)$, it is known that we can define a (multi-valued) Laplacian operator $L_H : \mathbb{R}^V \to 2^{\mathbb{R}^V}$, so that

$$Q_H(x) = x^\top y$$

for every $x \in \mathbb{R}^V$ and $y \in L_H(x)$ Louis (2015); Chan et al. (2018); Yoshida (2019) (hence we can write $Q_H(x)$ also as $x^\top L_H(x)$ without ambiguity). Although the Laplacian operator L_H is no longer a linear operator, its mathematical property has been actively investigated Ikeda et al. (2019); Fujii et al. (2018); Ikeda et al. (2021) through the theory of monotone operators and evolution equations Komura (1967); Miyadera (1992).

Yoshida Yoshida (2016) proposed a Laplacian operator for directed graphs and used it to study structures of real-world networks. The Laplacian operators for graphs, hypergraphs, and directed graphs mentioned above were later unified and generalized as Laplacian operator for submodular transformations/submodular hypergraphs Li and Milenkovic (2018); Yoshida (2019).

3.2 Preliminaries

In this chapter, we deal with spectral sparsification of hypergraphs. For the sake of generality, we consider weighted hypergraphs denoted $H = (V, E, w)$, where V is the vertex set of size n, E is the hyperedge set of size m, and $w : E \to \mathbb{R}_+$ is the set of hyperedge weights. We will also, however, deal with ordinary graphs, that is graphs where each edge contains two vertices exactly. In order to distinguish clearly between graphs and hypergraphs, we will typically denote graphs as $G = (V, F, z)$, where V is the vertex set, F is the edge set, and $z : F \to \mathbb{R}_+$ is the set of edge weights. In general we will use f and g to denote ordinary edges, while reserving e to denote hyperedges.

For simplicity *all graphs and hypergraphs we consider in this chapter will be connected.*

3.2.1 Spectral Graph Theory

Definition 3.2.1. *The Laplacian of a weighted graph $G = (V, F, z)$ is defined as the matrix $L_G \in \mathbb{R}^{V \times V}$ such that*

$$(L_G)_{uv} = \begin{cases} d(u) & \text{if } u = v, \\ -z(u,v) & \text{if } (u,v) \in F, \\ 0 & \text{otherwise.} \end{cases}$$

Here $d(u)$ denotes the weighted degree of u, that is the sum of all weights of incident edges. Thus L_G is a positive semidefinite matrix, and its quadratic form can be written as

$$x^\top L_G x = \sum_{(u,v) \in F} z(u,v) \cdot (x_u - x_v)^2.$$

The spectral sparsifier of G is defined as a reweighted subgraph which closely approximates the quadratic form of the Laplacian on every possible vector.

Definition 3.2.2. *Let $G = (V, F, z)$ be a weighted ordinary graph. Let $\widetilde{G} = (V, \widetilde{F}, \widetilde{z})$ be a reweighted subgraph of G, defined by $\widetilde{z} : F \to \mathbb{R}_+$, where $\widetilde{F} = \{f \in F \mid \widetilde{z}(f) > 0\}$. For $\epsilon > 0$, $\widetilde{G}$ is an ϵ-spectral sparsifier of G if for every $x \in \mathbb{R}^V$*

$$x^\top L_{\widetilde{G}} x \in (1 \pm \epsilon) \cdot x^\top L_G x.$$

The quadratic form of the graph Laplacian from Definition 3.2.1 can be generalized to hypergraphs. Although this generalization is highly non-linear, we still refer to it as the "quadratic form" of the hypergraph.

Definition 3.2.3. *The quadratic form (or sometimes* energy*) of a hypergraph $H = (V, E, w)$ is defined on the input vector $x \in \mathbb{R}^V$ as*

$$Q_H(x) = \sum_{e \in E} w(e) \cdot \max_{u,v \in e} (x_u - x_v)^2.$$

Consequently, we may also define the concept of spectral sparsification in hypergraphs, analogously to Definition 3.2.2:

Definition 3.2.4. *Let $H = (V, E, w)$ be a weighted hypergraph. Let $\widetilde{H} = (V, \widetilde{E}, \widetilde{w})$ be a reweighted subgraph of H, defined by $\widetilde{w} : E \to \mathbb{R}_+$, where $\widetilde{E} = \{e \in E \mid \widetilde{w}(e) > 0\}$. For $\epsilon > 0$, $\widetilde{H}$ is an ϵ-spectral sparsifier of H if for every $x \in \mathbb{R}^V$*

$$Q_{\widetilde{H}}(x) \in (1 \pm \epsilon) \cdot Q_H(x).$$

3.2.2 Effective Resistance

Definition 3.2.5. *Let $G = (V, F, z)$ be a weighted ordinary graph. The effective resistance of a pair of vertices (u, v) is defined as*

$$R_G(u, v) = (\chi_u - \chi_v)^\top L_G^+ (\chi_u - \chi_v).$$

Here $\chi_u \in \mathbb{R}^V$ is the vector with all zeros, and a single 1 at the coordinate corresponding to u. L_G^+ is the Moore-Penrose pseudo-inverse of L_G, which is positive semidefinite.

We may write $R(u, v)$ in cases where G is clear from context.

We will often use the notation $R_G(f) = R_G(u, v)$ where $f = (u, v)$ is an edge. It is important to note, however, that effective resistance is a function of the vertex pair, not the edge, and does not depend directly on the weight of f.

We now state several well-known and useful facts about effective resistance.

Fact 3.2.6. *The effective resistance of an edge (u, v) is alternatively defined as*

$$R_G(u, v) = \max_{x \in \mathbb{R}^V} \frac{(x_u - x_v)^2}{x^\top L x}.$$

Fact 3.2.7. *Effective resistance constitutes a metric on V.*

Fact 3.2.8. *For any weighted graph $G = (V, F, z)$ and any edge $f \in F$ we have $z(f) \cdot R_G(f) \le 1$, with equality if and only if f is a bridge.*

Fact 3.2.9. *For any weighted graph $G = (V, F, z)$ we have*

$$\sum_{f \in F} z(f) \cdot R_G(f) = n - 1.$$

3.2.3 Chernoff Bound

Theorem 3.2.10 (Chernoff bound, see for example Alon and Spencer (2008))**.** *Let $Z_1, Z_2, \ldots, Z_k$ be independent random variables in the range $[0, a]$. Furthermore, let $\sum Z_i = Z$ and let $\mu \ge \mathbb{E}(Z)$. Then for any $\delta \in (0, 1)$,*

$$\mathbb{P}\left(|Z - \mathbb{E}(Z)| \ge \delta\mu\right) \le 2\exp\left(-\frac{\delta^2 \mu}{3a}\right).$$

3.3 Technical Overview

3.3.1 Analyzing Ordinary Graphs

The sparsification of ordinary graphs is a highly studied topic, with several techniques proposed for the construction of spectral sparsifiers throughout the years Spielman and Teng (2011); Spielman and Srivastava (2011); Batson et al. (2012); Zhu et al. (2015); Lee and Sun (2015, 2017). However, the analysis of spectral sparsifiers always relies heavily on the linear nature of the graph Laplacian, e.g., using matrix concentration results such as matrix Bernstein Tropp (2011) or the work of Rudelson and Vershynin (2007). This presents a significant problem when attempting to generalize these techniques to the highly non-linear setting of hypergraph spectral sparsification. Indeed, all previous results lose at least a factor of r due to this obstacle. We therefore dedicate the entirety of our first technical section (Section 3.4) to presenting a new proof of the existence of nearly linear spectral sparsifiers for ordinary graphs. We use the algorithm from Spielman and Srivastava (2011), which constructs a sparsifier $\widetilde{G}$ by sampling each edge with probability proportional to its effective resistance. However, our proof avoids using matrix concentration inequalities, and instead relies on a more direct chaining technique for proving the concentration of $x^\top L_{\widetilde{G}} x$ around its expectation, i.e. $x^\top L_G x$, for all x simultaneously. To our knowledge, this is the first nearly-optimal direct analysis of spectral sparsification through effective resistance sampling. It will also be the basis of our main result, as we adapt it to the hypergraph setting in Sections 3.5 and 3.6.

More formally, for an input graph $G = (V, F, z)$, we define $\widetilde{G}$ as the result of sampling each edge f of G independently with probability $p(f) \approx z(f) \cdot R_G(f)$, and setting its weight to $\widetilde{z}(f) = z(f)/p(f)$. Our aim is then to prove

$$x^\top L_{\widetilde{G}} x \approx x^\top L_G x \tag{3.3}$$

simultaneously for all $x \in \mathbb{R}^V$. For simplicity we assume that $x^\top L_G x = 1$. Eq. (3.3) is in fact the concentration of a random variable around its expectation, and so we can use Chernoff bound to prove it for any specific x. Our plan is then to use a combination of Chernoff and union bounds to prove it for all possible x. Since x can take any value in $\mathbb{R}^V$ we must discretize it to some ϵ-net while retaining a good approximation to its quadratic form, i.e. $x^\top L_G x$.

Let us take a closer look at the application of Chernoff bound to Eq. (3.3): $x^\top L_{\widetilde{G}} x$ is the sum of the independent random variables $\widetilde{z}(u,v) \cdot (x_u - x_v)^2$ for $(u,v) \in F$; hence, by Theorem 3.2.10, the strength of the bound depends crucially on the upper bound a on values that each random individual random variable can possibly attain. The maximum value of $\widetilde{z}(u,v) \cdot (x_u - x_v)^2$ is attained when (u,v) is sampled in $\widetilde{G}$, in which case it is $\approx (x_u - x_v)^2 / R_G(u,v)$. Thus

$$\mathbb{P}\left(x^\top L_{\widetilde{G}} x \not\approx x^\top L_G x\right) \lessapprox \exp\left(-\frac{1}{\max_{(u,v)\in F} (x_u - x_v)^2 / R_G(u,v)}\right).$$

This upper bound can be as bad as $\exp(-\widetilde{O}(1))$ and is far too crude for our purposes—no sufficiently sparse rounding scheme (i.e., discretization) exists for x. We turn to the technique of chaining—the use of progressively finer and finer rounding schemes.

As seen above, the strength of our Chernoff bound depends primarily on the quantity $(x_u - x_v)^2 / R_G(u,v)$ for each edge (u,v), which we call the “power” of the edge. Therefore, it makes sense to partition the edges of G into a logarithmic number of classes based on their power, that is F_i contains edges (u,v) for which $(x_u - x_v)^2 \approx 2^{-i} \cdot R_G(u,v)$. When focusing only on the subgraphs $G(F_i)$ induced by F_i, we get the more fine-tuned Chernoff bound

$$\mathbb{P}\left(x^\top L_{\widetilde{G}(F_i)} x \not\approx x^\top L_{G(F_i)} x\right) \lessapprox \exp\left(-\frac{1}{\max_{(u,v)\in F_i} (x_u - x_v)^2 / R_G(u,v)}\right) \lessapprox \exp\left(-2^i\right).$$

We thus have the task of proposing a rounding scheme $\varphi_i : \mathbb{R}^V \to \mathbb{R}^V$ specially for each class F_i such that

- the image of φ_i is a finite set of size at most $\approx \exp\left(2^i\right)$,
- the rounding approximately preserves the quantity $(x_u - x_v)^2$ for $(u,v) \in F_i$.

To gain more intuition on what such a rounding scheme must look like, we draw inspiration from the idea of resistive embedding from Spielman and Srivastava (2011). We map the edges in F_i, as well as our potential vector x, into vectors in $\mathbb{R}^n$ in such a way that all the relevant quantities arise as norms or scalar products:

$$(u,v) \mapsto \underline{a_{u,v}} = \frac{L_G^{+/2}(\chi_u - \chi_v)}{\left\|L_G^{+/2}(\chi_u - \chi_v)\right\|},$$
$$x \mapsto \underline{y_x} = L_G^{1/2} x.$$

Notice that both $\underline{a_{u,v}}$ and $\underline{y_x}$ are normalized (since $x^\top L_G x$=1). Furthermore, the crucial quantity, the power of the edge (u,v) arises as the square of a scalar product:

$$\langle \underline{a_{u,v}}, \underline{y_x} \rangle^2 = \frac{(x^\top(\chi_u - \chi_v))^2}{(\chi_u - \chi_v)^\top L_G^+ (\chi_u - \chi_v)} = \frac{(x_u - x_v)^2}{R_G(u,v)}.$$

Thus we are interested in rounding $\underline{y_x}$ in a way that preserves $\langle \underline{a_{u,v}}, \underline{y_x} \rangle^2$ up to small multiplicative error *in all cases where it was* $\approx 2^{-i}$ *to begin with.* Thus, it suffices to guarantee an additive error of at most $\lessapprox 2^{-i}$ in our rounding scheme. This is the known problem of "compression of approximate inner products" and has been previously studied; Alon and Klartag (2017) guarantees a rounding scheme whose image is of size at most $\approx \exp\left(2^i\right)$. This can be translated into a rounding scheme for $x \in \mathbb{R}^V$, with the same image-size, exactly as desired (see Lemma 3.4.2).

With the desired rounding scheme in hand, we can now use a combination of Chernoff and union bounds to prove that for all x simultaneously

$$x^\top L_{\widetilde{G}(F_i)} x \approx x^\top L_{G(F_i)} x.$$

Summing this over all edge-classes gives us Eq. (3.3).

For the detailed proof, which is considerably more complicated than the above sketch, see the proof of Theorem 3.4.1 in Section 3.4.

3.3.2 Extension to Hypergraphs

To adapt the previous argument to the hypergraph setting, we use the idea of balanced weight assignments from Chen et al. (2020). Essentially, we construct an ordinary graph $G = (V, F, z)$ to accompany our input hypergraph $H = (V, E, w)$ by replacing each hyperedge e with a clique F_e over the vertices in e. However, unlike in some previous works on hypergraph sparsification, the clique F_e is not assigned weights uniformly, but instead the weight is carefully distributed among the edges. Intuitively, all the weight is shifted onto the most "important" edges. In the case of Chen et al. (2020), the measure of importance was "strength", a quantity relevant to cut sparsification, while in our case it is effective resistance.

More formally, a weighting assignment z of the cliques is considered γ-balanced if for all $e \in E$

- $\sum_{f\in F_e} z(f) = w(e)$,
- and
$$\gamma \cdot \min_{g\in F_e\colon z(g)>0} R_G(g) \geq \max_{f\in F_e} R_G(f).$$

In words, all but the zero-weight edges of F_e have approximately the same effective resistance. This allows hyperedge e to inherit this effective-resistance value as its importance when sampling hyperedges. Our task is now to prove the existence of balanced weight assignments for all hypergraphs, and then to adapt the proof of Section 3.4.

Finding balanced weight assignments. In Chen et al. (2020), balanced weight assignments are constructed through the following intuitive process: Find a pair of edges violating the second constraint, that is $f, g \in F_e$ where $z(g) \neq 0$ and f has significantly higher importance than g. Then shift weight from g to f; this alleviates the constraint violation either because the importances of g and f become more similar, or simply because the weight of g decreases to 0. We call this resolving the imbalance of f and g. Chen et al. (2020) strings together such steps, carefully ordered and discretized, to eventually produce a balanced weight assignment of the input hypergraph.

However, their analysis relies heavily on a certain lemma about how "strength" (their measure of edge importance) behaves under weight updates. Lemma 6 of Chen et al. (2020) states that altering the weight of an edge f, will not affect edges of significantly greater "strength" than f. This is not the case for effective resistances. It is easy to construct scenarios to the contrary; even ones in which altering the weight of edges of low resistance can increase the maximum effective resistance in the graph.

Thus the analysis of Chen et al. (2020) does not extend to our setting. Instead we use a potential function argument to say that we make irreversible progress whenever we resolve the imbalance of two edges f and g. Our choice of potential function is surprising, and is one of the main technical contributions of this chapter. We define the spanning tree potential (or ST-potential) of a connected weighted ordinary graph $G = (V, F, z)$, denoted $\Psi(G)$. For edge weights that equal 1 uniformly (that is for unweighted graphs) it is simply the logarithm of the number of distinct spanning trees in G. In weighted graphs it is generalized to

$$\Psi(G) = \log\left(\sum_{T\in\mathbb{T}}\prod_{f\in T} z(f)\right),$$

where $\mathbb{T}$ denotes the set of all spanning trees in G. Due to the relationship between spanning tree sampling and effective resistances (see for example Lovász (1993)) we can prove a crucial update formula for $\Psi(G)$: if an edge f has its weight changed by $\lambda \in \mathbb{R}$, the ST-potential increases by $\log(1+\lambda\cdot R(f))$. Since whenever we resolve the imbalance of a pair of edges, we shift weight from the edge of lower effective resistance to that of higher effective resistance, this allows us to argue that the ST-potential always increases throughout the process, which eventually terminates in a balanced weight assignment (see Algorithm 3 and Theorem 3.5.8).

This proves the existence of balanced weight assignments, which suffices to show the existence of nearly linear size spectral sparsifiers for all hypergraphs. However, to improve running time (from exponential to polynomial in the input size), we introduce the novel concept of *approximate* balanced weight assignments, by slightly relaxing the definition. These are still sufficient to aid in constructing spectral sparsifiers, and are faster to construct using Algorithm 4.

For more details on the ST-potential, as well as the construction of balanced weight assignments see Section 3.5.

Using balanced weight assignments to construct hypergraph spectral sparsifiers. Given a hypergraph $H=(V,E,w)$ and its balanced weight assignment $G=(V,F,z)$ we assign importance to each hyperedge proportionally to the maximum effective resistance in F_e (the clique corresponding to e). Thus we perform importance sampling, which samples each hyperedge independently with probability $p(e) \approx w(e)\cdot\max_{f\in F_e} R_G(f)$.

The broad strokes of the hypergraph proof in Section 3.6 proceed very similarly to those of the proof for ordinary graphs in Section 3.4. However, numerous details need to be figured out in order to bridge the gap between the two settings. It is interesting to note that our rounding scheme is exactly the same as in Section 3.4, to the point of even being defined in terms of G, not H. (Indeed it is impossible to define such a rounding scheme directly in terms of H; Lemma 3.4.2 relies heavily on the linear nature of the ordinary graph Laplacian.) Nevertheless, we manage to extend the approximation guarantee of the rounding scheme from edges to hyperedges (see Claim 3.6.3).

For the detailed analysis of hypergraph spectral sparsification through effective resistance-based importance sampling, see Section 3.6.

3.3.3 Speed-Up

Using a results of Sections 3.5 and 3.6 we can put together a polynomial time algorithm for spectral sparsification of hypergraphs. Simply run Algorithm 4 to produce an approximate balanced weight assignment, and then use importance sampling (Algorithm 5). The bottleneck of this procedure is constructing the weight assignment, which takes time $O(m \cdot \text{poly}(n))$. (Given the weight assignment, it is trivial to implement importance sampling).

In Section 3.7 we reduce this to the nearly optimal $\widetilde{O}(mr + \text{poly}(n))$. (Note that $O(mr)$ is the size of the input.) Our first step is the common trick of using a faster sparsification algorithm, but one which produces a larger output, to preprocess the input hypergraph. We use the algorithm of Bansal et al. (2019) which – with small modifications – can be made to run in the desired $\widetilde{O}(mr + \text{poly}(n))$ time. The resulting hypergraph has only polynomially many hyperedges (in n); however, the aspect ratio of edge weights (that is the ratio between the largest and smallest edge weights) can naturally be exponential in n.

Unfortunately, Algorithm 4 scales linearly in the aspect ratio of edge weights (see Theorem 3.5.10) and so we propose another algorithm for finding a balanced weight assignment – one specifically designed for the setting when the input graph is polynomially sparse, but has exponential aspect ratio.

Suppose our input hypergraph, $H = (V, E, w)$ has edge weights in the range $[1, \exp(n)]$. We then divide hyperedges into weight categories such as $E_i = \{e \in E | w(e) \in [n^{10(i-1)}, n^{10i})\}$. We then bisect H into two hypergraphs H_1 and H_2, where H_1 contains all hyperedges in odd numbered categories, and H_2 all those in even numbered categories. This results in hyergraphs (H_1 and H_2) where hyperedges fall into extremely well-separated categories; so extremely in fact, that the weight of a hyperedge in a higher category (for example $e \in E_i \subseteq H_1$) has higher weight than all hyperedges of all lower categories *combined*, that is

$$w(e) \gg \sum_{e' \in E_{<i} \cap H_1} w(e').$$

We use this property to independently find weight assignments on H_1 and H_2. Informally, we go through the categories of hyperedges, from heaviest to lightest, resolving all instances of imbalance. We never return to a category once we moved on, and we prove that no amount of changes to the weight assignment of lower categories can disrupt the balance of a higher category, due to the huge discrepency in weights. For a more detailed and formal argument see Section 3.7.2.

3.3.4 Lower Bounds

The most common method for approximating the Laplacian of a (hyper)graph is to take a weighted subset of the original (hyper)edges. While asympotically optimal for graphs Andoni et al. (2016); Carlson et al. (2019), this method has obvious limitations as a data structure: it is not hard to come up with an example where $\Omega(n)$ hyperedges are required even for the sparsifier to be connected, and if the input hypergraph is r-uniform, this translates into $\Omega(nr \log n)$ bit complexity, a linear loss in the arity r of the hypergraph. It is therefore natural to ask whether there are more efficient ways of storing a spectral approximation to a hypergraph. As concrete example, we could permit the inclusion of hyperedges not in the original hypergraph – could this or another scheme lead to a data structure that can approximate the spectral structure of a hypergraph using $\widetilde{O}(n)$ space, avoiding a dependence on r?

In Section 3.8, we study this question in full generality:

Is it possible to compress a hypergraph into a $o(n\cdot r)$ size data structure that can approximate the energy $Q_G(x)$ (defined in Definition 3.2.3) simultaneously for all $x\in\mathbb{R}^V$?

In Section 3.8, we show a space lower bound of $\Omega(nr)$ for sparsifying a hypergraph on n vertices with maximum hyperedge-size r[3]. In fact, our lower bound applies even to the weaker notion of cut sparsification (where one only wants to approximate $Q_G(x)$ for all $x\in\{0,1\}^V$), and is tight by the recent result of Chen et al. (2020), who gave a sampling-based cut sparsification algorithm that produces hypergraph sparsifiers with $O(n\log^{O(1)} n)$ hyperedge. In what follows we give an outline of our lower bound.

We start by formally defining the data structure for approximating the cut structure of a hypergraph that we prove a lower bound for. A *hypergraph cut sparsification scheme (HCSS)* is an algorithm for compressing the cut structure of a hypergraph such that queries on the size of cuts can be answered within a small multiplicative error:

Definition 3.3.1. *Let $\mathfrak{H}(n,r)$ be the set of hypergraphs on a vertex set $[n]$ with each hyperedge having size at most r. A pair of functions* $\textsc{Sparsify}:\mathfrak{H}(n,r)\to\{0,1\}^k$ *and* $\textsc{Cut}:\{0,1\}^k\times 2^{[n]}\to\mathbb{N}$ *is said to be an (n,r,k,ε)-HCSS if for all inputs $G=(V,E)\in\mathfrak{H}(n,r)$ the following holds.*

- *For every query* $S\in 2^{[n]}$, $\left|\textsc{Cut}(\textsc{Sparsify}(G),S)-|E(S,\overline{S})|\right|\le\varepsilon\cdot|E(S,\overline{S})|$.

To argue a lower bound on the space requirement (parameter k above), we use a reduction to string compression. It is known that $\{0,1\}$-strings of length ℓ cannot be significantly compressed to a small space data structure that allows even extremely crude additive approximations to subset sum queries — see, e.g., the LP decoding paper of Dinur and Nissim (2003) (here we only need a lower bound for computationally unbounded adversaries), or Section 3.8.1. We manage to encode a $\{0,1\}$-string of length ℓ into the cut structure of a hypergraph H with *fewer hyperedges than* ℓ — a testament to the higher complexity of hypergraph cut structures, as opposed to the cut structures of ordinary graphs.

Our string encoding construction utilizes Ruzsa-Szemerédi graphs. These are (ordinary) graphs whose edge-sets are the union of *induced* matchings. Our construction works generally on any Ruzsa-Szemerédi graphs and as a result we get several lower bounds in various parameter regimes (values of the hyperedge arity r and the precision parameter ϵ) based on the specific Ruzsa-Szemerédi graph constructions we choose to utilize. In particular, for the setting where $r=n^{O(1/\log\log n)}$ we are able to conclude that any hypergraph cut sparsification scheme requires $\Omega(rn)$ bits of space even for constant ϵ, matching the upper bound of Chen et al. (2020) to within logarithmic factors. For larger r we get a lower bound of $n^{1-o(1)}r$ bits of space for $\epsilon=n^{-o(1)}$. The latter in particular rules out the possibility of an ϵ-sparsifier that can be described with asymptotically fewer than $(\epsilon^{-1})^{O(1)}nr$ bits of space.

Here we briefly describe how we encode strings into hypergraphs generated from Ruzsa-Szemerédi graphs. Let G be a bipartite Ruzsa-Szemerédi-graph (with bipartition $P\cup Q$) composed of t induced

[3]With some limits on the range of r. For more formal statements of our results see Section 3.8.2.

matchings of size a each. We can then use the $a \cdot t$ edges of the graph to encode a string s of length $\ell = at$: simply order the edges of G and remove any edges corresponding to 0 coordinates in s, while keeping edges corresponding to 1's. This graph — which we call G_s — already encodes s when taken as a whole. However, its cut structure is not sufficient for decoding it. For that we need to turn G_s into a hypergraph H_s as follows: For each vertex u on one side of the bipartition, say P, we combine all edges adjacent on u into one hyperedge containing $\{u\} \cup \Gamma(u)$. This means that each hyperedge will have only a single vertex in P, but many vertices in Q (see Fig. 3.1).

To decode the original string s from the cut structure of H, we must be able to answer subset sum queries $q \subseteq [at]$, that is return how many 1-coordinates s has, restricted to q. (For more details see the definition of string compression – Definition 3.8.1 in Section 3.8.1.) To do this, consider each induced matching one at a time and decode s restricted to the corresponding coordinates. We measure the size of a carefully chosen cut in H_s. Consider Fig. 3.1: We restrict our view to a single matching M_j supported on P_j and Q_j in the two sides of the bipartition. Suppose for simplicity that q is entirely contained in this matching, and we are interested in the Hamming-weight of s restricted to a subset of coordinates q. To create our cut, in the top half of the hypergraph (P), we take the endpoints of edges corresponding to q – we call this set A. In the bottom half (Q), we take everything except for Q_j. The cut, which we call S, is depicted in red in Fig. 3.1.

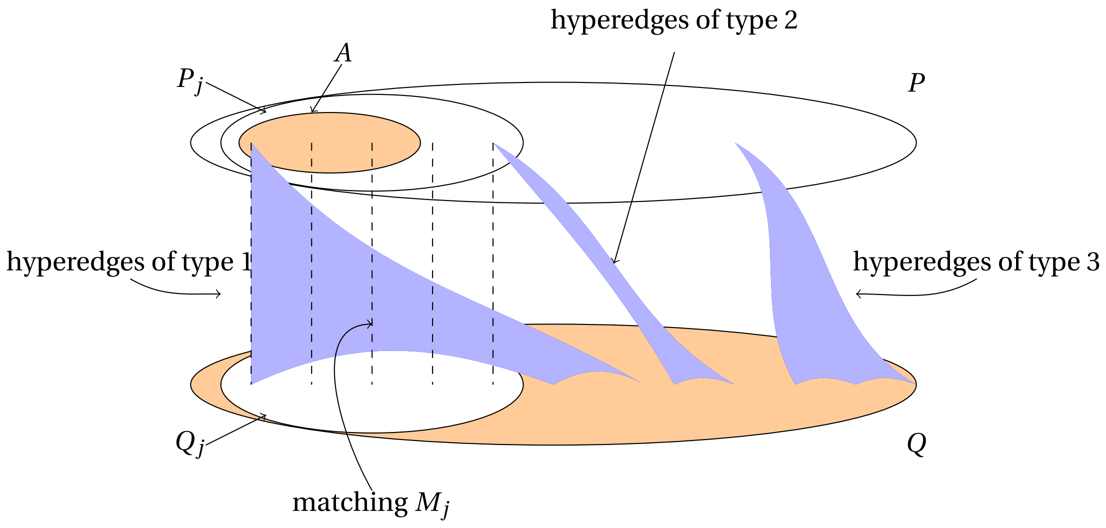


Figure 3.1 – Illustration of the decoding process. One side of the cut S is depicted in orange.

Informally, the crux of the decoding is the observation that the number of hyperedges crossing from A to Q_j is exactly the quantity we want to approximate. Indeed, consider a coordinate in q. If it has value 1 in s, the corresponding hyperedge crosses from A to Q_j, thus crossing the cut S. If however this coordinate is 0 in s, the corresponding hyperedge does not cross to Q_j, thus not crossing the cut. These types of hyperedges are denoted by 1 in Fig. 3.1.

Unfortunately, there are more hyperedges crossing S, adding noise to our measurement of s. One might hope to prove that the noise is small, i.e., can be attributed to measurement error, but this is not the case. Instead, we show that while this noise is not small, it is predictable enough to subtract accurately without knowing s. Hyperedges denoted 2 in Fig. 3.1 cross from $P_j \setminus A$ to $Q \setminus Q_j$. Here we observe that nearly all hyperedges from $P_j \setminus A$ do in fact cross the cut, for almost all choices of s. Hyperedges denoted 3 in Fig. 3.1 cross from $P \setminus P_j$ to $Q \setminus Q_j$. Here we cannot say much about the quantity of such hyperedges crossing the cut. However, we observe that this quantity does not depend on q, and therefore we can use Chernoff bounds (Theorem B.0.1) to prove that it concentrates around its expectation with high probability *over* s. This allows us to predict and subtract the noise caused by type 3 hyperedges, for whatever instance of Ruzsa-Szemerédi-graph we use (see the proof of Theorem 3.8.9).

Ultimately, we show that efficient cut sparsification for such hypergraphs would result in an equally efficient compression of $\{0,1\}$-strings, which implies our lower bounds. For more details see Section 3.8.

3.3.5 Spectral Sparsification of Directed Hypergraphs

In Section 3.9, we apply our discretization technique from to the spectral sparsification of directed hypergraphs. As a testiment to the versitility of this technique, we are able to produce an $O(n^2 r^3 \log^2 n/\epsilon^2)$-sized ϵ-spectral sparsifier. This is a factor n better than the previous state of the art by Soma and Yoshida (2019), and nearly optimal in the setting where r is constant.

The broad arc of the proof is very similar to that of Section 3.4: We construct our sparsifier using importance sampling. We then divide the set of hyperarcs into a logarithmic number of categories, E_i. For each category separately, we show using discretization that the energy of the proposed sparsifier approximates the energy of the input hypergraph with respect to *all* $x \in \mathbb{R}^V$ simultaneously with high probability.

However, the details of each of these steps differ from their corresponding step in Section 3.4. Here we mention only a few key differences. Instead of looking at degrees or expansion, we define a novel quantity characterizing each hyperarc we call its *overlap*. Intuitively, this denotes the highest density of an induced subgraph in which the paericular hyperarc resides. We then sample each hyperarc with probability inverse proportional to its overlap. We show that this produces a sufficiently small sparsifier with high probability (see Lemma 3.9.2).

Perhaps the most crucial departure from Section 3.4 occurs during the discretization step when proving $Q_x(E_i) = \widetilde{Q}_x(E_i)$. Instead of discretizing the vector $x \in \mathbb{R}^V$, we discretize the derived vector of energies on the hyperarcs, that is $Q_x \in \mathbb{R}^E$. So for each x and i we define a vector $Q_x^{(i)}$ — from a finite set of possibilities — such that, informally

$$Q_x(E_i) \cong Q_x^{(i)}(E_i) \cong \widetilde{Q}_x^{(i)}(E_i) \cong \widetilde{Q}_x(E_i).$$

For more details on the definition of $Q_x^{(i)}$, see the proof of Lemma 3.9.7. This additional trick is necessary; we do not know of a way to make the discretization argument work by rounding x itself.

For more details on the construction of directed hypergraph sparsifiers and their analysis see Section 3.9.

3.4 Warm-Up: Ordinary Graphs

We begin by reproving the famous theorem of Spielman and Srivastava Spielman and Srivastava (2011), which states that sampling edges of a graph with probability proportional to their effective resistance (and then reweighting appropriately) results in a spectral sparsifier with high probability. We prove a somewhat weaker version of the theorem, where we oversample by an $O(\epsilon^{-4}\log^3 n)$ factor, as opposed to the $\epsilon^{-2}\log n$ factor in the original. Another slight difference is that our version samples every edge independently, instead of sampling a predetermined number of edges with replacement in Spielman and Srivastava (2011). More recent proofs of the theorem of Spielman and Srivastava (2011) that use the matrix Bernstein inequality as opposed to Rudelson and Vershynin (2007) also use the same distribution as ours.

Theorem 3.4.1 (A slightly weaker version of Spielman and Srivastava (2011))**.** *Let $G = (V, F, z)$ be a weighted ordinary graph with n vertices and let $1/n \le \epsilon \le 1/2$. For every edge $f \in F$, let $p(f) = \min(1, \lambda \cdot z(f) \cdot R_G(f))$ for a sufficiently large factor $\lambda = \Theta(\epsilon^{-4}\log^3 n)$. Sample each edge $f \in F$ independently with probability $p(f)$, and give it weight $\widetilde{z}(f) = z(f)/p(f)$ if sampled. The resulting graph, $\widetilde{G} = (V, \widetilde{F}, \widetilde{z})$ is an ϵ-spectral sparsifier of G with probability at least $1 - O(\log n/n)$.*

The original proof of this theorem used a concentration bound for matrices Rudelson and Vershynin (2007) (later simplified to use the matrix Bernstein inequality) to prove that $x^\top L_{\widetilde{G}} x$ is close to its expectation simultaneously for all $x \in \mathbb{R}^n$, as required by Definition 3.2.2. This type of argument is difficult to adapt to hypergraph sparsification, because the extension of quadratic forms to hypergraphs is highly non-linear. We thus present an alternative proof that uses more primitive techniques to bypass the reliance on linear algebra.

Proof of Theorem 3.4.1: By Definition 3.2.2, we must prove that for every $x \in \mathbb{R}^V$,

$$x^\top L_{\widetilde{G}} x \in (1 \pm \epsilon) \cdot x^\top L_G x. \tag{3.4}$$

We may assume without loss of generality that $x^\top L_G x = 1$. We denote the set of vectors x where this is satisfied as $S^G \subseteq \mathbb{R}^V$. Furthermore, we simplify notation by denoting L_G as L, and $L_{\widetilde{G}}$ as $\widetilde{L}$. Moreover, for any subset of edges $F' \subseteq F$, we denote the Laplacian of the subgraph of G corresponding to F' by $L_{F'}$, and similarly for the subgraph of $\widetilde{G}$ by $\widetilde{L}_{F'}$.

It is clear from the construction of $\widetilde{G}$ that

$$\mathbb{E}\left(x^\top \widetilde{L} x\right) = x^\top L x.$$

Therefore, we are in effect trying to prove the concentration of a random variable around its expectation in Eq. (3.4). Indeed, for any specific x, Eq. (3.4) holds with high probability by Chernoff bound (Theorem 3.2.10). (One can consider $x^\top L x$ as the sum of independent random variables of the form $\widetilde{z}(u,v) \cdot (x_u - x_v)^2$.)

In order to prove the concentration for all $x \in S^G$ simultaneously, we employ a net argument, where we "round" x to some vector from a finite set and apply a union bound on the rounded vectors. However, our rounding scheme is progressive and has $O(\log n)$ "levels" with increasingly finer resolution. Each x

will then determine a partition of the edges into levels, and we will prove concentration for each rounded vector and each level (subset of edges), and then apply a union bound over all these choices.

The existence of these rounding functions is guaranteed by the following lemma, which we will prove in Section 3.4.1.

Lemma 3.4.2. *Let $G = (V, F, z)$ be a connected weighted graph. Then for every $i \in \mathbb{N}$ there exists a rounding function*

$$\varphi_i : S^G \to \mathbb{R}^V$$

such that for all $x \in S^G$, denoting $x^{(i)} := \varphi_i(x)$, we have:

1. *The image of φ_i is a finite set of cardinality $|\varphi_i(S^G)| \le \exp\left(800C \log n \cdot 2^i/\epsilon^2\right)$, where $C > 0$ is the absolute constant from Theorem 3.4.5.*
2. *For every edge $f = (u, v) \in F$ such that $\max\left((x_u - x_v)^2, (x_u^{(i)} - x_v^{(i)})^2\right) \ge 2^{-i} \cdot R_G(f)$,*

$$(x_u - x_v)^2 \in \left(1 \pm \frac{\epsilon}{7}\right) \cdot (x_u^{(i)} - x_v^{(i)})^2.$$

The second guarantee of Lemma 3.4.2 can be expressed in terms of the Laplacian of a single edge, resulting in the following corollary.

Corollary 3.4.3. *For a rounding function φ satisfying the guarantees of Lemma 3.4.2, and an edge $f = (u, v) \in F$ such that $\max\left((x_u - x_v)^2, (x_u^{(i)} - x_v^{(i)})^2\right) \ge 2^{-i} \cdot R_G(f)$,*

$$x^\top L_{\{f\}} x \in \left(1 \pm \frac{\epsilon}{7}\right) \cdot x^{(i)\top} L_{\{f\}} x^{(i)}.$$

Let us take a sequence of the rounding functions φ_i guaranteed by Lemma 3.4.2 for $i = 1, \dots, I := \log_2(7n/\epsilon) \le 3 \log n$. For each $x \in S^G$, it yields a sequence of rounded vectors $x^{(i)} = \varphi_i(x)$ for $i = 1, \dots, I$. Furthermore, we use $x^{(i)}$ to define the subset of edges $F_i' \subseteq F$ by

$$F_i' := \left\{ f = (u, v) \in F \;\middle|\; \left(x_u^{(i)} - x_v^{(i)}\right)^2 \ge 2^{-i} \cdot R_G(f) \right\}.$$

That is, the second guarantee of Lemma 3.4.2 holds for φ_i on edges in F_i'. Finally, we use $\{F_i'\}_i$ to partition F as follows. Let the base case be $F_0 = F_0' := \{f \in F \mid p(f) = 1\}$, where we recall that $p(f) = \min(1, \lambda \cdot z(f) \cdot R_G(f))$. For each $i \in [I]$, let $F_i := F_i' \setminus \bigcup_{j=0}^{i-1} F_j'$, and finally let $F_{I+1} = F \setminus \bigcup_{i=0}^{I} F_i'$.

Thus we have partitioned F in such a way that the second guarantee of Lemma 3.4.2 applies to edges in F_i, with respect to φ_i. Furthermore, F_i are defined in terms of $x^{(i)}$ (and $x^{(j)}$ for $j < i$) instead of x, so that the number of possible sets F_i is finite, and bounded thanks to the first guarantee of Lemma 3.4.2.

We establish the following claim for later use.

Claim 3.4.4. *For all $i \in [I]$ and $f = (u, v) \in F_i$, we have*

$$(x_u^{(i)} - x_v^{(i)})^2 \le 3 \cdot 2^{-i} \cdot R_G(f).$$

Proof. The second guarantee of Lemma 3.4.2 for φ_i applies to f, and thus $(x_u^{(i)} - x_v^{(i)})^2 \le (x_u - x_v)^2 \cdot (1 - \epsilon/7)^{-1}$.

Consider first the case $i = 1$. By Fact 3.2.6 and since $x \in S^G$, we have $(x_u - x_v)^2 \le R_G(f) \cdot x^\top L x = R_G(f)$, and we indeed get $(x_u^{(i)} - x_v^{(i)})^2 \le R_G(f) \cdot (1-\epsilon/7)^{-1} \le 3 \cdot 2^{-1} \cdot R_G(f)$.

Now consider $i > 1$, and suppose towards contradiction that $(x_u^{(i)} - x_v^{(i)})^2 > 3 \cdot 2^{-i} \cdot R_G(f)$. Notice that the second guarantee of Lemma 3.4.2 also applies to f for φ_{i-1}, and thus $(x_u^{(i-1)} - x_v^{(i-1)})^2 \ge (x_u^{(i)} - x_v^{(i)})^2 \cdot (1+\epsilon/7)^{-1} \cdot (1-\epsilon/7) \ge 2^{-i+1} \cdot R_G(f)$. This implies that $f \in F'_{i-1}$, which contradicts the assumption $f \in F_i = F'_i \setminus \bigcup_{j=0}^{i-1} F'_j$. □

We will consider each group of edges F_i separately, and prove that $x^\top \widetilde{L}_{F_i} x$ concentrates around its expectation, $x^\top L_{F_i} x$. More precisely, we will first prove concentration for every specific $(x^{(i)}, F_i)$, and then extend the concentration to all possibilities simultaneously via union bound. This is well-defined because each F_i depends on $x^{(1)}, \ldots, x^{(i)}$ but not directly on x.

Edges in F_0. By definition, every edge $f \in F_0$ has $p(f) = 1$, and thus $x^\top \widetilde{L}_{F_0} x$ is completely deterministic and equal to $x^\top L_{F_0} x$.

Edges in F_i for $i \in [I]$. Note that F_i is designed so that, by Corollary 3.4.3, for every edge $f \in F_i$ we have $\left|x^\top L_{\{f\}} x - x^{(i)\top} L_{\{f\}} x^{(i)}\right| \le \epsilon/7 \cdot x^{(i)\top} L_{\{f\}} x^{(i)}$, and since $\widetilde{L}_{\{f\}}$ is a multiple of $L_{\{f\}}$ similarly have $\left|x^\top \widetilde{L}_{\{f\}} x - x^{(i)\top} \widetilde{L}_{\{f\}} x^{(i)}\right| \le \epsilon/7 \cdot x^{(i)\top} \widetilde{L}_{\{f\}} x^{(i)}$. Informally, this allows us to prove concentration only for vectors $x^{(i)}$ instead of all x, and thus we next aim to bound the error

$$\left|x^{(i)\top} L_{F_i} x^{(i)} - x^{(i)\top} \widetilde{L}_{F_i} x^{(i)}\right|$$

for each $i \in [I]$ with high probability. It will then remain to bound the error introduced on the remaining edges (the ones in F_{I+1}).

Fix $i \in [I]$ and notice that over all possible vectors $x \in S^G$, there are only finitely many possible values for $(x^{(i)}, F_i)$. Therefore, we can focus on a single value of $x^{(i)}$ and F_i, and then use a union bound over all settings.

Let us therefore fix also $x^{(i)}$ and F_i. We will use Chernoff bounds to prove that with high probability, over the randomness of sampling edges to $\widetilde{G}$,

$$\left|x^{(i)\top} L_{F_i} x^{(i)} - x^{(i)\top} \widetilde{L}_{F_i} x^{(i)}\right| \le \frac{\epsilon}{7I}. \tag{3.5}$$

Indeed, note that

$$x^{(i)\top} \widetilde{L}_{F_i} x^{(i)} = \sum_{f=(u,v)\in F_i} \widetilde{z}(f) \cdot (x_u^{(i)} - x_v^{(i)})^2,$$

where $\widetilde{z}(f)$ are independent random variables with expectation $\mathbb{E}(\widetilde{z}(f)) = z(f)$. Therefore, we can apply the Chernoff bound from Theorem 3.2.10 with $\{Z_i\}_i$ being $\widetilde{z}(f) \cdot (x_u - x_v)^2$ for each $f = (u,v) \in F_i$, and their sum being $Z = x^{(i)\top} \widetilde{L}_{F_i} x^{(i)}$ with $\mathbb{E}(Z) = x^{(i)\top} L_{F_i} x^{(i)}$. We need to set a as an upper bound on $\widetilde{z}(f) \cdot$

$(x_u^{(i)} - x_v^{(i)})^2$. Observe that $\widetilde{z}(f)$ is maximal when f is sampled, in which case it equals $z(f)/p(f)$ where $p(f) = \lambda \cdot z(f) \cdot R_G(f)$, since $f \notin F_0$. We thus get, using Claim 3.4.4,

$$\forall f = (u,v) \in F_i, \qquad \frac{z(f) \cdot (x_u^{(i)} - x_v^{(i)})^2}{\lambda \cdot z(f) \cdot R_G(f)} = \frac{1}{\lambda} \cdot \frac{(x_u^{(i)} - x_v^{(i)})^2}{R_G(f)} \le \frac{3 \cdot 2^{-i}}{\lambda} =: a.$$

We let $\delta := \epsilon/(14I)$, we can bound

$$x^{(i)\top} L_{F_i} x^{(i)} \le \left(1 + \frac{\epsilon}{7}\right) \cdot x^\top L_{F_i} x \le \left(1 + \frac{\epsilon}{7}\right) \cdot x^\top L x = 1 + \frac{\epsilon}{7} \le 2 =: \mu.$$

(This is true for an arbitrary preimage $x \in \varphi_i^{-1}(x^{(i)})$.)

Finally, Theorem 3.2.10 implies

$$\begin{aligned}
\mathbb{P}\left(\left|x^{(i)\top} L_{F_i} x^{(i)} - x^{(i)\top} \widetilde{L}_{F_i} x^{(i)}\right| \ge \frac{\epsilon}{7I}\right) &\le 2\exp\left(-\frac{\delta^2 \mu}{3 \cdot a}\right) \\
&= 2\exp\left(-\frac{\frac{\epsilon^2}{196 I^2} \cdot 2}{9 \cdot 2^{-i}/\lambda}\right) \\
&\le 2\exp\left(-\frac{\epsilon^2 \cdot 2^i \cdot \lambda}{10000 \log^2(n)}\right) \\
&= 2\exp\left(-\frac{2000 C \log n \cdot 2^i}{\epsilon^2}\right),
\end{aligned}$$

where the last step by setting $\lambda = 2 \cdot 10^7 \cdot C \log^3 n/\epsilon^4$, where $C > 0$ is the absolute constant from Theorem 3.4.5.

We can now use a union bound to bound the probability that Eq. (3.5) holds simultaneously for all values of $(x^{(i)}, F_i)$. F_i depends only on F'_j for $j \in [i]$, which in turn depend on $x^{(j)}$ for the same values of j. By the first guarantee of Lemma 3.4.2, the number of possible vectors $x^{(j)}$ is at most $\exp(800 C \log n \cdot 2^j/\epsilon^2)$, where $C > 0$ is the absolute constant from Theorem 3.4.5. Therefore, the number of possible pairs $(x^{(i)}, F_i)$ is at most

$$\prod_{j=1}^{i} \exp\left(\frac{800 C \log n \cdot 2^j}{\epsilon^2}\right) = \exp\left(\sum_{j=1}^{i} \frac{800 C \log n \cdot 2^j}{\epsilon^2}\right) \le \exp\left(\frac{1600 C \log n \cdot 2^i}{\epsilon^2}\right).$$

Finally, the probability that Eq. (3.5) *does not* hold simultaneously for all pairs $(x^{(i)}, F_i)$ is at most

$$\exp\left(\frac{1600 C \log n \cdot 2^i}{\epsilon^2}\right) \cdot 2\exp\left(-\frac{2000 C \log n \cdot 2^i}{\epsilon^2}\right) = 2\exp\left(-\frac{400 C \log n \cdot 2^i}{\epsilon^2}\right) \le \frac{1}{n}.$$

Edges in F_{I+1} . First we show that for any $x \in S^G$ and any edge $f = (u,v) \in F_{I+1}$ we have that $(x_u - x_v)^2 \le \epsilon \cdot R_G(f)/(6n)$. Suppose for contradiction that this is not the case. Then the second guarantee of Lemma 3.4.2 applies and $(x_u^{(I)} - x_v^{(I)})^2 \ge (x_u - x_v)^2 \cdot (1 - \epsilon/7) \ge \epsilon \cdot R_G(e)/(6n) \cdot (1 - \epsilon/7) \ge \epsilon \cdot R_G(e)/(7n)$. Therefore $f \in F'_I$, which contradicts the assumption $f \in F_{I+1}$. (Here we used that I was defined to be $\log_2(7n/\epsilon)$.)

Next, we would like to bound $\left|x^\top \widetilde{L}_{F_{I+1}} x - x^\top L_{F_{I+1}} x\right|$ by showing that both terms are small. First,

$$x^\top L_{F_{I+1}} x = \sum_{f=(u,v)\in F_{I+1}} z(f)\cdot(x_u - x_v)^2 \qquad \le \sum_{f\in F_{I+1}} z(f)\cdot\epsilon\cdot\frac{R_G(f)}{6n} \le \frac{\epsilon}{6n}\cdot\sum_{f\in F} z(f)\cdot R_G(f) \qquad \le \frac{\epsilon}{6},$$

where the last inequality uses Fact 3.2.9. Second, we start similarly,

$$x^\top \widetilde{L}_{F_{I+1}} x = \sum_{f=(u,v)\in F_{I+1}} \widetilde{z}(f)\cdot(x_u - x_v)^2 \qquad \le \sum_{f\in F_{I+1}} \widetilde{z}(f)\cdot\epsilon\cdot\frac{R_G(f)}{6n} \le \frac{\epsilon}{6n}\cdot\sum_{f\in F} \widetilde{z}(f)\cdot R_G(f),$$

and ideally we would like to show that $\sum \widetilde{z}(f)\cdot R_G(f) \le 2n$. This is not always true, but it is a random event, independent of the choice of x, and can be shown to hold with high probability using our Chernoff bound from Theorem 3.2.10. Indeed, $\widetilde{z}(f)\cdot R_G(f)$ are independent random variables with maximum value when f is sampled, in which case $\widetilde{z}(f) = z(f)/p(f)$, and thus

$$\widetilde{z}(f)\cdot R_G(f) \le \frac{z(f)\cdot R_G(f)}{p(f)} = \frac{z(f)\cdot R_G(f)}{\min(1,\lambda\cdot z(f)\cdot R_G(f))} = \max\left(z(f)\cdot R_G(f), 1/\lambda\right) \le 1 =: a,$$

where the last inequality uses Fact 3.2.8. We apply Theorem 3.2.10 by setting $\delta := 1$ and $\mu := n$ (which we may do by Fact 3.2.9), and obtain

$$\mathbb{P}\left(\sum_{f\in F} \widetilde{z}(f)\cdot R_G(f) \ge 2n\right) \le 2\exp\left(-\frac{n}{3}\right).$$

Therefore, with probability at least $1 - O(1/n)$,

$$\left|x^\top \widetilde{L}_{F_{I+1}} x - x^\top L_{F_{I+1}} x\right| \le \frac{\epsilon}{2}. \tag{3.6}$$

Putting everything together.

By the above derivations, Eq. (3.5) holds for all i and all $(x^{(i)}, F_i)$ simultaneously, as well as Eq. (3.6) holds with probability at least $1 - O(\log n/n)$. Assuming henceforth that this high probability event occurs, we shall deduce that Eq. (3.4) holds for all $x \in S^G$. Indeed, by the triangle inequality and Eq. (3.6),

$$\left|x^\top \widetilde{L} x - x^\top L x\right| \le \sum_{i=0}^{I+1} \left|x^\top \widetilde{L}_{F_i} x - x^\top L_{F_i} x\right| \le 0 + \sum_{i=1}^{I} \left|x^\top \widetilde{L}_{F_i} x - x^\top L_{F_i} x\right| + \frac{\epsilon}{2}.$$

Now for each $i \in [I]$, we can approximate terms involving x by $x^{(i)}$ and vice versa, formalized by the aforementioned fact that for every $(u,v)\in F_i$ we have $|(x_u - x_v)^2 - (x_u^{(i)} - x_v^{(i)})^2| \le \epsilon/7\cdot(x_u^{(i)} - x_v^{(i)})^2$ (see the second condition of Lemma 3.4.2 and the definition of $F_i' \subseteq F_i$), and get

$$\begin{aligned}\left|x^\top \widetilde{L}_{F_i} x - x^\top L_{F_i} x\right| &\le \left|x^\top \widetilde{L}_{F_i} x - x^{(i)\top} \widetilde{L}_{F_i} x^{(i)}\right| + \left|x^{(i)\top} \widetilde{L}_{F_i} x^{(i)} - x^{(i)\top} L_{F_i} x^{(i)}\right| + \left|x^{(i)\top} L_{F_i} x^{(i)} - x^\top L_{F_i} x\right| \\ &\le \frac{\epsilon}{7}\cdot x^{(i)\top} \widetilde{L}_{F_i} x^{(i)} + \left|x^{(i)\top} \widetilde{L}_{F_i} x^{(i)} - x^{(i)\top} L_{F_i} x^{(i)}\right| + \frac{\epsilon}{7}\cdot x^{(i)\top} L_{F_i} x^{(i)}\end{aligned}$$

now we use the triangle inequality,

$$\leq \frac{\epsilon}{7} \cdot x^{(i)\top} L_{F_i} x^{(i)} + \left(1 + \frac{\epsilon}{7}\right) \cdot \left| x^{(i)\top} \widetilde{L}_{F_i} x^{(i)} - x^{(i)\top} L_{F_i} x^{(i)} \right| + \frac{\epsilon}{7} \cdot x^{(i)\top} L_{F_i} x^{(i)}$$

and now we crucially use Eq. (3.5),

$$\begin{aligned} &\leq \left(1 + \frac{\epsilon}{7}\right) \cdot \frac{\epsilon}{7I} + \frac{2\epsilon}{7} \cdot \left(1 - \frac{\epsilon}{7}\right)^{-1} \cdot x^\top L_{F_i} x \\ &\leq \frac{\epsilon}{6I} + \frac{2\epsilon}{6} \cdot x^\top L_{F_i} x. \end{aligned}$$

Substituting this into our previous bound, we obtain

$$\left| x^\top \widetilde{L} x - x^\top \widetilde{L} x \right| \leq \sum_{i=1}^{I} \left(\frac{\epsilon}{6I} + \frac{\epsilon}{3} \cdot x^\top L_{F_i} x \right) + \frac{\epsilon}{2} \leq \frac{\epsilon}{6} + \frac{\epsilon}{3} \cdot x^\top L x + \frac{\epsilon}{2} = \epsilon \cdot x^\top L x,$$

where the last equality uses $x^\top L x = 1$. This completes the proof of Theorem 3.4.1. □

3.4.1 Proof of Lemma 3.4.2

To prove Lemma 3.4.2, we use the following Theorem:

Theorem 3.4.5 (Theorem VI.1 of Alon and Klartag (2017))**.** *Let $a_1, \ldots, a_m \in \mathbb{R}^n$ be vectors of norm at most 1 and let $\eta \in (0,1)$. Then, over all vectors $y \in \mathbb{R}^n$ of norm at most 1, the number of possible values of the "rounded vector"*

$$\left(\left\lfloor \frac{\langle a_1, y \rangle}{\eta} \right\rfloor, \left\lfloor \frac{\langle a_2, y \rangle}{\eta} \right\rfloor, \ldots, \left\lfloor \frac{\langle a_k, y \rangle}{\eta} \right\rfloor \right)$$

is at most $\exp\left(\frac{C \log m}{\eta^2}\right)$ for some absolute constant $C > 0$.

Remark 3.4.6. *In fact, the original theorem (Theorem 6.1 in Alon and Klartag (2017)) is stated with stronger requirements on m and η, and a stronger consequence. However, we can easily get the weaker upper bound of $\exp(O(\log m/\eta^2))$ stated in Theorem 3.4.5 of this chapter, by setting the variables appropriately: $\varepsilon := \eta$, $n := \max(m, 1/\eta^2)$, and $k := n$, where the left hand side always represents their variable names and the right hand side ours.*

Proof of Lemma 3.4.2: We use the idea of resistive embedding introduced in . Note that $L = L_G$ is a positive semidefinite matrix, and we denote by $L^{+/2}$ the square root of its Moore-Penrose pseudo-inverse. For each (unordered) vertex pair (u, v), let $b_{u,v} \in \mathbb{R}^V$ be the vector with all zero coordinates, except for the coordinates associated with u and v, which are 1 and -1 (ordered arbitrarily). With each vertex pair (u, v), we associate the vector

$$a_{u,v} = \frac{L^{+/2} b_{u,v}}{\|L^{+/2} b_{u,v}\|_2}.$$

Furthermore, we associate with each $x \in S^G$ the vector $y_x = L^{1/2} x$.

We can then apply Theorem 3.4.5 to $\{a_{u,v} \mid (u,v) \in \binom{V}{2}\}$ and all possible y_x, setting $\eta = \epsilon \cdot 2^{-i/2}/20$. Indeed,

$a_{u,v}$ is normalized by definition, and also y_x is normalized because $x \in S^G$ and thus

$$\|y_x\|_2^2 = x^\top L^{1/2} L^{1/2} x = x^\top L x = 1.$$

For each possible value of the rounded vector

$$\left(\left\lfloor \frac{\langle a_{u,v}, y_x \rangle}{\eta} \right\rfloor \right)_{(u,v)\in\binom{V}{2}}$$

choose a representative $x \in S^G$, and let φ_i map each $x \in S^G$ to its representative (i.e., with the same rounded vector). Then by Theorem 3.4.5, the image of φ_i is of size $|\varphi_i(S^G)| \le \exp\left(800C \log n \cdot 2^i/\epsilon^2\right)$, as claimed. Recall that we denote $\varphi_i(x)$ by $x^{(i)}$; then

$$\left(\left\lfloor \frac{\langle a_{u,v}, y_x \rangle}{\eta} \right\rfloor \right)_{(u,v)\in\binom{V}{2}} = \left(\left\lfloor \frac{\langle a_{u,v}, y_{x^{(i)}} \rangle}{\eta} \right\rfloor \right)_{(u,v)\in\binom{V}{2}}.$$

It follows that for all $f = (u, v) \in F$ and all $x \in S^G$,

$$\left| \langle a_{u,v}, y_x \rangle - \langle a_{u,v}, y_{x^{(i)}} \rangle \right| \le \eta. \tag{3.7}$$

Furthermore, $b_{u,v}$ is perpendicular to the null-space of L (which is spanned by the all-ones vector because G is connected), thus $L^{1/2} L^{+/2} b_{u,v} = b_{u,v}$ and

$$\langle a_{u,v}, y_x \rangle^2 = \frac{\left(x^\top L^{1/2} L^{+/2} b_{u,v}\right)^2}{b_{u,v}^\top L^{+/2} L^{+/2} b_{u,v}} = \frac{\left(x^\top b_{u,v}\right)^2}{b_{u,v}^\top L^{+} b_{u,v}} = \frac{(x_u - x_v)^2}{R_G(u,v)}. \tag{3.8}$$

To prove the second guarantee of Lemma 3.4.2, let $f = (u, v) \in F$ and $x \in S^G$ and consider first the case $\left(x_u^{(i)} - x_v^{(i)}\right)^2 \ge 2^{-i} \cdot R_G(f)$, which by Eq. (3.8) is equivalent to $\langle a_{u,v}, y_{x^{(i)}} \rangle^2 \ge 2^{-i}$. This means that the absolute error bound η in Eq. (3.7) implies a relative error bound, namely,

$$\left| \langle a_{u,v}, y_x \rangle - \langle a_{u,v}, y_{x^{(i)}} \rangle \right| \le \eta = \frac{\epsilon \cdot 2^{-i/2}}{20} \le \frac{\epsilon}{20} \cdot \left| \langle a_{u,v}, y_{x^{(i)}} \rangle \right|.$$

The other case $(x_u - x_v)^2 \ge 2^{-i} \cdot R_G(f)$ is similar up to constants; by Eq. (3.8), this case is equivalent to $\langle a_{u,v}, y_x \rangle^2 \ge 2^{-i}$, and thus

$$\left| \langle a_{u,v}, y_x \rangle - \langle a_{u,v}, y_{x^{(i)}} \rangle \right| \le \eta = \frac{\epsilon \cdot 2^{-i/2}}{20} \le \frac{\epsilon}{20} \cdot \left| \langle a_{u,v}, y_x \rangle \right|,$$

which implies

$$\left| \langle a_{u,v}, y_x \rangle - \langle a_{u,v}, y_{x^{(i)}} \rangle \right| \le \frac{\epsilon}{20} \cdot \left(1 - \frac{\epsilon}{20}\right)^{-1} \left| \langle a_{u,v}, y_x \rangle \right| \le \frac{\epsilon}{16} \cdot \left| \langle a_{u,v}, y_{x^{(i)}} \rangle \right|.$$

Now in both cases,

$$\left| \langle a_{u,v}, y_x \rangle^2 - \langle a_{u,v}, y_{x^{(i)}} \rangle^2 \right| = \left| \langle a_{u,v}, y_x \rangle - \langle a_{u,v}, y_{x^{(i)}} \rangle \right| \cdot \left| \langle a_{u,v}, y_x \rangle + \langle a_{u,v}, y_{x^{(i)}} \rangle \right|$$

$$
\begin{aligned}
&\le \frac{\epsilon}{16} \cdot |\langle a_{u,v}, y_{x^{(i)}} \rangle| \cdot \left(2 + \frac{\epsilon}{16}\right) \cdot |\langle a_{u,v}, y_{x^{(i)}} \rangle| \\
&\le \frac{\epsilon}{7} \cdot \langle a_{u,v}, y_{x^{(i)}} \rangle^2 .
\end{aligned}
$$

Using Eq. (3.8) and scaling by $R_G(u,v)$, we can write this as $(x_u - x_v)^2 \in (1 \pm \epsilon/7) \cdot (x_u^{(i)} - x_v^{(i)})^2$, which completes the proof of Lemma 3.4.2. □

3.5 γ-Balanced Weight Assignments

Our strategy for generalizing the techniques of Section 3.4 to hypergraphs is similar to that of Chen et al. (2020). Intuitively, we wish to replace each hyperedge of the input hypergraph with a weighted clique in such a way that the "importance" of each edge in the same clique is roughly the same. However, our measure of importance is effective resistance, whereas in Chen et al. (2020) it is the *strength* of the edge (see Benczúr and Karger (2015)), which is a measure particularly useful to cut sparsification. Specifically, we use the following definition.

Definition 3.5.1. *Given a hypergraph $H = (V, E, w)$, a weight assignment of H is a weighted (ordinary) graph $G = (V, F, z)$ such that*

- *F is the multiset $\bigcup_{e \in E} F_e$, where F_e is a set of edges forming a clique on the support of e*
- *$\sum_{f \in F_e} z(f) = w(e)$.*

Note that this definition allows for parallel edges in G.

Moreover, if G satisfies

$$\gamma \cdot \min_{g \in F_e \colon z(g) > 0} R_G(g) \ge \max_{f \in F_e} R_G(f)$$

for $\gamma > 1$, then we call it γ-balanced.

The goal of this section is to show the existence of constant-balanced weight assignments for all weighted hypergraphs, and to define an efficient algorithm that outputs such a weight assignment.

3.5.1 Spanning Tree Potential

In order to show the existence (and give an efficient construction) of γ-balanced weight assignments we introduce the concept of spanning tree potentials for weighted ordinary graphs.

Definition 3.5.2 (Spanning tree potential, ST-potential for short). *For a connected weighted graph $G = (V, F, z)$ let $\mathbb{T}(G)$ be the set of all spanning trees of G. Then we define the spanning tree potential of G as*

$$\Psi(G) = \log \left[\sum_{T \in \mathbb{T}(G)} \prod_{f \in T} z(f) \right]. \tag{3.9}$$

Remark 3.5.3. *Note that the value of ST-potential stays the same after replacing parallel edges f_1 and f_2 with a single edge f of weight $z(f) := z(f_1) + z(f_2)$.*

We formalize the concept of edge updates to graphs and see how those updates affect ST-potential.

Definition 3.5.4. *If $G = (V, F, z)$ is a weighted graph, $\lambda \in \mathbb{R}$, and $f \in \binom{V}{2}$, then $G + \lambda \cdot f$ is the weighted graph (V, F', z'), where the weight of f is increased by λ. Formally,*

- *$F' = F$, $z'(f) = z(f) + \lambda$, and $z'(g) = z(g)$ for all $g \neq f$, if $f \in F$,*
- *or $F' = F + \{f\}$, $z'(f) = \lambda$, and $z'(g) = z(g)$ for all $g \neq f$, if $f \notin F$.*

Note that the definition applies to $\lambda < 0$ as well, but in this case f must be present in the graph with weight at least $|\lambda|$ in order for $G + \lambda \cdot f$ to be valid.

Lemma 3.5.5. *For any weighted graph G, $\lambda \in \mathbb{R}$ and $f \in \binom{V}{2}$ such that $G + \lambda \cdot f$ is well defined, we have*

$$\Psi(G + \lambda \cdot f) = \Psi(G) + \log\big(\lambda R_G(f) + 1\big), \tag{3.10}$$

Proof. We use the famous result that if we sample a spanning tree $\mathcal{T}$ randomly from $\mathbb{T}(G)$ such that

$$\mathbb{P}(\mathcal{T} = T) \propto \prod_{g \in T} z(g),$$

then the marginal probability $\mathbb{P}(f \in T)$ is $z(f) \cdot R(f)$ for all $f \in F$ (see, e.g., Lovász (1993)). Let $\mathcal{T}$ be a random variable drawn from such a distribution. By the definition of the distribution of $\mathcal{T}$ we have that

$$\mathbb{P}(f \in \mathcal{T}) = \sum_{T \in \mathbb{T}(G) : T \ni f} \mathbb{P}(\mathcal{T} = T) = \frac{\sum_{T \in \mathbb{T}(G) : T \ni f} \prod_{g \in T} z(g)}{\sum_{T \in \mathbb{T}(G)} \prod_{g \in T} z(g)} \tag{3.11}$$

Therefore, we have for the weight function z' of $G + \lambda \cdot f$,

$$\begin{aligned}
\Psi(G + \lambda \cdot f) &= \log\left[\sum_{T \in \mathbb{T}(G')} \prod_{g \in T} z'(g)\right] \\
&= \log\left[\sum_{T \in \mathbb{T}(G)} \prod_{g \in T} z(g) \cdot \left(1 + \frac{\lambda}{z(f)}\right)^{\mathbb{1}(g=f)}\right] \\
&= \log\left[\sum_{T \in \mathbb{T}} \left(1 + \frac{\lambda}{z(f)}\right)^{\mathbb{1}(f \in T)} \prod_{g \in T} z(g)\right] \\
&= \log\left[\sum_{T \in \mathbb{T}(G)} \prod_{g \in T} z(g) + \frac{\lambda}{z(f)} \cdot \sum_{T \in \mathbb{T}(G) : T \ni f} \prod_{g \in T} z(g)\right].
\end{aligned}$$

We can transform the second term by Eq. (3.11) to continue:

$$\begin{aligned}
&= \log\left[\sum_{T \in \mathbb{T}(G)} \prod_{g \in T} z(g) + \frac{\lambda}{z(f)} \cdot \mathbb{P}(f \in \mathcal{T}) \cdot \sum_{T \in \mathbb{T}(G)} \prod_{g \in T} z(g)\right] \\
&= \log\left[\sum_{T \in \mathbb{T}(G)} \prod_{g \in T} z(g)\right] + \log\left(\frac{\lambda}{z(f)} \cdot \mathbb{P}(f \in \mathcal{T}) + 1\right) \\
&= \Psi(G) + \log(\lambda R_G(f) + 1)
\end{aligned}$$

as claimed. Note that the above calculation is legitimate for positive λ as well as negative. □

3.5.2 Existence of γ-Balanced Weight Assignments

We will now use ST-potential to construct a γ-balanced weight assignment of an arbitrary hypergraph. We can analyze a simple greedy algorithm, which identifies edge pairs contradicting the γ-balancedness condition in Definition 3.5.1, and simply shifts weight from the one with smaller effective resistance to the one with the larger effective resistance (see Algorithm 3). One can show that such an step can be designed to only increase the ST-potential of a weight assignment, and thus the algorithm eventually terminates, returning a γ-balanced weight assignment.

Algorithm 3 Algorithm for constructing a γ-balanced weight assignment.

```
1: procedure GreedyBalancing($H = (V, E, w), \gamma$)
2:     For all $e \in E$ and for all $f \in F_e$, initialize $z(f)$ to $w(e)/\binom{|e|}{2}$
3:     $G \leftarrow (V, \bigcup_{e\in E} F_e, z)$
4:     while $G$ is not a $\gamma$-balanced weight assignment of $H$ do
5:         Select $e \in E$, and $f, g \in F_e$, such that $R_G(f) > \gamma \cdot R_G(g)$ and $z(g) > 0$
6:         $\lambda \leftarrow \min\left(z(g), (\gamma - 1)/(2\gamma \cdot R_G(g))\right)$
7:         $z(f) \leftarrow z(f) + \lambda$
8:         $z(g) \leftarrow z(g) - \lambda$
9:     return $G$
```

The following tells how much the effective resistance of an edge changes by updating the weight of another edge. Although the proof is simple and this result is already known, we include the proof for completeness.

Lemma 3.5.6. *If $G = (V, F, z)$ is a weighted graph, let $\lambda \in \mathbb{R}$, and $f \in \binom{V}{2}$, then for any $g \in \binom{V}{2}$*

$$R_{G+\lambda\cdot f}(g) = R_G(g) - \frac{\lambda \cdot \left(b_g^\top L_G^+ b_f\right)^2}{1 + \lambda \cdot R_G(f)}.$$

Proof. Note that

$$L_{G+\lambda\cdot f} = L_G + \lambda b_f b_f^\top.$$

Therefore, by the Sherman-Morrison formula for Moore-Penrose pseudoinverse (see for example Meyer (1973)), we can expand the formula for the effective resistance of g:

$$R_{G+\lambda\cdot f}(g) = b_g^\top L_{G+\lambda\cdot f}^+ b_g = b_g^\top \left(L_G + \lambda b_f b_f^\top\right)^+ b_g = b_g^\top \left(L_G^+ - \frac{L_G^+ b_f \lambda b_f^\top L_G^+}{1 + \lambda b_f^\top L_G^+ b_f}\right) b_g$$

$$= b_g^\top L_G^+ b_g - \frac{\lambda b_g^\top L_G^+ b_f b_f^\top L_G^+ b_g}{1 + \lambda b_f^\top L_G^+ b_f} = R_G(g) - \frac{\lambda \cdot \left(b_g^\top L_G^+ b_f\right)^2}{1 + \lambda \cdot R_G(f)}.$$ □

Lemma 3.5.7. *Let $G = (V, F, z)$ be a weighted graph and let $\gamma > 1$. Let f, g be two edges in F such that $R_G(f) > \gamma \cdot R_G(g)$. Then for any $\lambda \le z(g)$, shifting λ weight from g to f results in an increase of at least*

$$\log\left(1 + \lambda\gamma \cdot R_G(g) - \lambda \cdot R_G(g) - \lambda^2\gamma \cdot R_G(g)^2\right)$$

in the ST-potential of G.

Proof. We simply apply the update formula for ST-potential (Lemma 3.5.5) twice, along with Lemma 3.5.6. For simplicity, we use t_{fg} to denote $b_f^\top L_G^+ b_g$. Then the increase in ST-potential is

$$\begin{aligned}
&\log\left(\lambda \cdot R_G(f) + 1\right) + \log\left(-\lambda \cdot R_{G+\lambda \cdot f}(g) + 1\right) \\
&= \log\left(\lambda \cdot R_G(f) + 1\right) + \log\left(-\lambda \cdot \left(R_G(g) - \frac{\lambda t_{fg}^2}{1 + \lambda R_G(f)}\right) + 1\right) \\
&= \log\left[\left(\lambda \cdot R_G(f) + 1\right) \cdot \left(-\lambda \cdot \left(R_G(g) - \frac{\lambda t_{fg}^2}{1 + \lambda R_G(f)}\right) + 1\right)\right] \\
&= \log\left(1 + \lambda \cdot R_G(f) - \lambda \cdot R_G(g) - \lambda^2 \cdot R_G(f) R_G(g) + \lambda^2 t_{fg}^2\right) \\
&\ge \log\left(1 + \lambda\gamma \cdot R_G(g) - \lambda \cdot R_G(g) - \lambda^2\gamma \cdot R_G(g)^2\right),
\end{aligned}$$

as claimed. □

Theorem 3.5.8. *For $\gamma > 1$, Algorithm 3 terminates and returns a γ-balanced weight assignment of H.*

Proof. It is clear by the condition of the while-loop that if Algorithm 3 terminates, it returns a γ-balanced weight assignment. Also, the condition $\sum_{f \in F_e} z(f) = w(e)$ is never violated. Therefore, in Algorithm 3, there must indeed always be some $e \in E$ and some $f, g \in F_e$ such that $R_G(f) > \gamma \cdot R_G(g)$ and $z(g) > 0$, otherwise G would already be γ-balanced. It remains to prove that Algorithm 3 always terminates.

Let us examine the evolution of $\Psi(G)$ throughout the algorithm. First, note that G never becomes disconnected, and hence $\Psi(G)$ always remains defined. Indeed, in order for G to become disconnected, we would have to set λ to $z(g)$ for a *bridge* g. However, if g is a bridge, $R_G(g) = 1/z(g)$ by Fact 3.2.8, and λ is set instead to $(\gamma - 1)z(g)/(2\gamma) < z(g)$.

In each iteration of the while-loop (which we call a step) we move λ weight from some edge g to some other edge f. By Lemma 3.5.7 this results in a change of

$$\log\left(1 + \lambda\gamma \cdot R_G(g) - \lambda \cdot R_G(g) - \lambda^2\gamma \cdot R_G(g)^2\right).$$

Here we distinguish between the following two cases: If $\lambda = (\gamma - 1)/(2\gamma \cdot R_G(g))$ exactly, then the increase in $\Psi(G)$ is at least $\log(1 + (\gamma - 1)^2/(2\gamma) - (\gamma - 1)^2/(4\gamma)) \ge \log(1 + (\gamma - 1)^2/(4\gamma)) =: c_\gamma$. On the other hand, if $\lambda = z(g) \le (\gamma - 1)/(2\gamma \cdot R_G(g))$ then the increase in $\Psi(G)$ is at least $\log(1 + \lambda\gamma \cdot R_G(g) - \lambda \cdot R_G(g) - \lambda(\gamma - 1) \cdot R_G(g)/2) > \log(1) = 0$.

Overall there are two possibilities each step:

1. $\Psi(G)$ increases by at least $c_\gamma > 0$,

2. or $z(g)$ becomes 0, and $\Psi(G)$ increases by a positive amount.

Let the initial setting of G (before the while-loop) be G_0. Let G_∞ be the complete graph on V with uniform edge weights of $\sum_{e\in E} w(e)$ on each edge. Since G always satisfies

$$\sum_{f\in F} z(f) = \sum_{e\in E} \sum_{f\in F_e} z(f) = \sum_{e\in E} w(e),$$

$\Psi(G)$ will always be less than $\Psi(G_\infty)$ by monotonicity of Ψ. Thus, there can be at most $(\Psi(G_\infty) - \Psi(G_0))/c_\gamma$ steps of type 1. Therefore, after a certain point, there can only be steps of type 2; we focus on this stage of the algorithm.

We further categorize steps of type 2 based on the initial weight of f:

2a. The initial weight of f is greater than 0,

2b. the initial weight of f is exactly 0.

Steps of type 2a increase the total number of edges of weight exactly 0, since by definition, neither f nor g starts out with weight 0, but $z(g)$ becomes 0. On the other hand, steps of type 2b do not decrease the total number of edges of weight 0. Therefore, after a certain point, there can only be steps of type 2b; we focus on this stage of the algorithm.

At this point, the set of all edge weight values, that is

$$\bigcup_{f\in F} \{z(f)\},$$

remains unchanged. Indeed at every step we simply switch the values of $z(f)$ and $z(g)$. Therefore, there are only a finite number of possible states for G to be in. None of these can be repeated, as $\Psi(G)$ increases by a positive amount after each step, and therefore, Algorithm 3 must terminate, returning a γ-balanced weight assignment of H. □

This proves the existence of γ-balanced weight assignment, which suffices to show the existence of nearly-linear-sized spectral sparsifiers (as we will see in Section 3.6). Unfortunately, the above proof shows no bound on the running time of Algorithm 3 beyond $2^{O(m)}$.

3.5.3 Polynomial-Time Construction

In this section, we introduce a relaxation of the concept of γ-balanced weight assignment. This will still be sufficient to get spectral sparsifiers of nearly linear size, while also allowing the greedy algorithm to terminate in a polynomial number of steps.

Definition 3.5.9. *For $0 < \eta \leq 1$ and $\gamma > 1$, an η-approximate γ-balanced weight assignment of $H = (V, E, w)$ is defined exactly as above in Definition 3.5.1, except with the final condition relaxed to*

$$\gamma \cdot \min_{g \in F_e:\, z(g) \geq \eta \cdot w(e)} R_G(g) \geq \max_{f \in F_e} R_G(f).$$

That is, we allow edges not only of weight 0, but also of weight less than $\eta \cdot w(e)$ to be small outliers in terms of effective resistance.

We modify Algorithm 3 to search for approximate γ-balanced weight assignments.

Algorithm 4 Algorithm for constructing an η-approximate γ-balanced weight assignment.

```
1: procedure GreedyApproxBalancing($H = (V, E, w), \gamma, \eta$)
2:     For all $e \in E$ and for all $f \in F_e$, initialize $z(f)$ to $w(e)/\binom{|e|}{2}$
3:     $G \leftarrow (V, \bigcup F_e, z)$
4:     while $G$ is not an $\eta$-approximate $\gamma$-balanced weight assignment of $H$ do
5:         Select $e \in E$, and $f, g \in F_e$, such that $R_G(f) > \gamma \cdot R_G(g)$ and $z(g) > \eta \cdot w(e)$
6:         $\lambda \leftarrow \min\big(z(g), 1/(2R_G(g))\big)$
7:         $z(f) \leftarrow z(f) + \lambda$
8:         $z(g) \leftarrow z(g) - \lambda$
9:     return $G$
```

Theorem 3.5.10. *Let $H = (V, E, w)$ be a weighted hypergraph Then for $\gamma \geq 4$ and $\eta > 0$, Algorithm 4 terminates* within $M/(\eta w_{\min}) \cdot \text{poly}\big(n \log(M/w_{\min})\big)$ rounds *and returns an η-approximate γ-balanced weight assignment for H, where $w_{\min} := \min_{e \in E} w(e)$ and $M := \sum_{e \in E} w(e)$.*

Proof. It is clear by the condition of the while-loop that if Algorithm 4 terminates, then it returns an η-approximate γ-balanced weight assignment. Also, the condition $\sum_{f \in F_e} z(f) = w(e)$ is never violated. Therefore, in Algorithm 4, there must indeed always be some $e \in E$ and some $f, g \in F_e$ such that $R_G(f) > \gamma$ and $z(g) > \eta \cdot w(e)$, otherwise G would already be η-approximately γ-balanced. It remains to prove that Algorithm 4 terminates within $M/(\eta w_{\min}) \cdot \text{poly}\big(n \log(M/w_{\min})\big)$ rounds.

Let G_0 be the starting graph of Algorithm 4, and G_∞ be the complete graph on V with uniform edge weights of M. Since $\Psi(G_\infty)$ is always greater than $\Psi(G)$ at every moment in the algorithm, we can upper bound the total increase in ST-potential throughout the algorithm by $\Psi(G_\infty) - \Psi(G_0)$. Let us upper bound $\Psi(G_\infty) - \Psi(G_0)$. To this end we will produce a sequence of updates to G_0 resulting in G_∞, and we will upper bound the contribution of each update.

Since $\min_{e \in E} w(e) = w_{\min}$, $\min_{f \in F} z(f) \geq w_{\min}/n^2$ in G_0. Furthermore, since H is connected, G_0 is connected as well and must contain a spanning tree with edges of weight at least $w_{\min}/n^2$ and therefore $\min_{u,v \in V} R_{G_0}(u, v) \leq n/(w_{\min}/n^2) = n^3/w_{\min}$. To transform G_0 to G_∞, we simply add to each vertex pair (u, v) a sufficient amount of weight to make $w(u, v) = M$. This is an update of $+\lambda \cdot (u, v)$ with $\lambda \leq M$ and $R(u, v) \leq n^3/w_{\min}$, which contributes by at most $\log(Mn^3/w_{\min} + 1)$ to Ψ by Lemma 3.5.5. The total contribution of all such updates on the way from G_0 to G_∞ is at most

$$\sum_{u,v\in V} \log\left(\frac{Mn^3}{w_{\min}}+1\right) = \text{poly}\left(n\log\left(\frac{M}{w_{\min}}\right)\right).$$

We now lower bound the minimum increase of Ψ after each step of Algorithm 4. We are able to do this thanks to the modification in Definition 3.5.9 of *approximate* γ-balanced weight assignment.

Due to Lemma 3.5.7, the contribution of each update to the potential $\Psi(G)$ is at least

$$\log\left(1+\lambda\gamma\cdot R_G(g)-\lambda\cdot R_G(g)-\lambda^2\gamma\cdot R_G(g)^2\right)$$

At each step, λ is set to either $z(g)$ or $1/(2R_G(g))$. If $\lambda = 1/(2R_G(g))$, the increase in $\Psi(G)$ is at least $\log(1+(\gamma-1)/2-\gamma/4) \geq \log(5/4)$. On the other hand, if $\lambda = z(g) \leq 1/(2R_G(g))$, then the increase in $\Psi(G)$ is at least

$$\begin{aligned}\log(1+\lambda\gamma\cdot R_G(g)-\lambda\cdot R_G(g)-\lambda\gamma\cdot R_G(g)/2) &\geq \log(1+\lambda\cdot R_G(g))\\ &= \log(1+z(g)\cdot R_G(g))\\ &\geq \log(1+\eta\cdot w_{\min}\cdot R_G(g)).\end{aligned}$$

To lower bound this, note that for all g

$$R_G(g) \geq R_{G_\infty}(g) = \frac{2}{Mn}.$$

This gives us that after each round of the algorithm $\Psi(G)$ increases by at least

$$\log(1+\eta\cdot w_{\min}\cdot R_G(g)) \geq \frac{2\cdot 2\eta w_{\min}/(nM)}{2+2\eta w_{\min}/(nM)} \geq \frac{4\eta w_{\min}/(nM)}{4} = \frac{\eta w_{\min}}{nM},$$

where we used $\log(1+x) \geq (2x)/(2+x)$ for $x \geq 0$ in the first inequality and we assumed n is sufficiently large in the second inequality. Thus the algorithm takes at most $M/(\eta w_{\min})\cdot \text{poly}\left(n\log(M/w_{\min})\right)$ steps. □

3.6 Hypergraph Sparsification

In this section, we prove the existence of a spectral sparsifier with a nearly linear number of hyperedges, that is, the first part of Theorem 1. We discuss efficient construction of spectral sparsifier in Section 3.7.

To construct a spectral sparsifier, we first produce a $1/n^2$-approximate γ-balanced weight assignment for the input hypergraph, where $\gamma \geq 4$. We can use Algorithm 4 for this. We then assign to each hyperedge e importance equal to the maximum effective resistance in F_e (see Definition 3.5.1). We then perform typical importance sampling on the hyperedges, oversampling by a factor of λ (to be set later). A formal description is given in Algorithm 5.

Algorithm 5 ϵ-spectral sparsification for a hypergraph, using importance sampling.

```
1: procedure SPARSIFICATION($H = (V, E, w), G = (V, F, z), \epsilon, \lambda$)
2:     $\widetilde{H} = (V, \widetilde{E}, \widetilde{w}) \leftarrow (V, \emptyset, 0)$
3:     for all $e \in E$ do
4:         $R^{\max}(e) \leftarrow \max_{f \in F_e} R_G(f)$
5:         $p(e) \leftarrow \min(1, w(e) \cdot R^{\max}(e) \cdot \lambda)$
6:         With probability $p(e)$, $\widetilde{E} \leftarrow E \cup \{e\}$ and $\widetilde{w}(e) \leftarrow w(e)/p(e)$
7:     return $\widetilde{H}$
```

In the rest of this section, we show the correctness of this approach. We first bound the size of the hypergraph output by Algorithm 5. We then prove that the output $\widetilde{H}$ is indeed an ϵ-spectral sparsifier of H—this is the technical core of the section.

Lemma 3.6.1. *Let $H = (V, E, w)$ be a weighted hypergraph and let $G = (V, F, z)$ be its $1/n^2$-approximate γ-balanced weight assignment for $\gamma \geq 4$. Then, Algorithm 5 returns a hypergraph of expected size $\mathbb{E}(|\widetilde{E}|) \leq 2\lambda\gamma n$.*

Proof. Each hyperedge e contributes $p(e) \leq w(e) \cdot R^{\max}(e) \cdot \lambda$ to $\mathbb{E}(|\widetilde{E}|)$, and it thus suffices to bound

$$\sum_{e \in E} w(e) \cdot R^{\max}(e) \leq 2\gamma \cdot n. \tag{3.12}$$

We proceed to prove this inequality. For each $e \in E$, let us partition F_e into two groups, $F_e^{(1)} = \{f \in F_e \mid \gamma \cdot R_G(f) \geq R^{\max}(e)\}$ and the remaining edges $F_e^{(2)} = F_e \setminus F_e^{(1)}$. By Definition 3.5.9, these remaining edges $f \in F_e^{(2)}$ satisfy $z(f) \leq w(e)/n^2$. Then, by Fact 3.2.9, we have

$$n - 1 = \sum_{f \in F} z(f) \cdot R_G(f) = \sum_{e \in E} \sum_{f \in F_e} z(f) \cdot R_G(f) \geq \sum_{e \in E} \sum_{f \in F_e^{(1)}} z(f) \cdot \frac{1}{\gamma} \cdot R^{\max}(e),$$

where the second equality is due to the fact that F is the multiset $\bigcup_{e \in E} F_e$ by Definition 3.5.1. For the inner summation, we can bound

$$\sum_{f \in F_e^{(1)}} z(f) \geq \sum_{f \in F_e} z(f) - \sum_{f \in F_e^{(2)}} \frac{w(e)}{n^2} \geq w(e) - \frac{1}{2} \cdot w(e) = \frac{1}{2} \cdot w(e),$$

and altogether we obtain $\sum_{e \in E} w(e) \cdot R^{\max}(e) \leq 2\gamma \cdot (n-1)$, which completes the proof. □

Setting $\gamma = 4$, for example, thus produces a linear-size output. We now prove that the output is indeed a spectral sparsifier of H.

Lemma 3.6.2. *Let $H = (V, E, w)$ be a weighted hypergraph and let $G = (V, F, z)$ be its $1/n^2$-approximate γ-balanced weight assignment for some* constant *$\gamma \geq 4$. Then executing Algorithm 5 on H, G, $1/n \leq \epsilon < 1$, and $\lambda = O(\log^3(n)/\epsilon^4)$, returns with probability at least $1 - O(\log(n)/n)$ an ϵ-spectral sparsifier of H.*

Proof. We proceed similarly to our proof of Theorem 3.4.1 in Section 3.4. By Definition 3.2.4, we must prove that for every $x \in \mathbb{R}^V$,

$$\left|Q_H(x) - Q_{\widetilde{H}}(x)\right| \le \epsilon \cdot Q_H(x), \tag{3.13}$$

Since Eq. (3.13) is invariant to scaling, we may assume without loss of generality that $x^\top L_G x = 1$, and we denote the set of such vectors by $S^G \subseteq \mathbb{R}^V$. Notice that S^G is defined with respect to L_G and not Q_H, however it implies that $Q_H(x) \ge 1$ because for all $x \in S^G$,

$$\begin{aligned} Q_H(x) &= \sum_{e\in E} w(e)\cdot \max_{u^*,v^*\in e}(x_{u^*}-x_{v^*})^2 = \sum_{e\in E}\sum_{f\in F_e} z(f)\cdot \max_{u^*,v^*\in e}(x_{u^*}-x_{v^*})^2 \\ &\ge \sum_{e\in E}\sum_{f=(u,v)\in F_e} z(f)\cdot(x_u-x_v)^2 = \sum_{f=(u,v)\in F} z(f)\cdot(x_u-x_v)^2 = x^\top L_G x = 1. \end{aligned} \tag{3.14}$$

For any $E' \subseteq E$, we denote the quadratic form $Q(x)$ restricted only to the hyperedges E' in H by $Q_{E'}(x)$, and similarly in the hypergraph $\widetilde{H}$ by $\widetilde{Q}_{E'}(x)$. If $E' = E$, we omit the subscript. It is clear by the construction of $\widetilde{H}$ that

$$\mathbb{E}\left(\widetilde{Q}(x)\right) = Q(x).$$

Therefore, for any specific vector x, Eq. (3.13) holds by Chernoff bound (Theorem 3.2.10). In order to prove it simultaneously for all $x \in S^G$, we again use progressively finer and finer roundings of x, as guaranteed by Lemma 3.4.2.

Indeed, fix a sequence of the rounding functions φ_i guaranteed by Lemma 3.4.2 for $i = 1,\ldots,I := \log_2(14\gamma n/\epsilon) \le 3\log n$ (since γ is a fixed constant and n is sufficiently large), and denote the sequence of rounded vectors for $x \in S^G$ by $x^{(i)} = \varphi_i(x)$. We define for each $x^{(i)}$ the set of hyperedges

$$E'_i := \left\{ e \in E \;\middle|\; \max_{u,v\in e}\left(x_u^{(i)} - x_v^{(i)}\right)^2 \ge 2^{-i}\cdot R^{\max}(e)\right\},$$

where by definition $R^{\max}(e) = \max_{f\in F_e} R_G(f)$. This set E'_i is designed such that the rounding φ_i will conserve (to within a small multiplicative error) $Q_{\{e\}}(x)$ for any $e \in E'_i$ (we will see a proof of this fact later on). We can now define a partition of E based on the sets E'_i. First, recall that $p(e) = \min(1, \lambda\cdot w(e)\cdot R^{\max}(e))$, and let the base case be $E_0 = E'_0 := \{e \in E \mid p(e) = 1\}$. Then for each $i = 1,\ldots,I$, let

$$E_i := E'_i \setminus \bigcup_{j=0}^{i-1} E'_j.$$

Finally, let $E_{I+1} := E \setminus \bigcup_{i=0}^{I} E'_i$.

We begin with two useful claims that will help us bridge the gap between the ordinary graph and the hypergraph settings. First, we extend the second guarantee of Lemma 3.4.2 to hyperedges.

Claim 3.6.3. *For all $x \in S^G$ and $e \in E$ such that*

$$\max\left(\max_{u,v\in e}(x_u - x_v)^2, \max_{u,v\in e}(x_u^{(i)} - x_v^{(i)})^2\right) \ge 2^{-i}\cdot R^{\max}(e),$$

we have

$$\max_{u,v\in e}(x_u - x_v)^2 \in \left(1 \pm \frac{\epsilon}{7}\right) \cdot \max_{u,v\in e}(x_u^{(i)} - x_v^{(i)})^2. \tag{3.15}$$

Proof. We focus on the case when $\max_{u,v\in e}(x_u^{(i)} - x_v^{(i)})^2 \geq 2^{-i} \cdot R^{\max}(e)$. The case when $\max_{u,v\in e}(x_u - x_v)^2 \geq 2^{-i} \cdot R^{\max}(e)$ follows by an identical argument.

For one direction, let the pair $u^*, v^* \in e$ maximize $(x_u^{(i)} - x_v^{(i)})^2$. Now

$$(x_{u^*}^{(i)} - x_{v^*}^{(i)})^2 = \max_{u,v\in E}(x_u^{(i)} - x_v^{(i)})^2 \geq 2^{-i} \cdot R^{\max}(e) = 2^{-i} \cdot \max_{f\in F_e} R_G(f) \geq 2^{-i} \cdot R_G(u^*, v^*),$$

hence the second guarantee of Lemma 3.4.2 holds for (u^*, v^*), and consequently

$$(x_{u^*}^{(i)} - x_{v^*}^{(i)})^2 \leq \left(1 - \frac{\epsilon}{7}\right)^{-1} \cdot (x_{u^*} - x_{v^*})^2 \leq \left(1 - \frac{\epsilon}{7}\right)^{-1} \cdot \max_{u,v\in e}(x_u - x_v)^2.$$

The other direction follows by a similar argument, but is slightly more complicated. The asymmetry is due to our assumption (that $\max_{u,v\in e}(x_u^{(i)} - x_v^{(i)}) \geq 2^{-i} \cdot R^{\max}(e)$) being in terms of $x^{(i)}$ instead of x.

Let u^*, v^* be a vertex pair $u, v \in e$ that maximizes $(x_u - x_v)^2$. Here we distinguish two cases: If $(x_{u^*} - x_{v^*})^2 < 2^{-i} \cdot R^{\max}(e)$, we are immediately done, since

$$\max_{u,v\in e}(x_u - x_v)^2 = (x_{u^*} - x_{v^*})^2 \leq 2^{-i} \cdot R^{\max}(e) \leq \max_{u,v\in e}(x_u^{(i)} - x_v^{(i)})^2.$$

On the other hand, if $(x_{u^*} - x_{v^*})^2 \geq 2^{-i} \cdot R^{\max}(e)$, we proceed identically to the first half of the proof:

$$(x_{u^*} - x_{v^*})^2 \geq 2^{-i} \cdot R^{\max}(e) = 2^{-i} \cdot \max_{f\in F_e} R_G(f) \geq 2^{-i} \cdot R_G(u^*, v^*).$$

Therefore, the second guarantee of Lemma 3.4.2 holds for (u^*, v^*) in G, and in particular

$$(x_{u^*} - x_{v^*})^2 \leq \left(1 + \frac{\epsilon}{7}\right) \cdot (x_{u^*}^{(i)} - x_{v^*}^{(i)})^2 \leq \left(1 + \frac{\epsilon}{7}\right) \cdot \max_{u,v\in e}(x_u^{(i)} - x_v^{(i)})^2.$$ □

We thus obtain an analogue to Corollary 3.4.3 from Section 3.4.

Corollary 3.6.4. *For every edge $e \in E_i$,*

$$Q_{\{e\}}(x) \in \left(1 \pm \frac{\epsilon}{7}\right) \cdot Q_{\{e\}}(x^{(i)}).$$

The same holds also for $\widetilde{Q}_{\{e\}}$ (because it is a multiple of $Q_{\{e\}}$).

Proof. We have that $e \in E_i \subseteq E_i'$ and hence $\max_{u,v\in e}(x_u^{(i)} - x_v^{(i)})^2 \geq 2^{-i} \cdot R^{\max}(e)$. We can therefore apply Claim 3.6.3 and scale up by $w(e)$ to get the desired result. □

Next, we prove an analogue to Claim 3.4.4 from Section 3.4.

Claim 3.6.5. *For all $i \in [I]$ and $e \in E_i$, we have*

$$\max_{u,v\in e}(x_u^{(i)} - x_v^{(i)})^2 \le 3\cdot 2^{-i}\cdot R^{\max}(e).$$

Proof. The proof proceeds nearly identically to that of Claim 3.4.4.

The condition of Claim 3.6.3 holds for $e \in E_i$ and hence we have $\max_{u,v\in e}(x_u^{(i)} - x_v^{(i)})^2 \le (1-\epsilon/7)^{-1}\cdot \max_{u,v\in e}(x_u - x_v)^2$.

Consider first the case $i = 1$. Let $u^*, v^* \in e$ maximize $(x_u - x_v)^2$. Then by Fact 3.2.6 and since $x \in S^G$, we have $\max_{u,v\in e}(x_u - x_v)^2 = (x_{u^*} - x_{v^*})^2 \le R_G(u^*, v^*)\cdot x^\top L_G x = R_G(u^*, v^*)$, and we indeed get

$$\begin{aligned}
\max_{u,v\in e}(x_u^{(i)} - x_v^{(i)})^2 &\le \max_{u,v\in e}(x_u - x_v)^2\cdot\left(1-\frac{\epsilon}{7}\right)^{-1}\\
&\le R_G(u^*, v^*)\cdot\left(1-\frac{\epsilon}{7}\right)^{-1}\\
&\le R^{\max}(e)\cdot\left(1-\frac{\epsilon}{7}\right)^{-1}\\
&\le 3\cdot 2^{-1}\cdot R^{\max}(e).
\end{aligned}$$

Now consider $i > 1$, and suppose towards contradiction that $\max_{u,v\in e}(x_u^{(i)} - x_v^{(i)})^2 > 3\cdot 2^{-i}\cdot R^{\max}(e)$. Notice that since $\max_{u,v\in e}(x_u - x_v)^2 \ge 2^{-i+1}\cdot R^{\max}(e)$, Claim 3.6.3 still applies to e with respect to φ_{i-1}. Thus

$$\begin{aligned}
\max_{u,v\in e}(x_u^{(i-1)} - x_v^{(i-1)})^2 &\ge \max_{u,v\in e}(x_u - x_v)^2\cdot\left(1+\frac{\epsilon}{7}\right)^{-1}\\
&\ge \max_{u,v\in e}(x_u^{(i)} - x_v^{(i)})^2\cdot\left(1-\frac{\epsilon}{7}\right)\cdot\left(1+\frac{\epsilon}{7}\right)^{-1}\\
&\ge 2^{-i+1}\cdot R^{\max}(e).
\end{aligned}$$

This implies that $e \in E'_{i-1}$, which contradicts the assumption $e \in E_i$. □

We consider each group of hyperedges E_i separately, and prove that the quadratic form is well approximated on this subset with high probability, that is

$$\widetilde{Q}_{E_i}(x) \approx Q_{E_i}(x).$$

Hyperedges in E_0. By definition, for all $e \in E_0$, we have $p(e) = 1$. Therefore, $\widetilde{Q}_{E_0}(x) = Q_{E_0}(x)$ deterministically.

Hyperedges in E_i for $i \in [I]$. Similarly to the case of ordinary graphs in Section 3.4, we want to consider $x^{(i)}$ instead of x. We may do this due to Corollary 3.6.4. Therefore, we can focus on proving that $Q_{E_i}(x^{(i)}) \approx$

$\widetilde{Q}_{E_i}(x^{(i)})$.

Consider a specific $i \in [I]$. We wish to prove that with high probability, simultaneously for all values of $x^{(i)}$ and $E^{(i)}$,

$$\left|\widetilde{Q}_{E_i}(x^{(i)}) - Q_{E_i}(x^{(i)})\right| \le \frac{\epsilon \cdot Q(x)}{7I}. \tag{3.16}$$

Since (by the first guarantee of Lemma 3.4.2) there are only finitely many values $(x^{(i)}, E_i)$, we can prove that Eq. (3.16) holds with high probability individually for each such pair, and then use a union bound over all pairs.

The former can be done using Chernoff bounds (Theorem 3.2.10); we begin with this. Let us fix $x^{(i)}$ and E_i. Recall that

$$\widetilde{Q}_{E_i}(x^{(i)}) = \sum_{e \in E_i} \widetilde{w}(e) \cdot \max_{u,v \in e}(x_u^{(i)} - x_v^{(i)})^2$$

is the sum of independent random variables with expectation $\mathbb{E}(\widetilde{Q}_{E_i}(x^{(i)})) = Q_{E_i}(x^{(i)})$ because $\mathbb{E}(\widetilde{w}(e)) = w(e)$ by definition. To apply Chernoff bounds (Theorem 3.2.10), we need to set the variables a, δ and μ. We set $\delta := \epsilon/(14I)$, and using Corollary 3.6.4, we can bound

$$Q_{E_i}(x^{(i)}) \le \left(1 - \frac{\epsilon}{7}\right)^{-1} \cdot Q_{E_i}(x) \le 2Q(x) =: \mu.$$

(This is true for an arbitrary preimage $x \in \varphi_i^{-1}(x^{(i)})$.) We also need to set a as an upper bound on the largest possible value of any variable of the form $\widetilde{w}(e) \cdot \max_{u,v\in e}(x_u^{(i)} - x_v^{(i)})^2$ for $e \in E_i$. Such a variable takes its maximum value when e is sampled to the sparsifier $\widetilde{H}$, in which case $\widetilde{w}(e) = w(e)/p(e) = w(e)/(\lambda \cdot w(e) \cdot R^{\max}(e)) = 1/(\lambda \cdot R^{\max}(e))$, since $e \notin E_0$. Therefore, the random variable in question is upper bounded, using Claim 3.6.5, by

$$\max_{e \in E_i} \frac{\max_{u,v\in e}(x_u^{(i)} - x_v^{(i)})^2}{\lambda \cdot R^{\max}(e)} \le \frac{3 \cdot 2^{-i}}{\lambda} =: a.$$

Finally, Theorem 3.2.10 implies

$$\begin{aligned}\mathbb{P}\left(\left|Q_{E_i}(x^{(i)} - \widetilde{Q}_{E_i}(x^{(i)}))\right| \ge \frac{\epsilon \cdot Q(x)}{7I}\right) &\le 2\exp\left(-\frac{\delta^2 \mu}{3a}\right) \\ &= 2\exp\left(-\frac{\frac{\epsilon^2}{196I^2} \cdot Q(x)}{3 \cdot 3 \cdot 2^{-i}/\lambda}\right) \\ &\le 2\exp\left(-\frac{\epsilon^2 \cdot 2^i \cdot \lambda}{10000 \log^2(n)}\right) \\ &= 2\exp\left(-\frac{2000C\log(n) \cdot 2^i}{\epsilon^2}\right),\end{aligned}$$

where the last step is by setting $\lambda = 2 \cdot 10^7 \cdot C \log^3(n)/\epsilon^4$.

We now turn to applying a union bound over all possible values of $(x^{(i)}, E_i)$. Much like F_i in the proof of Theorem 3.4.1, E_i depends on all E'_j for $j \le i$, which in turn depend on all $x^{(j)}$ for $j \le i$. Since $x^{(j)} = \varphi_j(x)$, by the first guarantee of Lemma 3.4.2, there are at most $\exp(800C\log(n) \cdot 2^j/\epsilon^2)$ possible vectors $x^{(j)}$ (where

C is the absolute constant from Theorem 3.4.5). Therefore, the number of possible pairs $(x^{(i)}, E_i)$ is at most

$$\prod_{j=1}^{i} \exp\left(\frac{800C\log(n)\cdot 2^j}{\epsilon^2}\right) = \exp\left(\sum_{j=1}^{i} \frac{800C\log(n)\cdot 2^j}{\epsilon^2}\right) \le \exp\left(\frac{1600C\log(n)\cdot 2^i}{\epsilon^2}\right).$$

Putting together the Chernoff bound and the union bound, we get that Eq. (3.16) holds simultaneously for all values of $(x^{(i)}, E_i)$ except with probability at most

$$\exp\left(\frac{1600C\log(n)\cdot 2^i}{\epsilon^2}\right)\cdot 2\exp\left(-\frac{2000C\log(n)\cdot 2^i}{\epsilon^2}\right) = 2\exp\left(-\frac{400C\log(n)\cdot 2^i}{\epsilon^2}\right) \le \frac{1}{n}.$$

Hyperedges in E_{I+1} . Recall that $I = \log_2(14\gamma\cdot n/\epsilon)$. First we show that for any hyperedge $e \in E_{I+1}$, we have that $\max_{u,v\in e}(x_u - x_v)^2 \le \epsilon \cdot R^{\max}(e)/(12\gamma\cdot n)$. Indeed, suppose for contradiction that this is not the case, and let $u^*, v^* \in e$ be a vertex pair that maximizes $(x_u - x_v)^2$. Then, by the second guarantee of Lemma 3.4.2, we have that $(x^{(I)}_{u^*} - x^{(I)}_{v^*})^2 \ge (x_{u^*} - x_{v^*})^2\cdot(1-\epsilon/7) \ge \epsilon\cdot R^{\max}(e)\cdot(1-\epsilon/7)/(12\gamma\cdot n) \ge \epsilon\cdot R^{\max}(e)/(14\gamma\cdot n)$. Therefore, $e \in E'_I$, which contradicts $e \in E_{I+1}$.

We show that $|\widetilde{Q}_{E_{I+1}} - Q_{E_{I+1}}|$ is small by showing that each of the two terms is individually small. First,

$$Q_{E_{I+1}}(x) = \sum_{e\in E_{I+1}} w(e)\cdot \max_{u,v\in e}(x_u - x_v)^2 \le \sum_{e\in E_{I+1}} w(e)\cdot\frac{\epsilon\cdot R^{\max}(e)}{12\gamma\cdot n} \le \frac{\epsilon}{12\gamma\cdot n}\cdot\sum_{e\in E} w(e)\cdot R^{\max}(e) \le \frac{\epsilon}{6},$$

where the last inequality uses Eq. (3.12). Second, we start similarly,

$$\widetilde{Q}_{E_{I+1}}(x) = \sum_{e\in E_{I+1}} \widetilde{w}(e)\cdot \max_{u,v\in e}(x_u - x_v)^2 \le \sum_{e\in E_{I+1}} \widetilde{w}(e)\cdot\frac{\epsilon\cdot R^{\max}(e)}{12\gamma\cdot n} \le \frac{\epsilon}{12\gamma\cdot n}\cdot\sum_{e\in E} \widetilde{w}(e)\cdot R^{\max}(e),$$

and ideally we would like to show that $\sum \widetilde{w}(e)\cdot R^{\max}(e) \le 4\gamma\cdot n$. This is a random event, independent of the choice of x, whose probability we can bound using the Chernoff bound (Theorem 3.2.10). Recall that $E_0 = \{e\in E \mid p(e) = 1\}$, and denote its complement by $\overline{E}_0 := \{e \in E \mid p(e) = \lambda\cdot w(e)\cdot R^{\max}(e)\}$. Since $\mathbb{E}(\sum \widetilde{w}(e)\cdot R^{\max}(e)) = \sum w(e)\cdot R^{\max}(e) \le 2\gamma\cdot n$ (using Eq. (3.12)) and $\sum_{e\in E_0}\widetilde{w}(e)\cdot R^{\max}(e)$ is deterministic by definition of E_0, we have that

$$\begin{aligned}\mathbb{P}\left(\sum_{e\in E}\widetilde{w}(e)\cdot R^{\max}(e) \ge 4\gamma\cdot n\right) &\le \mathbb{P}\left(\left|\sum_{e\in E}\widetilde{w}(e)\cdot R^{\max}(e) - \mathbb{E}\left(\sum_{e\in E}\widetilde{w}(e)\cdot R^{\max}(e)\right)\right| \ge 2\gamma\cdot n\right)\\ &= \mathbb{P}\left(\left|\sum_{e\in \overline{E}_0}\widetilde{w}(e)\cdot R^{\max}(e) - \mathbb{E}\left(\sum_{e\in \overline{E}_0}\widetilde{w}(e)\cdot R^{\max}(e)\right)\right| \ge 2\gamma\cdot n\right).\end{aligned}$$

We bound this by applying Theorem 3.2.10 for the independent random variables $\widetilde{w}(e)\cdot R^{\max}(e)$ where $e\in\overline{E}_0$. The maximum value of such a variable occurs when e is sampled, in which case it is exactly $\widetilde{w}(e)\cdot R^{\max}(e) = R^{\max}(e)\cdot w(e)/p(e) = 1/\lambda \le 1 =: a$. Setting $\delta := 1$ and $\mu := 2\gamma\cdot n$ (we may do this due to Eq. (3.12)), we get

$$\mathbb{P}\left(\sum_{e\in E}\widetilde{w}(e)\cdot R^{\max}(e) \ge 4\gamma\cdot n\right) \le 2\exp\left(-\frac{2\gamma\cdot n}{3}\right) \le \frac{1}{n}.$$

In conclusion, with probability at least $1-O(1/n)$,

$$\left|\widetilde{Q}_{E_{I+1}}(x)-Q_{E_{I+1}}(x)\right|\le\frac{\epsilon}{2}. \tag{3.17}$$

Putting everything together. The final part of the proof proceeds identically to that of Theorem 3.4.1. We have seen that Eq. (3.16) holds simultaneously for all i and $(x^{(i)},E_i)$, as well as Eq. (3.17) holds with probability at least $1-O(\log(n)/n)$. Conditioning on these events, we can deduce Eq. (3.13), thus concluding the proof of Lemma 3.6.2.

For completeness we repeat the derivation:

$$\begin{aligned}
\left|Q_{E_i}(x)-\widetilde{Q}_{E_i}(x)\right| &\le \left|Q_{E_i}(x)-Q_{E_i}(x^{(i)})\right|+\left|Q_{E_i}(x^{(i)})-\widetilde{Q}_{E_i}(x^{(i)})\right|+\left|\widetilde{Q}_{E_i}(x^{(i)})-\widetilde{Q}_{E_i}(x)\right| \\
&\le \frac{\epsilon}{7}\cdot Q_{E_i}(x^{(i)})+\left|Q_{E_i}(x^{(i)})-\widetilde{Q}_{E_i}(x^{(i)})\right|+\frac{\epsilon}{7}\cdot\widetilde{Q}_{E_i}(x^{(i)}) && \text{(by Corollary 3.6.4)} \\
&\le \frac{2\epsilon}{7}\cdot Q_{E_i}(x^{(i)})+\left(1+\frac{\epsilon}{7}\right)\cdot\left|Q_{E_i}(x^{(i)})-\widetilde{Q}_{E_i}(x^{(i)})\right| \\
&\le \frac{2\epsilon}{7}\cdot\left(1-\frac{\epsilon}{7}\right)^{-1}\cdot Q_{E_i}(x)+\left(1+\frac{\epsilon}{7}\right)\cdot\frac{\epsilon\cdot Q(x)}{7I} && \text{(by Corollary 3.6.4 and Eq. (3.16))} \\
&\le \frac{\epsilon}{3}\cdot Q_{E_i}(x)+\frac{\epsilon\cdot Q(x)}{6I}.
\end{aligned}$$

Therefore, we have

$$\begin{aligned}
\left|Q(x)-\widetilde{Q}(x)\right| &\le \left|Q_{E_0}(x)-\widetilde{Q}_{E_0}(x)\right|+\sum_{i=1}^{I}\left|Q_{E_i}(x)-\widetilde{Q}_{E_i}(x)\right|+\left|Q_{E_{I+1}}(x)-\widetilde{Q}_{E_{I+1}}(x)\right| \\
&\le 0+\sum_{i=1}^{I}\left(\frac{\epsilon}{3}\cdot Q_{E_i}(x)+\frac{\epsilon\cdot Q(x)}{6I}\right)+\frac{\epsilon}{2} && \text{(by Eq. (3.17))} \\
&\le \frac{\epsilon}{3}\cdot Q(x)+\frac{\epsilon}{6}\cdot Q(x)+\frac{\epsilon}{2} \\
&\le \epsilon\cdot Q(x), && \text{(by Eq. (3.14))}
\end{aligned}$$

as claimed. □

Combining Theorem 3.5.10 and Lemmas 3.6.1 and 3.6.2, we get the first part of Theorem 1.

3.7 Nearly Optimal Speed-Up

In the previous section, we have proved the existence of nearly linear sized spectral sparsifiers for hypergraphs. We have also provided an method for constructing such sparsifiers: we construct an approximate balanced weight assignment of the input hypergraph using Algorithm 4, and then construct a sparsifier using Algorithm 5. However, the running time of Algorithm 4 on an unweighted hypergraph is $m\cdot\mathrm{poly}(n)$, which is large; in this section we improve this to $\widetilde{O}(mr)+\mathrm{poly}(n)$, that is, we prove the second part of Theorem 1. As we mentioned in the introduction, with a small modification, this leads to an algorithm with time complexity $\widetilde{O}(\sum_{e\in E}|e|+\mathrm{poly}(n))$ that constructs an ϵ-spectral-sparsifier of nearly linear size. This running time is optimal to within polylogarithmic factors in n, unless the size of the input hypergraph is polynomially small in n.

Our algorithm consists of two steps. First we apply the algorithm of Bansal et al. (2019), which—with small modifications—can be shown to run in $\widetilde{O}(mr) + \text{poly}(n)$ time, but which produces a larger spectral sparsifier. We then aim to sparsify the resulting weighted hypergraph using our methods. Unfortunately, even though the resulting hypergraph has only polynomially many hyperedges (in n), the range of edge weights may still be exponential, meaning that Algorithm 4 is not efficient for finding an approximate balanced weight assignment on it (recall Theorem 3.5.10). We propose a variation of Algorithm 4 adapted for this setting which runs in polynomial time.

3.7.1 Fast Algorithm for Constructing Polynomial-Sized Sparsifiers

In this section, we recall and slightly modify the algorithm of Bansal et al. (2019) for producing polynomial-sized spectral sparsifiers for hypergraphs.

Definition 3.7.1. *For a weighted hypergraph $H = (V, E)$, let $G(H)$ denote the "associated graph" of H, which is defined as follows: Replace each hyperedge e of E with a clique of uniform weight $w(e)$ on the support of e. (Note that this may produce parallel edges.)*

Theorem 3.7.2 (Bansal et al. (2019))**.** *Let $H = (V, E, w)$ be a hypergraph where all hyperedges have size between $r/2$ and r. Then, for some absolute constant c, the following process produces an ϵ-spectral sparsifier for H with probability at least $1 - 1/n$: Let $G(H)$ be the associated graph of H. For each hyperedge $e \in E$, let*

$$R^{\max}(e) = \max_{u,v \in e} R_{G(H)}(u, v).$$

Sample each hyperedge e independently with some probability

$$p(e) \geq \min\left(1, \frac{w(e) \cdot R^{\max}(e) \cdot r^4 \log n}{c\epsilon^2}\right), \tag{3.18}$$

and if sampled give it weight $\widetilde{w}(e) = w(e)/p(e)$.

Remark 3.7.3. *In fact, in Bansal et al. (2019), the result is stated slightly less generally, for unweighted hypergraphs, and without allowing oversampling (that is $p(e)$ is set* exactly *to the right hand side in Eq. (3.18), instead of being lower bounded by it). However, this version holds by an identical proof.*

The trivial implementation of this takes time $\Omega(mr^2)$ in general for two different reasons: First, it takes $\Omega(mr^2)$ time to replace each of the m hyperedges with a clique of size r^2. Second, it takes $\Omega(mr^2)$ time to find $\max_{u,v \in e} R_{G(H)}(u, v)$ for all m hyperedges. However, with some simple tricks in the implementation both bottlenecks can be avoided to reduce the running time to $\widetilde{O}(mr + \text{poly}(n))$.

To achieve this, we first replace cliques in the associated graph of the input hypergraph with sparse graphs guaranteed by the following fact:

Fact 3.7.4. *It follows by Theorem 3.4.1 that for every r, there exists a (weighted) graph G_r^* with r vertices and $\widetilde{O}(r)$ edges such that G_r^* is a $1/2$-spectral sparsifier to the r-clique.*

Second, to calculate $R^{\max}(e)$ approximately, we do not take the maximum over all pairs of vertices in e, but only over (u_0, v) for all $v \in e$ but for some fixed u_0. Since effective resistance is a metric (see Fact 3.2.7), this provides a 2-approximation to $R^{\max}(e)$ by triangle inequality.

Algorithm 6 Fast algorithm for computing a polynomial-sized spectral sparsifier for an approximately uniform hypergraph.

```
1:  procedure UniformSparsification($H = (V, E, w), r, \epsilon$)
2:      $G = (V, F, z) \leftarrow (V, \emptyset, 0)$
3:      for all $e \in E$ do
4:          Add a copy of $G^*_{|e|}$ to $G$, supported on $e$, with weights scaled up by $w(e)$
5:      Calculate $R_G(u, v)$ for all $u, v \in V$
6:      $\widetilde{H} = (V, \widetilde{E}, \widetilde{w}) \leftarrow (V, \emptyset, 0)$
7:      for all $e \in E$ do
8:          $u_0 \leftarrow$ an arbitrary vertex in $e$
9:          $\widetilde{R}^{\max}(e) \leftarrow \max_{v \in e} R_G(u_0, v)$
10:         $p(e) \leftarrow \min\left(1, \frac{4w(e) \cdot \widetilde{R}^{\max}(e) \cdot r^4 \log n}{c\epsilon^2}\right)$
11:         Add $e$ to $\widetilde{E}$ with probability $p(e)$ with weight $\widetilde{w}(e) \leftarrow w(e)/p(e)$
12:     return $\widetilde{H}$
```

Lemma 3.7.5. *If the input hypergraph $H = (V, E, w)$ only has hyperedges of size between $r/2$ and r, then Algorithm 6 runs in $\widetilde{O}(mr) + \mathrm{poly}(n)$ time, returning an ϵ-spectral sparsifier to H with probability at least $1 - 1/n$. Furthermore, the output has at most $4n^5 \log n/(c\epsilon^2)$ hyperedges in expectation, where c is the absolute constant from Theorem 3.7.2.*

Proof. It takes $\widetilde{O}(mr)$ time to construct the graph G (here Line 4 takes $\widetilde{O}(r)$ time) and its Laplacian. All pairs effective resistances can then be calculated in $\mathrm{poly}(n)$ time and stored in a table, resulting in an $O(r)$ time bound for the calculation of $\widetilde{R}^{\max}(e)$ in Line 9.

To show that the output is an ϵ-spectral sparsifier of H with high probability, it suffices to verify Eq. (3.18) of Theorem 3.7.2, ie. that $p(e)$ is always at least

$$w(e) \cdot \max_{u,v \in e} R_{G(H)}(u, v) \cdot \frac{r^4 \log n}{c\epsilon^2}.$$

For this, it suffices to show that $\widetilde{R}^{\max}(e)$ (as defined in Line 9 of Algorithm 6) is at least $\max_{u,v \in e} R_{G(H)}(u, v)/4$. Since $w(e) \cdot G^*_{|e|}$ is a $1/2$-spectral sparsifier of the clique on e (of uniform edge weight $w(e)$), it follows by the additivity of Laplacians that G from Algorithm 6 is a $1/2$-spectral sparsifier of $G(H)$ from Definition 3.7.1. Therefore, $R_G(u, v) \geq R_{G(H)}(u, v)/2$ for all $u, v \in V$ (recall Definition 3.2.5). Finally, since R_G is a metric on V (by Fact 3.2.7),

$$\max_{v \in e} R_G(u_0, v) \geq \max_{u,v \in e} R_G(u, v)/2.$$

Indeed, if (u^*, v^*) maximizes $R_G(u, v)$, then by triangle inequality $R_G(u^*, v^*) \leq R_G(u^*, u_0) + R_G(u_0, v^*)$; one of the latter two must be at least $R_G(u^*, v^*)/2$.

This concludes the proof of correctness; we must finally prove the upper bound on the expected size of the output $\widetilde{H}$. Since $r^4 \le n^4$, it suffices to show that $\sum_{e\in E} w(e)\cdot \widetilde{R}^{\max}(e) \le n$. First note that

$$\sum_{e\in E} w(e)\cdot \widetilde{R}^{\max}(e) \le \sum_{e\in E} w(e)\cdot \max_{u,v\in e} R_G(u,v) \le 2\sum_{e\in E} w(e)\cdot R_{G(H)}(u,v),$$

since G is a $1/2$-spectral sparsifier of $G(H)$. Then, since the weight of (u,v) in $G(H)$ is exactly $w(e)$ by definition, we have that

$$\sum_{e\in E} w(e)\cdot \widetilde{R}^{\max}(e) \le \sum_{f\in F(H)} z_{G(H)}(f)\cdot R_{G(H)}(f) = n-1,$$

by Fact 3.2.9, where $F(H)$ denotes the edge-set of $G(H)$ and $z_{G(H)}$ denotes the weight function of $G(H)$. □

Algorithm 6 only works under the constraint that the input hypergraph is approximately uniform – that is, the sizes of hyperedges all fall into the range $[r/2, r]$. This is easily circumvented, however: Given an arbitrary input hypergraph, one can simply partition the hyperedges into a logarithmic number of parts based on cardinality. We then sparsify the parts, and combine the resulting sparsifiers, losing only a $\log n$ factor in size.

We formalize this in the following algorithm and corollary.

Algorithm 7 Fast algorithm for computing a polynomial-sized spectral sparsifier for an arbitrary hypergraph.

```
1: procedure PolynomialSizeSparsification($H=(V,E,w),\epsilon$)
2:     for $i$ from 1 to $\log n$ do
3:         $E_i \leftarrow \{e\in E \,\big|\, |e|\in[2^i,2^{i+1})\}$
4:         $H_i \leftarrow (V,E_i,w)$
5:         $\widetilde{H}_i \leftarrow$ UniformSparsification($H_i, 2^{i+1}, \epsilon$)          ▷ Algorithm 6
6:     return $\widetilde{H} \leftarrow \cup_{i=1}^{\log n}\widetilde{H}_i$
```

Here the final line means that we take all hyperedges (with associated weights) from all of $\widetilde{H}_1,\dots,\widetilde{H}_{\log n}$.

Corollary 3.7.6. *Algorithm 7 runs in $\widetilde{O}(mr)+\mathrm{poly}(n)$ time, returning an ϵ-spectral sprasifier of the input H with probability at least $1-\log n/n$. Furthermore, the output has at most $4n^5\log^2(n)/(c\epsilon^2)$ hyperedges in expectation, where c is the absolute constant from Theorem 3.7.2.*

3.7.2 Even Faster Construction for γ-Balanced Weight Assignments

It is difficult to get a stronger bound on the number of rounds of Algorithm 4, than that of Theorem 3.5.10, at least in its full generality. Instead, here we define a specific class of hypergraphs for which a better algorithm exists.

Definition 3.7.7. *A weighted hypergraph $H=(V,E,w)$ is called* (α,β)-separated *for parameters $\alpha\ge 1$ and $\beta\ge 1$ if the hyperedge set E is partitioned into $E_1,\dots,E_\ell$, satisfying the two requirements:*

$$\text{for all } i\in\{1,2,\dots,\ell\}, \qquad \max_{e\in E_i} w(e) \le \alpha\cdot \min_{e\in E_i} w(e),$$

$$\text{for all } i, j \in \{1, 2, \ldots, \ell\}, i < j, \qquad \min_{e \in E_i} w(e) \geq \beta \cdot \max_{e \in E_j} w(e).$$

Our next algorithm exploits the structure of separated hypergraphs in order to more efficiently construct approximate balanced weight assignments on them. Intuitively, one can think of the different weight classes in separated hypergraphs as only weakly interacting, which is the source of our speedup. In particular, it is important to note that we will get a speedup for hypergraphs where the total number of hyperedges is small in comparison to the amount of separation between the weight classes (i.e., hypergraphs that are produced by the sparsification procedure from the previous section)—this is what ultimately ensures that the different weight classes only interact in a limited manner.

Algorithm 8 Computing an η-approximate γ-balanced weight assignment for an (α, β)-separated hypergraph.

```
1:  procedure SeparatedApproxBalancing($H = (V, E, w), (E_i)_{i=1}^{\ell}, \alpha, \beta, \gamma, \eta$)
2:      For all $e \in E$ and for all $f \in F_e$ initialize $z(f) \leftarrow w(e) / \binom{|e|}{2}$
3:      $G \leftarrow (V, \bigcup F_e, z)$
4:      for $i = 1, \ldots, \ell$ do
5:          while there exists $e \in E_i$ violating the conditions of $\eta$-approximate $\gamma$-balancedness do
6:              Select such $e \in E_i$ and $f, g \in F_e$, such that $R_G(f) > \gamma \cdot R_G(g)$ and $z(g) > \eta \cdot w(e)$
7:              $\lambda \leftarrow \min\left(z(g), 1/(2R_G(g))\right)$
8:              $z(f) \leftarrow z(f) + \lambda$
9:              $z(g) \leftarrow z(g) - \lambda$
10:     return $G$.
```

In fact, Algorithm 8 is very similar to Algorithm 4. However, it corrects discrepancies in heavier hyperedges first, and once a category of hyperedges (E_i) has been corrected, the algorithm never goes back to it, *not even if the approximate balancedness becomes violated.* For this reason, the resulting output, G, will not necessarily be γ-balanced. However, the structure of separated hypergraphs will allow us to bound the number of rounds using α, instead of the global aspect ratio of weights, which could be much larger.

The crucial property of separated hypergraphs (and their weight assignments) which we exploit is that the heavier edges have a much greater influence on the effective resistance of a vertex pair than the lighter edges. More specifically, for some $i \in [\ell]$ we can define the subgraph G_+ containing only edges in F_e for $e \in E_j$ where $j \leq i$. Then, when calculating the effective resistance $R_G(u, v)$—under certain circumstances—we can simply calculate $R_{G_+}(u, v)$ instead, ignoring the contribution of the remaining edges. We formalize this in the following lemma and its corollary.

Lemma 3.7.8. *Let $G_+ = (V, E_+, z_+)$ and $G_- = (V, E_-, z_-)$ be two weighted ordinary graphs on the same vertex set. Let the total weight of all edges in G_- be bounded by ζ. Let $G = G_+ \cup G_-$ be the union of the two graphs, that is $G = (V, E_+ \cup E_-, z_+ \cup z_-)$. Then, for any vertex pair $(u, v) \in V$ in the same connected component of G_+, we can bound the effective resistance $R_G(u, v)$ in terms of R_{G_+} as follows:*

$$\frac{1}{R_G(u, v)} \leq \frac{1}{R_{G_+}(u, v)} + \zeta.$$

Proof. To prove this inequality, we use the alternate definition of effective resistance from Fact 3.2.6. That is

$$R_{G_+}(u,v) = \max_{x\in\mathbb{R}^V} \frac{(x_u - x_v)^2}{x^\top L_{G_+} x}.$$

Let x^* maximize the above formula. We may assume without loss of generality that $x^*_u = 0$ and $x^*_v = 1$ since the formula is both shift and scale invariant. From this it follows that $x^{*\top} L_{G_+} x^* = 1/R_{G_+}(u,v)$. We can further assume without loss of generality that $x^*_a \in [0,1]$ for all $a \in V$. Indeed, let the connected component of u and v in G_+ be $C \subseteq V$. Then for $a \in V \setminus C$, we may simply assume that x^*_a is always 0. To see that $x^*_a \in [0,1]$ for all $a \in C$, suppose for contradiction that $\max_{a\in C} x^*_a > 1$. Let the highest value of x^* in C be $\mu_1 > 1$ and the second highest distinct value be μ_2. Then one can change the x^*-value of vertices from μ_1 to μ_2, thereby strictly increasing the value of $(x^*_u - x^*_v)^2 / x^{*\top} L_{G_+} x^*$. This is a contradiction, since x^* maximizes the formula by definition. An identical argument by contradiction rules out that $\min_{a\in C} x^*_a < 0$.

We can now upper bound $1/R_G(u,v)$ using the same alternate definition from Fact 3.2.6:

$$R_G(u,v) = \max_{x\in\mathbb{R}^V} \frac{(x_u - x_v)^2}{x^\top L_G x} \geq \frac{(x^*_u - x^*_v)^2}{x^{*\top} L_G x^*} = \frac{1}{1/R_{G_+}(u,v) + x^{*\top} L_{G_-} x^*}.$$

Now

$$x^{*\top} L_{G_-} x^* = \sum_{(a,b)\in E_-} z_-(a,b) \cdot (x^*_a - x^*_b)^2 \leq \sum_{(a,b)\in E_-} z_-(a,b) \leq \zeta.$$

Therefore

$$\frac{1}{R_G(u,v)} \leq \frac{1}{R_{G_+}(u,v)} + \zeta,$$

as claimed. □

Corollary 3.7.9. *Let $G_+ = (V, E_+, z_+)$ and $G_- = (V, E_-, z_-)$ be two weighted ordinary graphs on the same vertex set. Let the total weight of all edges in G_- be bounded by ζ. Let $G = G_+ \cup G_-$ be the union of the two graphs, that is $G = (V, E_+ \cup E_-, z_+ \cup z_-)$. Then for any vertex pair $u, v \in V$, if $R_G(u,v) \leq 1/(5\zeta)$ then we can bound the effective resistance $R_G(u,v)$ in terms of $R_{G_+}(u,v)$ as follows:*

$$R_G(u,v) \geq \frac{4}{5} \cdot R_{G_+}(u,v).$$

Proof. We wish to apply Lemma 3.7.8, for which we must first show that u and v are in the same connected component in G_+. This is indeed the case: If u and v are in different connected components of G_+, we can use the alternate definition of effective resistance from Fact 3.2.6:

$$R_G(u,v) = \max_{x\in\mathbb{R}^V} \frac{(x_u - x_v)^2}{x^\top L_G x} \geq \frac{(x^0_u - x^0_v)^2}{x^{0\top} L_G x^0},$$

where x^0 is 0 on the connected component of u in G_+, and 1 everywhere else (including v). Now

$$\frac{(x^0_u - x^0_v)^2}{x^{0\top} L_G x^0} = \frac{1}{x^{0\top} L_{G_+} x^0 + x^{0\top} L_{G_-} x^0} = \frac{1}{x^{0\top} L_{G_-} x^0} = \frac{1}{\sum_{(a,b)\in E_-} z_-(a,b) \cdot (x^0_a - x^0_b)^2} \geq \frac{1}{\zeta}.$$

This is a contradiction; therefore, u and v are indeed in the same connected component of G_+.

We can now apply Lemma 3.7.8:

$$\frac{1}{R_{G_+}(u,v)} \geq \frac{1}{R_G(u,v)} - \zeta \geq \frac{4}{5} \cdot \frac{1}{R_G(u,v)},$$

as claimed. □

We are now ready to analyze Algorithm 8.

Theorem 3.7.10. *Let $H = (V, E, w)$ be an (α, β)-separated weighted hypergraph with $\beta \geq 5|E| \cdot \gamma/\eta$. Let $M = w(E)$ be the total hyperedge weight in H, and let $w_{\min}$ the minimum hyperedge weight. Then for $\gamma \geq 4$, Algorithm 8 terminates* within $\alpha/\eta \cdot |E|^2 \cdot \operatorname{poly}\left(n \log(M/w_{\min})\right)$ rounds *and returns an η-approximate 2γ-balanced weight assignment for H.*

Proof. We call executions of the while-loop rounds and executions of the for-loop phases. It will be useful to denote $w_{i,\min} := \min_{e \in E_i} w(e)$ and $w_{i,\max} := \max_{e \in E_i} w(e)$. The definition of separated hypergraphs (Definition 3.7.7) then ensures that $w_{i,\max} \leq \alpha \cdot w_{i,\min}$ and $w_{i,\min} \geq \beta \cdot w_{i+1,\max}$.

Algorithm Correctness. We first prove that upon termination, Algorithm 8 indeed returns an η-approximate 2γ-balanced weight assignment. Consider a hyperedge e and edges $f, g \in F_e$, with $z(g) \geq \eta \cdot w(e)$. We will show that $R_{G^*}(f) \leq 2\gamma \cdot R_{G^*}(g)$, where G^* is the final weight assignment returned by the algorithm at Line 10. This is exactly what is required by the definition of approximate balancedness, i.e. Definition 3.5.9. Suppose $e \in E_i$. Note that by the condition of the while-loop, at the termination of the i^{th} phase, e, f, and g satisfied even the stronger condition of η-approximate γ-balancedness. Let us denote by G' the weight assignment graph at the end of phase i.

Let us partition the edges of G^* and G' based on the weight of the corresponding hyperedge. Denote $E_{\leq i} := \bigcup_{j=1}^{i} E_j$ and $E_{>i} := \bigcup_{j=i+1}^{\ell} E_j$. Similarly, let $F_{\leq i} := \bigcup_{e \in E_{\leq i}} F_e$ and $F_{>i} := \bigcup_{e \in E_{>i}} F_e$. Finally, let $G^*_{\leq i}$ and $G^*_{>i}$ be G^* restricted to $F_{\leq i}$ and $F_{>i}$ respectively, and define $G'_{\leq i}$ and $G'_{>i}$ similarly. Note that $G^*_{\leq i} = G'_{\leq i}$, since the algorithm never alters the weight assignments of $E_{\leq i}$ after phase i.

From here, we will use two applications of Corollary 3.7.9. First, set $G_+ := G'_{\leq i}$, $G_- := G'_{>i}$ (which makes $G = G'$), and $(u, v) := f$. Then ζ is the total weight of all edges in $G'_{>i}$, which is the total weight of all hyperedges in $E_{>i}$, which is at most $w_{i+1,\max} \cdot |E|$. We verify the condition on $R_{G'}(f)$ from Corollary 3.7.9: $R_{G'}(f) \leq \gamma \cdot R_{G'}(g)$, by the approximate γ-balancedness condition on e in G'. Since $z(g) \geq \eta \cdot w(e)$ (in both G^* and G'), we have that $z(g) \geq \eta \cdot w_{i,\min}$, and hence by Fact 3.2.8 $R_{G'}(g) \leq 1/(\eta \cdot w_{i,\min})$. Finally, putting these together, as well as using the (α, β)-separated quality of H, we get

$$R_{G'}(f) \leq \frac{\gamma}{\eta \cdot w_{i,\min}} \leq \frac{\gamma}{\eta\beta \cdot w_{i+1,\max}} \leq \frac{1}{5\zeta},$$

by assumption on β. We can indeed apply Corollary 3.7.9, which gives us

$$R_{G'}(f) \geq \frac{4}{5} \cdot R_{G_{\leq i}}(f). \tag{3.19}$$

We apply Corollary 3.7.9 again, this time setting $G_+ := G^*_{\le i}$, $G_- := G^*_{>i}$ (which makes $G = G^*$), and $(u, v) := g$. Then $\zeta \le w_{i+1,\max} \cdot |E|$ by an identical argument to the previous setting. We verify the condition on $R_{G^*}(g)$ from Corollary 3.7.9: $R_{G^*}(g) \le 1/(\eta \cdot w_{i,\min})$ by Fact 3.2.8, since $z(g) \ge \eta \cdot w(e) \ge \eta \cdot w_{i,\min}$ by assumption (in both G^* and G'). Therefore, by an identical derivation to the previous case $R_{G^*}(g) \le 1/(5\zeta)$. We can indeed apply Corollary 3.7.9, which gives us

$$R_{G^*}(g) \ge \frac{4}{5} \cdot R_{G_{\le i}}(g). \tag{3.20}$$

Putting together Equations Eq. (3.19) and Eq. (3.20), along with the fact that f and g satisfied the condition of approximate γ-balancedness at the time of G' (that is at the end of phase i), we get

$$R_{G^*}(f) \le R_{G_{\le i}}(f) \le \frac{5}{4} \cdot R_{G'}(f) \le \frac{5\gamma}{4} \cdot R_{G'}(g) \le \frac{5\gamma}{4} \cdot R_{G_{\le i}}(g) \le \frac{25\gamma}{16} \cdot R_{G^*}(g) \le 2\gamma \cdot R_{G^*}(g),$$

which concludes the proof that G^* is η-approximately 2γ-balanced.

Running Time. Next, we prove that Algorithm 8 terminates within $\alpha/\eta \cdot |E|^2 \cdot \mathrm{poly}\left(n \log(M/w_{\min})\right)$ rounds. Similarly to the proof of Theorem 3.5.10, we can bound the total growth of the ST-potential over the course of the algorithm by defining G_0 and G_∞. Specifically, if G_0 is the starting graph of Algorithm 8 and G_∞ is the complete graph of uniform edge weight M, then as before,

$$\Psi(G_\infty) - \Psi(G_0) \le \mathrm{poly}\left(n \log(M/w_{\min})\right).$$

We will now prove that the total number of rounds in the ith phase is at most $|E_i| \cdot \alpha/\eta \cdot |E| \cdot \mathrm{poly}\left(n \log(M/w_{\min})\right)$. Due to Lemma 3.5.7, the contribution of each update to the potential $\Psi(G)$ is at least

$$\log\left(1 + \lambda\gamma \cdot R_G(g) - \lambda \cdot R_G(g) - \lambda^2\gamma \cdot R_G(g)^2\right).$$

Similarly to the proof of Theorem 3.5.10, this either means that $\lambda = 1/(2R_G(g))$ and therefore the increase to Ψ is at least $\log(5/4)$, or that $\lambda = z(g) \le 1/(2R_G(g))$, in which case the increase to Ψ is at least $\log(1 + z(g) \cdot R_G(g))$. In the latter case, we further distinguish based on the value of $R_G(g)$.

1. $\lambda = 1/(2R_G(g))$. In this case the ST-potential increases by at least $\log(5/4)$, and so there can be at most $\mathrm{poly}\left(n \log(M/w_{\min})\right)$ such rounds.

2. $\lambda = z(g)$ and $R_G(g) > 1/(5w_{i,\max} \cdot |E|)$. In this case the ST-potential increases by at least $\log(1 + z(g) \cdot R_G(g))$. Since g was selected in this round, $z(g) \ge \eta \cdot w(e) \ge \eta \cdot w_{i,\min}$, and therefore the increase in $\Psi(G)$ is at least

$$\log(1 + z(g) \cdot R_G(g)) \ge \log\left(1 + \frac{\eta \cdot w_{i,\min}}{5w_{i,\max} \cdot |E|}\right) \ge \log\left(1 + \frac{\eta}{5\alpha \cdot |E|}\right) \ge \frac{\eta}{10\alpha \cdot |E|}.$$

   Thus, there can be at most $\alpha/\eta \cdot |E| \cdot \mathrm{poly}\left(n \log(M/w_{\min})\right)$ such rounds.

3. $\lambda = z(g)$ and $R_G(g) \le 1/(5w_{i,\max} \cdot |E|)$. This is the most complicated case to analyze and we will be

focusing on this for the rest of the proof.

Suppose we are in the i^{th} phase of Algorithm 8 and we have $e \in E_i$ and $f, g \in F_e$ such that $R_G(f) > \gamma \cdot R_G(g) \geq 4 \cdot R_G(g)$. Further suppose that $R_G(g) \leq 1/(5w_{i,\max} \cdot |E|)$. Similarly to before, we divide G into two graphs: $G_{<i}$ consisting of edges belonging to hyperedges from E_j for $j < i$, and $G_{\geq i}$ consisting of edges belonging to hyperedges from E_j for $j \geq i$. (Note that earlier in this proof we used the slightly different split into $G_{\leq i}$ and $G_{>i}$.) We can again use Corollary 3.7.9 to relate R_G to $R_{G_{<i}}$.

We apply Corollary 3.7.9, setting $G_+ := G_{<i}$, $G_- := G_{\geq i}$ (which makes G from Corollary 3.7.9 the current graph G), and $(u, v) := g$. Then ζ is the total weight of all edges in $G_{\geq i}$, which is the total weight of all hyperedges in $E_{\geq i}$, which is at most $w_{i,\max} \cdot |E|$. Our above assumption $R_G(g) \leq 1/(5w_{i,\max} \cdot |E|)$ exactly guarantees that $R_G(g) \leq 1/(5\zeta)$ is satisfied and we can therefore apply Corollary 3.7.9:

$$R_{G_{<i}}(g) \leq \frac{5}{4} \cdot R_G(g) \leq \frac{5}{4\gamma} R_G(f) \leq \frac{5}{16} R_G(f) \leq \frac{5}{16} R_{G_{<i}}(g) < R_{G_{<i}}(f).$$

In words, such weight transfers are always from edges of lower $G_{<i}$-resistance to those of strictly higher $G_{<i}$-resistance—a metric that never changes, since $G_{<i}$ is unchanged during the i^{th} stage of the algorithm. Therefore, there cannot be $n^4 \cdot |E_i|$ consecutive steps of type 3 in phase i. We prove this formally in the following claim.

Claim 3.7.11. *There cannot be $n^4 \cdot |E_i|$ consecutive rounds of type 3 in phase i.*

Proof. Suppose for contradiction that there is a sequence of $n^4 \cdot |E_i|$ rounds of type 3. Then there must be a hyperedge $e \in E_i$ for which at least n^4 of these updates take place in F_e. We know that updates can only shift weight from g to f where $R_{G_{<i}}(f) > R_{G_{<i}}(g)$. Therefore, let us order the edges of F_e by $R_{G_{<i}}$, that is, let $\rho : F_e \to \mathbb{N}$ be such that the j^{th} largest edge in terms of $R_{G_{<i}}$, say f, has $\rho(f) = j$ (breaking ties arbitrarily). Then we can define a local potential function for e:

$$\psi_e(G) = \sum_{f \in F_e} \rho(f) \cdot \mathbb{1}(z(f) > 0).$$

Note that ψ_e starts out as at most $1 + 2 + \cdots + |F_e| \leq n^4$, and after every update of type 3 it decreases by at least one. Since we have no updates of any type other than 3 (by assumption), and updates to other hyperedges do not affect ψ_e, we arrive at the contradiction that ψ_e must become negative. □

As a result of Claim 3.7.11, every consecutive sequence of $n^4 \cdot |E_i|$ rounds must contain an update of type 1 or 2. Since we showed that there are at most $\alpha/\eta \cdot |E| \cdot \operatorname{poly}\big(n \log(M/w_{\min})\big)$ such updates, there can be at most $|E_i| \cdot \alpha/\eta \cdot |E| \cdot \operatorname{poly}\big(n \log(M/w_{\min})\big)$ rounds in phase i. Summing this over all phases, we get that there can be at most $\alpha/\eta \cdot |E|^2 \cdot \operatorname{poly}\big(n \log(M/w_{\min})\big)$ rounds overall, as claimed. □

3.7.3 Proof of the Second Part of Theorem 1

We are now ready to prove the second part of Theorem 1. Specifically, we prove that FASTSPARSIFICATION (Algorithm 9 below) provides the result of Theorem 1:

Algorithm 9 Algorithm constructing nearly linear-sized spectral sparsifier, in nearly linear time.

```
1:  procedure FastSparsification($H = (V, E, w)$)
2:      $H' = (V, E', w') \leftarrow$ PolynomialSizeSparsification($H, \epsilon/3$)    ▷ Algorithm 7
3:      for $i$ from 1 to $n$ do
4:          $E'_i \leftarrow \{e \in E'_i \mid w'(e) \in [n^{10(i-1)}, n^{10i})\}$
5:      $H'_1 \leftarrow (V, \bigcup_{j \in \{1,\dots,n\}, j \text{ odd}} E_i, w')$
6:      $H'_2 \leftarrow (V, \bigcup_{j \in \{1,\dots,n\}, j \text{ even}} E_i, w')$
7:      $G_1 \leftarrow$ SeparatedApproxBalancing($H'_1, (E_j)_{j \text{ odd}}, n^{10}, n^{10}, 4, 1/n^2$)    ▷ Algorithm 8
8:      $G_2 \leftarrow$ SeparatedApproxBalancing($H'_2, (E_j)_{j \text{ even}}, n^{10}, n^{10}, 4, 1/n^2$)
9:      $\lambda \leftarrow \Theta(\epsilon^{-4} \log^3 n)$
10:     $\widetilde{H}_1 \leftarrow$ Sparsification($H'_1, G_1, \epsilon/3, \lambda$)    ▷ Algorithm 5
11:     $\widetilde{H}_2 \leftarrow$ Sparsification($H'_2, G_2, \epsilon/3, \lambda$)
12:     return $\widetilde{H} \leftarrow \widetilde{H}_1 \cup \widetilde{H}_2$
```

In the final line $\widetilde{H} \leftarrow \widetilde{H}_1 \cup \widetilde{H}_2$ means that we take the union of hyperedges and weight functions from $\widetilde{H}_1$ and $\widetilde{H}_2$, since both are on the same vertex set V.

Proof of the second part of Theorem 1: Indeed, Algorithm 9 does exactly that.

First note a few observations about the steps of Algorithm 9: Notice that PolynomialSizeSparsification produces hyperedge weights only in the range $[1, n^{10n})$, so E' is partitioned into E'_i for i from 1 to n. Next, H'_1 and H'_2 are indeed (n^{10}, n^{10})-separated hypergraphs by the definitions in Lines 4 to 6. Furthermore, note that $\beta \geq 5|E| \cdot \gamma/\eta$ holds (as required by Theorem 3.7.10), since $\beta = n^{10}$, $\eta = 1/n^2$, and $|E| = o(n^8)$ by the size guarantee of Corollary 3.7.6. Finally, G_1 and G_2 are $1/n^2$-approximate 8-balanced weight assignments of H'_1 and H'_2 respectively, by Theorem 3.7.10.

Therefore, by Lemma 3.6.2, $\widetilde{H}_1$ and $\widetilde{H}_2$ are $\epsilon/3$-spectral sparsifiers of H'_1 and H'_2 respectively. By the additivity of the hypergraph quadratic form $\widetilde{H}$ is an $\epsilon/3$-spectral sparsifier of H'. Since H' is itself an $\epsilon/3$-spectral sparsifier of H, this means that $\widetilde{H}$ is an ϵ-spectral sparsifier of H, as claimed.

Moreover, by Lemma 3.6.1, $\widetilde{H}_1$ and $\widetilde{H}_2$ are both of size at most $O(n \log^3(n)/\epsilon^4)$ (since we set λ to be $\Theta(\log^3(n)/\epsilon^4)$), and therefore so is $\widetilde{H}$.

Finally, Algorithm 9 runs in time $\widetilde{O}(mr + \text{poly}(n))$. Indeed PolynomialSizeSparsification runs in time $\widetilde{O}(mr) + \text{poly}(n)$, as shown in Corollary 3.7.6; SeparatedApproxBalancing runs in time poly(n) by Theorem 3.7.10, since $\alpha = \text{poly}(n)$ and $M = \exp(O(n))$; Sparsification runs in time poly(n), since the input hypergraph is of poly(n) size. □

3.8 Lower Bounds

In this section we prove our space lower bound for an arbitrary compression of the cut structure of a hypergraph. In Section 3.8.1 we introduce string compression, and reprove the corresponding lower bound result for completeness. In Section 3.8.2 we construct our generic hard example in Theorem 3.8.9. We then stateCorollaries 3.8.11 to 3.8.13 which result from applying Theorem 3.8.9 to various specific

Ruzsa-Szemerédi graph constructions.

3.8.1 String Compression

A *string compression scheme (SCS)* is an algorithm for compressing a long string into a short string, such that any subset sum query can be answered with small additive error. Formally, we define it as follows.

Definition 3.8.1. *For positive integers ℓ, k and $\epsilon, g > 0$, a pair of functions* $\textsc{Encode} : \{0,1\}^\ell \to \{0,1\}^k$ *and* $\textsc{Decode} : \{0,1\}^k \times 2^{[\ell]} \to \mathbb{N}$ *is considered to be an (ℓ, k, ϵ, g)-SCS, if there exists a set of strings $\mathscr{G} \subseteq \{0,1\}^\ell$, such that the following holds.*

- $|\mathscr{G}| \geq g \cdot 2^\ell$.
- *For every string $s \in \mathscr{G}$ and every query $q \in 2^{[\ell]}$,* $\left|\textsc{Decode}(\textsc{Encode}(s), q) - |s \cap q|\right| \leq \epsilon\ell/2$.

Remark 3.8.2. *In general we use subsets of $[\ell]$ and elements of $\{0,1\}^\ell$ interchangeably. For instance, in the above definition, in $|s \cap q|$, s is considered as a set.*

Remark 3.8.3. *It is important that although a compression scheme may only work on a subset of strings ($\mathscr{G}$), it must work on* all *queries. In fact, it is trivial to answer almost all queries on all inputs, by simply outputting $|q| \cdot |s|/\ell$.*

The lower following lower bound on the space requirement of string compression schemes has been known, and appears, for example, in Dinur and Nissim (2003). We reprove it here for completeness.

Theorem 3.8.4. *Suppose* $(\textsc{Encode}, \textsc{Decode})$ *is an (ℓ, k, ϵ, g)-SCS, where $\epsilon \leq 1/10$. Then*

$$k \geq \frac{\log g + 3\ell/50}{\log 2} - 1.$$

Proof. We know that $\textsc{Encode}$ maps $\mathscr{G}$ into $\{0,1\}^k$. Therefore, by pigeonhole principle, there must be some set of inputs $\mathscr{G}_0$ of size at least $|\mathscr{G}| \cdot 2^{-k} \geq g \cdot 2^{\ell-k}$ that maps to the same output, say c_0. Let s_0 be an arbitrary string in $\mathscr{G}_0$.

Define $B_H(s_0, 2\epsilon\ell)$ as the ball of radius $2\epsilon\ell$ in Hamming distance around s_0, that is, the set of strings $s \in \{0,1\}^\ell$ such that the number of coordinates where s and s_0 differ is at most $2\epsilon\ell$.

Claim 3.8.5. $\mathscr{G}_0 \subseteq B_H(s_0, 2\epsilon\ell)$.

Proof. Suppose there exists $s \in \mathscr{G}_0 \backslash B_H(s_0, 2\epsilon\ell)$, that is s and s_0 differ on more than $2\epsilon\ell$ coordinates. Without loss of generality, we may assume that there are more than $\epsilon\ell$ coordinates where s_0 is 0 but s_1 is 1; let the set of such coordinates be q. By the definition of a string compression scheme

$$\textsc{Decode}(\textsc{Encode}(s_0), q) = \textsc{Decode}(c_0, q) \leq |s_0 \cap q| + \epsilon\ell/2 = \epsilon\ell/2,$$

but

$$\textsc{Decode}(\textsc{Encode}(s), q) = \textsc{Decode}(c_0, q) \geq |s \cap q| - \epsilon\ell/2 = |q| - \epsilon\ell/2 > \epsilon\ell/2.$$

This is a contradiction. □

Claim 3.8.6. $|B_H(s_0, 2\epsilon\ell)| < 2^\ell \cdot 2\exp\left(-\frac{\ell(1-4\epsilon)^2}{6}\right)$.

Proof. Indeed,

$$B_H(s_0, 2\epsilon\ell) = B_H(0^\ell, 2\epsilon\ell) = 2^\ell \cdot \mathbb{P}(w_H(x) \le 2\epsilon\ell),$$

where x is a uniformly random vector in $\{0,1\}^\ell$. By Chernoff's bound

$$\mathbb{P}\left[w_H(x) \le 2\epsilon\ell\right] \le \mathbb{P}\left[\left|w_H(x) - \frac{\ell}{2}\right| \ge \frac{\ell}{2}(1-4\epsilon)\right] \le 2\exp\left(-\frac{\ell(1-4\epsilon)^2}{6}\right),$$

since $\epsilon \le 1/4$, and the claim holds. □

Combining Claims 3.8.5 and 3.8.6 we get that

$$\begin{aligned} g \cdot 2^{\ell-k} &\le 2^\ell \cdot 2\exp\left(-\frac{\ell(1-4\epsilon)^2}{6}\right), \\ \Rightarrow \log g - k\log 2 &\le \log 2 - \frac{\ell(1-4\epsilon)^2}{6}, \\ \Rightarrow k &\ge \frac{\log g - \log 2 + \frac{\ell(1-4\epsilon)^2}{6}}{\log 2} \ge \frac{\log g + 3\ell/50}{\log 2} - 1, \end{aligned}$$

since $\epsilon \le 1/10$. □

Corollary 3.8.7. *For $\ell \ge 200$, there does not exist an $(\ell, k, 1/10, 1/2)$-SCS with $k < \ell/20$.*

3.8.2 Construction

We will derive a lower bound on k from the existence of a *Ruzsa-Szemerédi (RS) graph*, defined as follows.

Definition 3.8.8 (Ruzsa-Szemerédi graph)**.** *We call an (ordinary) graph a (t,a)-RS graph if its edge set is the union of t induced matchings of size a.*

Recall the definition of hypergraph cut sparsification schemes from Section 3.3.4:

Definition 3.3.1. *Let $\mathfrak{H}(n,r)$ be the set of hypergraphs on a vertex set $[n]$ with each hyperedge having size at most r. A pair of functions* $\textsc{Sparsify}: \mathfrak{H}(n,r) \to \{0,1\}^k$ *and* $\textsc{Cut}: \{0,1\}^k \times 2^{[n]} \to \mathbb{N}$ *is said to be an (n,r,k,ε)-HCSS if for all inputs $G = (V,E) \in \mathfrak{H}(n,r)$ the following holds.*

- *For every query $S \in 2^{[n]}$,* $\left|\textsc{Cut}(\textsc{Sparsify}(G), S) - |E(S,\overline{S})|\right| \le \varepsilon \cdot |E(S,\overline{S})|$.

Theorem 3.8.9. *Suppose there exists a (t,a)-RS graph on n vertices where $a \ge 6000\sqrt{n\log n}$ and $at \ge 480n$. Then, any $(2n, t+1, k, \varepsilon)$-HCSS where $\varepsilon \le a/(60n)$ must have*

$$k = \Omega(at).$$

This is equivalent to Theorem 3.1.1.

Proof. Let us fix such a (t,a)-RS graph G on n vertices, and a $(2n, t+1, k, \varepsilon)$-HCSS (SPARSIFY, CUT). We will use this HCSS as a black box to construct a string compression scheme using k bits of space, then bound k by Corollary 3.8.7. First let us convert G into a bipartite graph G'. Let the vertex set of $G' = (V', E')$ be $V \times \{0,1\}$ where $P = V \times \{0\}$ and $Q = V \times \{1\}$ are the two sides of the bipartition. For each edge $e = (u,v) \in E$, we add two edges to E': $((u,0),(v,1))$ and $((v,0),(u,1))$, ensuring that G' is indeed bipartite. Note that E' is the union of t induced matchings of size $2a$. Let us call these matchings $M_1, \dots M_t$ and let each M_j be supported on P_j in P and Q_j in Q. The maximum degree in G' is t.

We will use G' to design a compression of strings of length $\ell = 2ta$. Note that there are exactly $2ta$ edges of G'. Let ϕ be an arbitrary bijection from E' to $[\ell]$. For a string $s \in \{0,1\}^\ell$, Let E_s be the subset of E' defined as

$$E_s = \left\{ e \in E' : s_{\phi(e)} = 1 \right\}.$$

Thus the graph $G_s = (P \cup Q, E_s)$ encodes the string s. We then transform G_s into the hypergraph $H_s = (P \cup Q, E_s^H)$. Let E_s^H consist of one hyperedge corresponding to each vertex $u \in P$:

$$E_s^H = \{\{u\} \cup \Gamma_s(u) \mid u \in P\},$$

where Γ_s denotes the neighborhood in G_s.

Our compression function ENCODE is then simply to sparsify H_s using SPARSIFY. This can indeed be done, since H_s is a hypergraph with $2n$ vertices and each edge has cardinality at most $t+1$. It remains to define the decoding function DECODE.

Given a query $q \subseteq [\ell]$, we must estimate the size of $s \cap q$, the number of coordinates of s within q having value 1. To do this, we partition q into segments $q^1, \dots, q^t$, and then estimate the size of each $s \cap q^j$. Specifically, let

$$q^j = \{i \in q \mid \phi^{-1}(i) \in M_j\}.$$

We can then define

$$\text{DECODE}(\text{SPARSIFY}(H_s), q) = \sum_{j=1}^{t} \text{DECODE}^j(\text{SPARSIFY}(H_s), q^j).$$

Here DECODE^j remains undefined for now. In what follows we will define it such that

$$\text{DECODE}^j(\text{SPARSIFY}(H_s), q^j) \cong |s \cap q^j|.$$

To estimate the size of $s \cap q^j$, we will observe the cut $E_s^H(S, \overline{S}) = E_s^H(S_s^j, \overline{S}_s^j)$ defined as follows:

- From P, S contains the subset of vertices in P_j corresponding to edges in q^j. Formally

$$S \cap P = \{P \cap e \mid e \in M_j \text{ s.t. } \phi(e) \in q^j\}.$$

- From Q, S contains all vertices except Q_j.

We will prove the the size of the cut $(S,\overline{S})$ is closely related to the size of $s\cap q$, as long as s satisfies some nice properties.

Note that each hyperedge in E_s^H corresponds to a vertex in P: for $u\in P$ we denote the hyperedge $\{u\}\cup\Gamma_s(u)$ as e_u. We will bound the contribution of e_u to the cut $(S,\overline{S})$ for all u in each of the following three categories:

1. $u\in P_j\cap S$:
   Let the edge from M_j adjacent on u be f_u. For any such u, e_u crosses the cut if and only if $s_{\phi(f_u)}=1$. Indeed, if $s_{\phi(f_u)}=1$, then $f\in E_s$ and $f\cap Q\in Q_j\subseteq\overline{S}$. On the other hand, if $s_{\phi(f_u)}=0$ then $f\not\in E_s$ and all edges adjacent on u in E_s correspond to matchings different from M_j (that is M_k for $k\neq j$). Since M_j is induced by the property of RS-graphs, $\Gamma_s(u)\subseteq Q\setminus Q_j\subseteq S$. Therefore, the total amount of hyperedges crossing the cut from this category is exactly $|s\cap q^j|$.

2. $u\in P_j\setminus S$:
   In this case e_u crosses the cut unless $d_s(u)<2$. Indeed, if $d_s(u)\geq 2$ then at least one edge adjacent on u in G_s does *not* come from M_j. The other endpoint of this edge is in $Q\setminus Q_j\subseteq S$, whereas u itself is in $\overline{S}$ by definition. In the the case where $d_s(u)<2$ we cannot say whether e_u crosses the cut or not. Therefore, the number of hyperedges crossing the cut from this category is approximately $m-|q^j|$ (that is all of them), but with a possible error of
   $$|\{u\in P_j \mid d_s(u)<2\}|.$$

3. $u\in P\setminus P_j$:
   In this case we cannot say anything about the number of edges crossing the cut, except that it is unlikely to deviate from its expectation when s is considered to be uniformly random on $\{0,1\}^\ell$. Let
   $$Z_j=|\{u\in P\setminus P_j \mid \Gamma_s(u)\not\subseteq Q_j\}|,$$
   or the number of hyperedges in E_s^H from this category crossing the cut.

Overall, we can approximate the size of the cut $(S,\overline{S})$ in H_s by
$$|s\cap q^j|+(a-|q^j|)+\mathbb{E}_sZ_j, \tag{3.21}$$
with an maximum additive error of
$$\big|\{u\in P_j \mid d_s(u)<2\}\big|+\big|Z_j-\mathbb{E}_sZ_j\big|. \tag{3.22}$$

Conversely, this allows us to approximate $|s\cap q^j|$ using the size of the same cut in our $(2n,t+1,k,\varepsilon)$-HCSS. Therefore, we define $\textsc{Decode}^j$ as follows:
$$\textsc{Decode}^j(\textsc{Sparsify}(H_s),q^j)=\textsc{Cut}(\textsc{Sparsify}(H_s),S)-(a-|q^j|)-\mathbb{E}_sZ_j. \tag{3.23}$$

It remains to bound the total error introduced by the inaccuracies above.

We will define the set of good input strings, $\mathscr{G}$ to be those where this additive error is small across all j's, and we will prove that this contains a majority of possible input strings.

Claim 3.8.10. *Let $\mathscr{G}$ be the set of strings $s \in \{0,1\}^\ell$ such that*

$$\sum_{j=1}^{t} \left(\left| \{u \in P_j \mid d_s(u) < 2\} \right| + \left| Z_j - \mathbb{E}_s Z_j \right| \right) \leq 8n + 100t\sqrt{n \log n}.$$

Then $|\mathscr{G}| \geq 2^{\ell-1}$.

Proof. Consider s to be a random string, chosen uniformly on $\{0,1\}^\ell$. We will prove that $\mathbb{P}[s \in \mathscr{G}] \geq 1/2$. We do this by considering the two bad events

$$\sum_{j=1}^{t} \left| \{u \in P_j | d_s(u) < 2\} \right| > 8n,$$
$$\sum_{j=1}^{t} \left| Z_j - \mathbb{E} Z_j \right| > 100t\sqrt{n \log n},$$

and prove that neither happens with probability more than $1/4$.

To bound the probability of the first event consider the expectation of the sum:

$$\begin{aligned}\mathbb{E} \sum_{j=1}^{t} \left| \{u \in P_j \mid d_s(u) < 2\} \right| &= \mathbb{E} \sum_{j=1}^{t} \sum_{u \in P_j} \mathbb{1}(d_s(u) < 2) = \sum_{j=1}^{t} \sum_{u \in P_j} \mathbb{P}[d_s(u) < 2] \\ &= \sum_{u \in P} \sum_{j : u \in P_j} \mathbb{P}[d_s(u) < 2] = \sum_{u \in P} |\{j \mid u \in P_j\}| \cdot \mathbb{P}[d_s(u) < 2] \\ &= \sum_{u \in P} d(u) \cdot (d(u)+1) \cdot 2^{-d(u)} \leq 2n,\end{aligned}$$

as the function $d(d+1) \cdot 2^{-d}$ is bounded by 2 for all non-negative d.

This means, that by the Markov inequality

$$\mathbb{P}\left[\sum_{j=1}^{t} \left| \{u \in P_j \mid d_s(u) < 2\} \right| > 8n \right] \leq \frac{1}{4}.$$

Now, for the second bad event, we apply Chernoff bound (Theorem B.0.1). Note that

$$Z_j = |\{u \in P \setminus P_j \mid \Gamma_s(u) \not\subseteq Q_j\}|,$$

is the sum of $n - m$ independent random variables bounded by one. Therefore,

$$\mathbb{P}\left[|Z_j - \mathbb{E} Z_j| > \delta n \right] \leq 2 \exp\left(-\frac{\delta^2 n}{3} \right).$$

Setting δ to $100\sqrt{(\log n)/n}$ and taking union bound over $j = 1, \ldots, t$ gives us that

$$\mathbb{P}\left[\sum_{j=1}^{t} |Z_j - \mathbb{E}Z_j| > 100t\sqrt{n\log n}\right] \le \frac{1}{4}.$$

Putting the bounds on the first and second event together gives us the statement of the claim. □

This $\mathcal{G}$ will be our set of good inputs in our $(\ell, k, 1/10, 1/2)$-SCS. Claim 3.8.10 essentially shows that the error in our estimate of $|s \cap q|$ would be at most $8n + 100t\sqrt{n\log n}$ without the inaccuracy introduced by our cut sparsifier. Since the size of the cut $(S, \overline{S})$ is at most n (the total number of hyperedges in the hypergraph H_s), this introduces an additional εn additive error.

Formally, when $s \in \mathcal{G}$

$$\begin{aligned}
&\big||s \cap q - \textsc{Decode}(\textsc{Encode}(s), q)\big| \\
&= \left||s \cap q| - \sum_{j=1}^{t} \textsc{Decode}^j(\textsc{Sparsify}(H_s), q^j)\right| \\
&\le \sum_{j=1}^{t} \big||s \cap q^j| - \textsc{Decode}^j(\textsc{Sparsify}(H_s), q^j)\big| \\
&\le \sum_{j=1}^{t} \big||s \cap q^j| - \textsc{Cut}(\textsc{Sparsify}(H_s), S) + (a - |q^j|) + \mathbb{E}_s Z_j\big| && \text{by Eq. (3.23)} \\
&\le \sum_{j=1}^{t} \left(\big||s \cap q^j| - |E_s^H(S_s^j, \overline{S}_s^j)| + (a - |q^j|) + \mathbb{E}_s Z_j\big| + \big||E_s^H(S_s^j, \overline{S}_s^j)| - \textsc{Cut}(\textsc{Sparsify}(H_s), S_s^j)\big|\right) \\
&\le \sum_{j=1}^{t} \left(\big|\{u \in P_j \mid d_s(u) < 2\}\big| + \big|Z_j - \mathbb{E}_s Z_j\big| + \varepsilon n\right) && \text{by Equations Eq. (3.21) and Eq. (3.22)} \\
&\le \left(8n + 100t\sqrt{n\log n}\right) + \sum_{j=1}^{t} \varepsilon n && \text{since } s \in \mathcal{G} \\
&\le 8n + 100t\sqrt{n\log n} + \varepsilon t n
\end{aligned}$$

This is less than $\ell/20 = at/20$ due to the theorem's assumptions on the parameters. Therefore, $(\textsc{Encode}, \textsc{Decode})$ is a $(\ell, k, 1/10, 1/2)$-SCS with the set of good inputs being $\mathcal{G}$. By Corollary 3.8.7 k must be at least $\Omega(\ell) = \Omega(at)$. □

We can now apply Theorem 3.8.9 to several RS-graph constructions known in the literature. Note that if there exists a (t, a)-RS graph, one can always reduce the parameters to get an (t', a')-RS graph for $t' \le t$ and $a' \le a$. We begin with Fischer et al. Fischer et al. (2002), which proves the existence of $(n^{\Omega(1/\log\log n)}, n/3 - o(1))$-Ruzsa-Szemerédi graphs, resulting in the following corollary.

Corollary 3.8.11. *Any (n, r, k, ε)-HCSS with $r = n^{O(1/\log\log n)}$ and small constant ε requires $k = \Omega(nr)$ space.*

In other words, any data structure that can provide a $(1+\varepsilon)$-approximation to the size of all cuts in an

r-uniform hypergraph with n vertices and $r = n^{O(1/\log\log n)}$ for small constant $\epsilon \in (0,1)$ requires $\Omega(nr)$ bits of space. This is tight due to the hypergraph cut sparsifier construction of Chen et al. (2020). A different construction, also from Fischer et al. (2002), is able to achieve an $(n^c, n/O(\sqrt{\log\log n/\log n}))$-RS graph for some small enough constant c. This results in the following:

Corollary 3.8.12. *For some constant c, any (n,r,k,ϵ)-HCSS with $r = O(n^c)$ and $\epsilon = O(\sqrt{\log\log n/\log n})$ requires $k = \Omega(nr/\sqrt{\log n/\log\log n})$ space.*

Finally, the original construction of Ruzsa and Szemeredi Ruzsa and Szemerédi (1978) guarantees the existence of an $(n/3, n/2^{O(\sqrt{\log n})})$-RS graphs, implying:

Corollary 3.8.13. *Any (n,r,k,ϵ)-HCSS with $\epsilon = 2^{-\Omega(\sqrt{\log n})}$ requires $k = nr/2^{O(\sqrt{\log n})}$ space.*

These results imply that for any value of r, it is impossible to compress the cut structure of a hypergraph with n vertices and maximum hyperedge size r, with significantly less than nr space, and a polynomial scaling in the error (that is with $nr^{1-\Omega(1)}\epsilon^{-O(1)}$ space).

3.9 Spectral Sparsification of Directed Hypergraphs

In this section, we discuss spectral sparsification of directed hypergraphs. First we introduce some notions and study basic properties of directed hypergraphs in Section 3.9.1. Then, we discuss spectrally sparsifying directed hypergraphs with hyperedges having nearly equal overlap (a concept to be defined in Section 3.9.1). Finally, we prove Theorem 3.1.2 in Section 3.9.4.

3.9.1 Preliminaries

A *directed hypergraph* $G = (V,E)$ is a pair of a vertex set V and a set E of hyperarcs, where a *hyperarc* $e \in E$ is an ordered pair of two disjoint vertex sets $h(e) \subseteq V$, the *head*, and $t(e) \subseteq V$, the *tail*. The *size* of a hyperarc $e \in E$ is $|h(e)| + |t(e)|$. We restrict ourselves to dealing with only *simple* directed hypergraphs, that is, in Section 3.9 E is always considered to be a set as opposed to a multiset.

We say that a vertex set $S \subseteq V$ *cuts* a hyperarc $e \in E$ if $S \cap t(e) \neq \emptyset$ and $(V \setminus S) \cap h(e) \neq \emptyset$. The *energy* of a hyperarc e with respect to a vector $x \in \mathbb{R}^V$ is defined as

$$\max_{a \in t(e), b \in h(e)} (x_a - x_b)_+^2,$$

where $(\alpha)_+ = \max\{\alpha, 0\}$. The energies of a set of arcs, or of an entire vector with respect to G, is defined identically to the undirected case. So in particular the energy of x with respect to G is

$$Q(x) = \sum_{e \in E} \max_{a \in t(e), b \in h(b)} (x_a - x_b)_+^2.$$

Note that $Q(1_S)$, where $1_S \in \mathbb{R}^V$ is the characteristic vector of S, is equal to the number of hyperarcs cut by S. Identically to Definition 3.2.4, for $\epsilon > 0$, a weighted subgraph $\widetilde{G}$ of G is said to be a ϵ-spectral sparsifier of G if

$$\widetilde{Q}(x) = (1 \pm \epsilon) Q(x),$$

where $Q(x)$ and $\widetilde{Q}(x)$ are energy of x with respect to G and $\widetilde{G}$, respectively.

In constructing our sparsifier, a useful object to consider will be the *clique graph* of G, the directed (ordinary) multigraph we get by replacing each hyperarc in G with a directed bipartite clique. Formally, the clique of a hyperarc $e \in E$ is the set of arcs $C(e) = \{(a,b) \mid a \in t(e), b \in h(e)\}$. The clique graph of a set of hyperarcs $E' \subseteq E$ is the multi-union of the individual cliques $C(E') = \biguplus_{e \in E'} C(e)$. Finally, the clique graph of G itself is $C(G) = (V, C(E))$. In the following, we make some observation about the multiplicities of arcs in the clique graph.

Definition 3.9.1. *Given a hypergraph $G = (V,E)$, we say that a subset of hyperarcs $E' \subseteq E$ k-*overlapping *if every arc in $C(E')$ appears with multiplicity at least k. Furthermore, the* overlap $k(e)$ *of a single hyperarc $e \in E$ is defined as the largest k such that there exists a k-overlapping set of hyperarcs containing e.*

Informally, we will use the inverse overlap of each hyperarc as a sampling rate in constructing our sparsifier. Thus, the following lemma will be a useful bound on the sum of these rates:

Lemma 3.9.2. *Let $G = (V,E)$ be a directed hypergraph. Then, we have*

$$\sum_{e \in E} \frac{1}{k(e)} \leq n^2.$$

Proof. Consider the following simple algorithm:

Algorithm 10

```
1: procedure OverlapPeeling($G = (V,E)$)
2:     $E' \leftarrow E$.
3:     for $k = 1, \dots, 2^{n-2}$ do
4:         $E'_k \leftarrow E'$.
5:         while there exists $(u,v) \in C(E')$ with multiplicity at most $k$ do
6:             for all hyperarcs $e \in E'$ such that $(u,v) \in C(e)$ do
7:                 $f(e) \leftarrow (u,v)$.
8:                 $E' \leftarrow E' \setminus \{e\}$.
```

This algorithm iterates through all possible overlaps (from 1 to 2^{n-2}) and peels off all hyperarcs with this overlap, until no hyperarcs remain. The algorithm maintains several variables (E'_k and $f(e)$) that are not used. However, these will be useful in proving the lemma.

Claim 3.9.3. *The set E'_k has overlap k for all k.*

Proof. Indeed, the variable k is augmented in the for-loop at Line 3 only after exiting the while-loop at Line 5. This means that there no longer existed any pairs (u,v) in $C(E')$ with multiplicity at most $k-1$, and therefore E' was k-overlapping. (The exception to this argument is $k = 1$, however all sets are 1-overlapping by definition.) □

Claim 3.9.4. *If a hyperarc e is removed at a time when $k = k^*$, then it has overlap exactly k^*.*

Proof. It is easy to see that e has overlap *at least* k^*, since it was an element of E'_{k^*} which is itself k^*-overlapping by Section 3.9.1.

We prove that e has overlap *at most* k^* by induction. By induction, we can assume that all hyperarcs removed before e had overlap corresponding to the value of k at the time, that is, at most k^*. Let E^* be the current value of E' at the time just before e is removed. Suppose for contradiction that e is at least (k^*+1)-overlapping, or equivalently there exists a (k^*+1)-overlapping set containing e, say $\widetilde{E}_{k^*+1}$. However, no hyperarc removed before e could be in this set, since we know they are at most k^*-overlapping. So $\widetilde{E}_{k^*+1} \subseteq E^*$. But some arc in $C(e)$ has multiplicity only at most k^* in E^*, which is a contradiction. □

Claim 3.9.5. *For any pair $(u,v) \in V^2$, we have*

$$\sum_{e: f(e)=(u,v)} \frac{1}{k(e)} \leq 1.$$

Proof. First note that all pairs (u,v) are only considered once in the while-loop of Line 5 throughout the whole algorithm. Indeed, once a pair is considered, all hyperarcs containing it are removed and (u,v) is no longer in $C(E')$.

Suppose (u,v) is removed in this way when $k = k^*$. Then all hyperarcs e such that $f(e) = (u,v)$ have overlap at most k^*. On the other hand, there are at most k^* such hyperarcs due to the condition in Line 5. This concludes the proof of the claim. □

From here the lemma statement follows simply:

$$\sum_{e\in E} \frac{1}{k(e)} = \sum_{(u,v)\in V^2} \sum_{e: f(e)=(u,v)} \frac{1}{k(e)} \leq \sum_{(u,v)\in V^2} 1 = n^2. \qquad \square$$

Remark 3.9.6. *Note that, by Claim 3.9.4, we can compute overlaps of hyperarcs by running Algorithm 10. Furthermore, we can make it run in polynomial time by, instead of incrementing k at Line 3, updating k to be the smallest multiplicity of an edge in $C(E')$.*

3.9.2 Nearly Equally Overlapping Directed Hypergraphs

In this section, we consider the simpler case where every hyperarc has a similar overlap.

Lemma 3.9.7. *There is an algorithm that, given $0 < \epsilon \leq 1/2$ and a directed hypergraph $G = (V,E)$ such that every hyperarc has overlap between k and $2k$ for some $k \geq 1$, and each hyperarc has size at most $r \leq \sqrt{\epsilon n}/11$, outputs in polynomial time a weighted subgraph $\widetilde{G} = (V, \widetilde{E}, w)$ of G satisfying the following with probability $1 - O(1/n)$:*

- *$\widetilde{G}$ is an ϵ-spectral sparsifier of G,*
- *$|\widetilde{E}| = O(n^2 r^2 \log n/\epsilon^2)$.*

Construction Let us construct $\widetilde{G} = (V, \widetilde{E})$ by sampling each hyperarc independently with the same probability $p = 1000r^2 \log n/(k\epsilon^2)$ and scaling them up by $1/p$. Let the weight of each hyperarc e in $\widetilde{G}$ be denoted as w_e. Then w_e is an independent random variable taking value $1/p$ with probability p and value 0 otherwise, for each e.

Clearly, we can compute the output in $O(m)$ time. Also, we can bound the size of $\widetilde{E}$ easily:

Lemma 3.9.8. *We have* $\mathbb{E}[|\widetilde{E}|] = 2000n^2r^2 \log n/\epsilon^2$ *and*

$$\mathbb{P}\left[|\widetilde{E}| > 4000n^2r^2 \log n/\epsilon^2\right] \le 2\exp\left(-\frac{2pkn^2}{3}\right).$$

Proof. Note that since the overlap of each hyperarc is at most $2k$, there are at most $2kn^2$ hyperarcs in total (in E) by Lemma 3.9.2. Each hyperarc is sampled with probability p to be in $\widetilde{E}$, so $\mathbb{E}[|\widetilde{E}|] = 2pkn^2 = 2000n^2r^2 \log n/\epsilon^2$, as claimed. By Chernoff bounds (Theorem B.0.1), the claimed concentration inequality holds. □

Correctness We now examine the spectral properties of $\widetilde{G}$. Recall that $C(G)$ is the clique graph of G. Let us denote by Q^C the energy with respect to the clique graph. We may assume without loss of generality that $Q^C(x) = 1$, since whether $Q(x) = (1 \pm \epsilon)\widetilde{Q}(x)$ holds or not is unaffected by scaling x. Define $\overline{\mathbb{R}^V}$ to be the set of vectors x such that this is satisfied. Note that this means that $Q_x(E) \ge 1/r^2$. Indeed

$$\begin{aligned} Q_x(E) &= \sum_{e\in E} \max_{u\in t(e),\, v\in h(e)} (x_u - x_v)_+^2 = \sum_{e\in E} \max_{f\in C(e)} Q_x^C(f) \ge \frac{1}{r^2} \sum_{e\in E} \sum_{f\in C(E)} Q_x^C(f) \\ &= \frac{1}{r^2} \sum_{f\in C(E)} Q_x^C(f) = \frac{Q^C(x)}{r^2} = \frac{1}{r^2}. \end{aligned}$$

Let us categorize the arcs in $C(E)$ based on their contributions to the total energy $Q^C(x) = 1$ in $C(G)$. The categories are

$$C_i = \left\{ f \in C(E) \,\middle|\, Q_x^C(f) \in \left(\frac{2^{-i}}{k}, \frac{2^{-i+1}}{k}\right] \right\},$$

for $i = 1, \ldots, i^*$ where $i^* := \lceil 3\log n \rceil$, as well as

$$C_* = \left\{ f \in C(E) \,\middle|\, Q_x^C(f) \le \frac{2^{-i^*}}{k} \right\}.$$

Recall that $C(E)$ is a multiset, and consequently so are C_i and C_*. Since each arc f appears with multiplicity at least k, any single arc can contribute at most $1/k$ to the energy. Therefore, all arcs of $C(G)$ are covered by these categories.

We then partition the hyperarcs into similar categories: A hyperarc e gets into category i (or E_i) if i is the smallest number for which $C(e)$ contains an arc in C_i. Formally

$$\begin{aligned} E_i &= \left\{ e \in E \,\middle|\, i = \max\{j \mid C(e) \cap C_j \ne \emptyset\} \right\} \ (i = 1, \ldots, i^*), \text{ and} \\ E_* &= \{e \in E \mid C(e) \subseteq C_*\}. \end{aligned}$$

To prove that $\widetilde{G}$ is an ϵ-spectral sparsifier, we will show that, for all i, $Q_x(E_i) \approx \widetilde{Q}_x(E_i)$. Similarly to the proof in Section 3.4 we will introduce a discretization of $Q_x(E_i)$. However, unlike in the proof in Section 3.4, instead of rounding the vertex potentials x_v, we will round the energies of hyperarcs, that is, $Q_x(e)$ for $e \in E$.

Let us first define $Q_x^{C,(i)}(f)$, the rounding of $Q_x^C(f)$. Firstly, if $Q_x^C(f) \le 2^{-i}/k$, that is the arc f is not relevant to E_i, we define $Q_x^{C,(i)}(f)$ to be zero. Otherwise, let $Q_x^{C,(i)}(f)$ be the rounding of $Q_x^C(f)$ to the nearest integer multiple of $1/(kn^3)$. Analogously with the definition of Q_x, for $e \in E$ let

$$Q_x^{(i)}(e) = \max_{f \in C(e)} Q_x^{C,(i)}(f), \quad Q_x^{(i)}(E') = \sum_{e \in E'} Q_x^{(i)}(E'),$$
$$\widetilde{Q}_x^{(i)}(e) = w_e Q_x^{(i)}(e), \quad \widetilde{Q}_x^{(i)}(E') = \sum_{e \in E'} \widetilde{Q}_x^{(i)}(e).$$

Informally, we prove the following chain of approximations for each i:

$$Q_x(E_i) \cong Q_x^{(i)}(E_i) \cong \widetilde{Q}_x^{(i)}(E_i) \cong \widetilde{Q}_x(E_i),$$

as well as

$$Q_x(E_*) \cong \widetilde{Q}_x(E_*).$$

We make this formal in the following claims:

Claim 3.9.9. *For all $x \in \mathbb{R}^V$ and all $i = 1, \ldots, i^*$,*

$$Q_x^{(i)}(E_i) = Q_x(E_i) \pm \frac{2}{n}.$$

Claim 3.9.10. *For all $i = 1, \ldots, i^*$,*

$$\mathbb{P}\left[\forall x \in \overline{\mathbb{R}^V} : \widetilde{Q}_x^{(i)}(E_i) = \left(1 \pm \frac{\epsilon}{2}\right) Q_x^{(i)}(E_i) \pm \frac{\epsilon Q(x)}{10 \log n}\right] \ge 1 - \frac{1}{n}.$$

Claim 3.9.11. *For all $i = 1, \ldots, i^*$,*

$$\mathbb{P}\left[\forall x \in \overline{\mathbb{R}^V} : \widetilde{Q}_x^{(i)}(E_i) = \widetilde{Q}_x(E_i) \pm \frac{4}{n}\right] \ge 1 - \frac{1}{n}.$$

Claim 3.9.12.

$$\mathbb{P}\left[\forall x \in \overline{\mathbb{R}^V} : \widetilde{Q}_x(E_*) = Q_x(E_*) \pm \frac{6}{n}\right] \ge 1 - \frac{1}{n}.$$

Before proving the above claims, which we do in the next section, we conclude the analysis of correctness of the sparsifier.

Lemma 3.9.13. *The directed hypergraph $\widetilde{G}$ is an ϵ-spectral sparsifier of G with probability $1 - O(1/n)$.*

Proof. The statements of Claims 3.9.10 to 3.9.12 all hold with high probability. Let us consider the event that they all hold simultaneously, then by Claims 3.9.9 to 3.9.11,

$$\begin{aligned}\left|Q_x(E_i)-\widetilde{Q}_x(E_i)\right| &\le \left|Q_x(E_i)-Q_x^{(i)}(E_i)\right|+\left|Q_x^{(i)}(E_i)-\widetilde{Q}_x^{(i)}(E_i)\right|+\left|\widetilde{Q}_x^{(i)}(E_i)-\widetilde{Q}_x(E_i)\right| \\ &\le \frac{2}{n}+\frac{\epsilon}{2}Q_x^{(i)}(E_i)+\frac{Q(x)}{10\log n}+\frac{4}{n}\le \frac{\epsilon}{2}Q_x(E_i)+\frac{Q(x)}{10\log n}+\frac{6}{n},\end{aligned}$$

using that $\epsilon \le 1$. Summing this over $i = 1\dots, i^* = \lceil 3\log n\rceil$ and adding Claim 3.9.12 we get

$$\begin{aligned}\left|Q(x)-\widetilde{Q}(x)\right| &\le \sum_{i=1}^{i^*}\left|Q_x(E_i)-\widetilde{Q}_x(E_i)\right|+\left|Q_x(E_*)-\widetilde{Q}_x(E_*)\right| \\ &\le \sum_{i=1}^{i^*}\left[\frac{\epsilon}{2}Q_x(E_i)+\frac{\epsilon Q(x)}{10\log n}+\frac{6}{n}\right]+\frac{6}{n}\le \frac{\epsilon}{2}Q(x)+\frac{4\epsilon}{10}Q(x)+\frac{12}{n}\le \epsilon Q(x),\end{aligned}$$

since $\epsilon Q(x)/10 \ge \epsilon/(10r^2) \ge 12/n$ because $11r \le \sqrt{\epsilon n}$. □

Lemma 3.9.7 follows by Lemmas 3.9.8 and 3.9.13 and a union bound.

3.9.3 Proofs of Claims 3.9.9 to 3.9.12

We begin with a preliminary claim examining the difference between Q_x and $Q_x^{(i)}$ on a single hyperarc.

Claim 3.9.14. *For all $x \in \mathbb{R}^V$, all $i = 1,\dots,i^*$, and any hyperarc $e \in E_i$,*

$$Q_x^{(i)}(e) = Q_x(e) \pm \frac{1}{kn^3}.$$

Proof. Suppose first that $Q_x(e) \ge Q_x^{(i)}(e)$. Recall that $e \in E_i$ and by definition of E_i there exist arcs in $C(e)\cap C_i$. In this case let $f = \operatorname{argmax}_{f\in C(e)} Q_x^C(f)$, guaranteeing that $f \in C_i$. Therefore, by definition $Q_x^{C,(i)}(f)$ is not zero, but a rounding to the nearest integer multiple of $1/(kn^3)$. Therefore,

$$Q_x(e) - Q_x^{(i)}(e) \le Q_x^C(f) - Q_x^{C,(i)}(f) \le \frac{1}{kn^3}.$$

Now suppose that $Q_x(e) < Q_x^{(i)}(e)$. In this case we define f to be $\operatorname{argmax}_{f\in C(e)} Q_x^{C,(i)}(e)$. Now if $Q_x^{C,(i)}(f)$ is zero the claim holds trivially, so we may assume that $Q_x^{C,(i)}(f)$ is instead a rounding to the nearest integer multiple of $1/(kn^3)$:

$$Q_x^{(i)}(e) - Q_x(e) \le Q_x^{C,(i)}(f) - Q_x^C(f) \le \frac{1}{kn^3}.$$ □

Claim 3.9.9. *For all $x \in \mathbb{R}^V$ and all $i = 1,\dots,i^*$,*

$$Q_x^{(i)}(E_i) = Q_x(E_i) \pm \frac{2}{n}.$$

Proof. We can simply sum over the hyperarcs in E_i. Since $|E_i| \le |E| \le 2kn^2$, we have that

$$\left|Q_x(E_i) - Q_x^{(i)}(E_i)\right| \le \sum_{e \in E_i} \left|Q_x(e) - Q_x^{(i)}(e)\right| \le \frac{|E_i|}{kn^3} \le \frac{2}{n}. \qquad \square$$

Claim 3.9.10. *For all $i = 1, \dots, i^*$,*

$$\mathbb{P}\left[\forall x \in \overline{\mathbb{R}^V} : \widetilde{Q}_x^{(i)}(E_i) = \left(1 \pm \frac{\epsilon}{2}\right) Q_x^{(i)}(E_i) \pm \frac{\epsilon Q(x)}{10 \log n}\right] \ge 1 - \frac{1}{n}.$$

Proof. We prove the stronger claim

$$\mathbb{P}\left[\forall x \in \overline{\mathbb{R}^V} : \widetilde{Q}_x^{(i)}(E_i) = \left(1 \pm \frac{\epsilon}{2}\right) Q_x^{(i)}(E_i) \pm \frac{\epsilon}{10 r^2 \log n}\right] \ge 1 - \frac{1}{n},$$

replacing $Q(x)$ by $1/r^2$ in the allowable additive error, which depends on x only through $Q_x^{(i)}$ and E_i.

We first consider a single setting of x (and consequently E_i and $Q_x^{(i)}$). Since $\mathbb{E}[\widetilde{Q}_x^{(i)}(e)] = Q_x^{(i)}(e)$, we can apply additive-multiplicative Chernoff (Theorem B.0.2) to get the desired bound. Each independent random variable ($\widetilde{Q}_x^{(i)}(e)$ for $e \in E_i$) is in the range $[0, 2^{-i+1}/(pk)]$ by definition of E_i. Therefore we get

$$\begin{aligned}\mathbb{P}\left[\left|\widetilde{Q}_x^{(i)}(E_i) - Q_x^{(i)}(E_i)\right| > \frac{\epsilon}{2} Q_x^{(i)}(E_i) + \frac{\epsilon}{10 r^2 \log n}\right] &\le 2 \exp\left(-\frac{\epsilon/2 \cdot \frac{\epsilon}{10 r^2 \log n}}{3 \cdot 2^{-i+1}/(pk)}\right) \\ &= 2 \exp\left(-\frac{\epsilon^2 pk \cdot 2^i}{120 r^2 \log n}\right).\end{aligned}$$

We will now use a union bound to prove that this holds simultaneously for all possible settings of E_i and $Q_x^{(i)}$. Recall that by definition $\bigcup_{j=1}^{i} C_j$ contains exactly arcs of $C(E)$ that contribute more than $2^{-i}/k$ energy to the total energy of $Q^C(x) = 1$. There are at most $k \cdot 2^i$ such arcs, but since each arc appears with multiplicity at least k, there are at most 2^i *distinct* arcs. The $Q_x^{C,(i)}$-value of all arcs not in $\bigcup_{j=1}^{i} C_j$ is zero. To select this multiset of non-zero valued arcs there are

$$\binom{n^2}{2^i} \le n^{2 \cdot 2^i} = \exp\left(2 \cdot 2^i \log n\right)$$

options. Furthermore, for each relevant arc, we must choose its $Q_x^{C,(i)}$-value: This is an integer multiple of $1/kn^3$ in the range $[-1/k, 1/k]$ and so there are $2n^3$ options per arc—for a total of

$$\left(2n^3\right)^{2^i} \le \exp\left(4 \cdot 2^i \log n\right)$$

options. Finally, for each relevant arc, we must choose which category among $E_1, \dots, E_i$ it belongs to (as this may not be evident from the value of $Q_x^{C,(i)}$). This is an additional $i \le 3 \log n + 1$ options per arc—for a total of

$$\left(3 \log n + 1\right)^{2^i} \le \exp\left(2^i \log n\right),$$

options among all arcs.

Ultimately, there are

$$\exp\left(2\cdot 2^i\log n+4\cdot 2^i\log n+2^i\log n\right)=\exp\left(7\cdot 2^i\log n\right)$$

possible settings of $(E_1,\dots,E_i,Q_x^{C,(i)})$.

Combining the above Chernoff bound for a single setting of x with this union bound we get the statement of the claim:

$$\begin{aligned}&\mathbb{P}\left[\forall x:\ \left|Q_x^{(i)}(E_i)-\widetilde{Q}_x^{(i)}(E_i)\right|>\frac{\epsilon}{2}Q_x^{(i)}(E_i)+\frac{\epsilon}{10r^2\log n}\right]\\ \le&2\exp\left(7\cdot 2^i\log n\right)\cdot\exp\left(-\frac{\epsilon^2pk\cdot 2^{-i}}{120r^2}\right)\\ =&2\exp\left(2^i\cdot\left(7\log n-\frac{\epsilon^2pk}{120r^2}\right)\right)\\ \le&\frac{1}{n},\end{aligned}$$

since $pk=1000r^2\log n/\epsilon^2$. □

Claim 3.9.11. *For all $i=1,\dots,i^*$,*

$$\mathbb{P}\left[\forall x\in\overline{\mathbb{R}^V}:\ \widetilde{Q}_x^{(i)}(E_i)=\widetilde{Q}_x(E_i)\pm\frac{4}{n}\right]\ge 1-\frac{1}{n}.$$

Proof. We consider the high probability event that $|\widetilde{E}|\le 4pkn^2$. (Lemma 3.9.8). Similarly to the proof of Claim 3.9.9 we simply sum over all edges of E_i. Note that if $e\in\widetilde{E}$,

$$\left|\widetilde{Q}_x(e)-\widetilde{Q}_x^{(i)}(e)\right|\le\sum_{e\in E_i}\left|\widetilde{Q}_x(e)-\widetilde{Q}_x^{(i)}(e)\right|=\sum_{e\in E_i\cap\widetilde{E}}\frac{1}{p}\left|Q_x(e)-Q_x^{(i)}(e)\right|\le\frac{|\widetilde{E}|}{pkn^3}\le\frac{4}{n}.$$ □

Claim 3.9.12.

$$\mathbb{P}\left[\forall x\in\overline{\mathbb{R}^V}:\ \widetilde{Q}_x(E_*)=Q_x(E_*)\pm\frac{6}{n}\right]\ge 1-\frac{1}{n}.$$

Proof. Note that

$$\left|Q_x(E_*)-\widetilde{Q}_x(E_*)\right|\le Q_x(E_*)+\widetilde{Q}_x(E_*).$$

We bound the two terms separately:

$$Q_x(E_*)\le|E_*|\cdot\frac{1}{kn^3}\le|E|\cdot\frac{1}{kn^3}\le\frac{2}{n},$$

and

$$\widetilde{Q}_x(E_*)\le|\widetilde{E}|\cdot\frac{1}{pkn^3}\le\frac{4}{n},$$

with high probability by Lemma 3.9.8. □

3.9.4 Proof of Theorem 3.1.2

Proof of Theorem 3.1.2. Given the results of Lemma 3.9.7, we only need to decompose G into directed hypergraphs with their hyperedges having nearly the same overlap. We will repeatedly separate and sparsify hyperarcs of the highest overlap until no hyperarcs remain. This results in an ϵ-spectral sparsifier of G, since the quality of being an ϵ-spectral sparsifier is additive.

Consider the following simple algorithm, where UNIFORMSAMPLINGSPARSIFY denotes the sparsification algorithm given in Lemma 3.9.7:

Algorithm 11 Directed hypergraph sparsification

```
1: procedure SPARSIFY($G = (V, E)$)
2:     $\widetilde{E} \leftarrow \emptyset$.
3:     while $E \neq \emptyset$ do
4:         $2k \leftarrow$ the largest overlap among hyperedges in $E$.
5:         $E' \leftarrow$ the maximal $k$-overlapping set in $E$.
6:         $E \leftarrow E \setminus E'$.
7:         $\widetilde{E} \leftarrow \widetilde{E} \cup$ UNIFORMSAMPLINGSPARSIFY($V, E'$).
8:     return $(V, \widetilde{E})$.
```

Note first that the maximal set of a certain multiplicity (in Line 5) is indeed unique. It follows from definition that the union of hyperarc sets of multiplicity k still has multiplicity k. Therefore, E' contains all hyperarcs of overlap at least k form (the current) E. Furthermore, removing E' from E reduces the maximum overlap of any hyperarc to below k, so by a factor of at least 2. Since the maximum overlap started out is at most n^{r-2}, Section 3.9.4 terminates in at most $r \log n$ iterations. Since the size of $\widetilde{E}$ increased by at most $O(n^2 r^2 \log n/\epsilon^2)$ in each iteration, by Lemma 3.9.7, this results in $|\widetilde{E}| = O(n^2 r^3 \log^2 n/\epsilon^2)$, as claimed.

The running time is polynomial because we can compute overlaps of hyperarcs in polynomial time by Remark 3.9.6, and hence can compute the largest overlap k and the maximal k-overlapping set in polynomial time. □

4 Massively Parallel Simulation of Random Walks

This chapter is based on joint work with Michael Kapralov, Silvio Lattanzi, and Navid Nouri. It has been accepted to the 34th Conference on Neural Information Processing Systems (**NeurIPS**) 2021 Kapralov et al. (2021c) under the title

Efficient and Local Parallel Random Walks.

4.1 Introduction

Random walks are key components of many machine learning algorithms with applications in computing graph partitioning Spielman and Teng (2004); Gluch et al. (2021), spectral embeddings Czumaj et al. (2015); Chiplunkar et al. (2018), or network inference Hoskins et al. (2018), as well as learning image segmentation Meila and Shi (2000), ranking nodes in a graph Agarwal and Chakrabarti (2007) and many other applications. With the increasing availability and importance of large scale datasets it is important to design efficient algorithms to compute random walks in large networks.

Several algorithms for computing random walks in parallel and streaming models have been proposed in the literature. In the streaming setting, Sarma, Gollapudi and Panigrahy Das Sarma et al. (2011) introduced multi-pass streaming algorithms for simulating random walks, and recently Jin Jin (2019) gave algorithms for generating a single random walk from a prespecified vertex in one pass. The first efficient parallel algorithms for this problem have been introduced in the PRAM model Karger et al. (1992); Halperin and Zwick (1996).

In a more recent line of work, Bahmani, Chakrabarti, and Xin Bahmani et al. (2011) designed a parallel algorithm that constructs a single random walk of length ℓ from every node in $O(\log \ell)$ rounds in the massively parallel computation model (MPC), with the important caveat that these walks are not independent (an important property in many applications). This was followed by the work of Assadi, Sun and Weinstein Assadi et al. (2019b), which gave an MPC algorithm for generating random walks in an undirected regular graph. Finally, Łącki et al. Łącki et al. (2020) presented a new algorithm to compute random walks of length ℓ from every node in an arbitrary undirected graph. The algorithm of Łącki et al. (2020) still uses only $O(\log \ell)$ parallel rounds, and walks computed are now independent.

From a high level perspective, the main idea behind all the MPC algorithms presented in Bahmani et al. (2011); Assadi et al. (2019b); Łącki et al. (2020) is to compute random walks of length ℓ by stitching together random walks of length $\ell/2$ in a single parallel round. The walks of length $\ell/2$ are computed by stitching together random walks of length $\ell/4$ and so on. It is possible to prove that such strategy leads to algorithms that run in $O(\log \ell)$ parallel rounds as shown in previous work (this is also optimal under the 1-vs-2 cycle conjecture, as shown in Łącki et al. (2020)). Note that this technique in order to succeed computes in round i several random walks of length 2^i for all the nodes in the network in parallel. This technique is very effective if we are interested in computing random walks from all the nodes in the graph, or, more precisely, when the number of walks computed out of a node is proportional to its stationary distribution. However, this approach leads to significant inefficiencies when we are interested in computing random walks only out of a subset of nodes or for a single node in the graph. This is even more important when we consider that in many applications as in clustering Gargi et al. (2011); Gleich and Seshadhri (2012); Whang et al. (2013) we are interested in running random walks only from a small subset of seed nodes. This leads to the natural question: *Is it possible to compute efficiently and in parallel random walks only from a subset of nodes in a graph?*

In this chapter we answer this question in the affirmative, and we show an application of such a result in local clustering. Before describing our results in detail, we discuss the precise model of parallelism that we use in this work.

The MPC model. We design algorithms in the massively parallel computation (MPC) model, which is a theoretical abstraction of real-world system, such as MapReduce Dean and Ghemawat (2008), Hadoop White (2012), Spark Zaharia et al. (2010) and Dryad Isard et al. (2007). The MPC model Karloff et al. (2010); Goodrich et al. (2011); Beame et al. (2013) is the de-facto standard for analyzing algorithms for large-scale parallel computing.

Computation in MPC is divided into synchronous *rounds* over multiple machines. Each machine has memory S and at the beginning data is partitioned arbitrarily across machines. During each round, machines process data locally and then exchange data with the restriction that no machine receives more than S bits of data. The efficiency of an algorithm in this model is measured by the number of rounds it takes for the algorithm to terminate, by the size of the memory of every machine and by the total memory used in the computation. In this chapter we focus on designing algorithm in the most restrictive and realistic regime where $S \in O(n^\delta)$ for a small constant $\delta \in (0,1)$ – these algorithms are called fully scalable.

Our contributions. Our first contribution is an efficient algorithm for computing multiple random walks from a single node in a graph efficiently.

Theorem 4.1.1. *There exists a fully scalable MPC algorithm that, given a graph $G = (V, E)$ with n vertices and m edges, a root vertex r, and parameters B^*, ℓ and λ, can simulate B^* independent random walks on G from r of length ℓ with an arbitrarily low error, in $O(\log \ell \log_\lambda B^*)$ rounds and $\widetilde{O}(m\lambda\ell^4 + B^*\lambda\ell)$ total space.*

Our algorithm also applies to the more general problem of generating independent random walks from a subset of nodes in the graph:

Theorem 4.1.2. *There exists a fully scalable MPC algorithm that, given a graph $G = (V, E)$ with n vertices* *and m edges and a collection of non-negative integer budgets $(b_u)_{u\in V}$ for vertices in G such that $\sum_{u\in V} b_u = B^*$, parameters ℓ and λ, can simulate, for every $u \in V$, b_u independent random walks on G of length ℓ from u with an arbitrarily low error, in $O(\log\ell \log_\lambda B^*)$ rounds and $\widetilde{O}(m\lambda\ell^4 + B^*\lambda\ell)$ total space. The generated walks are independent across starting vertices $u \in V$.*

The following remark clarifies the effect of parameter λ on the number of machines.

Remark 4.1.3. *The parameter λ has nothing to do with the input of the algorithm, but is a trade-off parameter between space and round complexity. It is useful to think of it as $\lambda = n^\epsilon$ for some ϵ (not necessarily a constant), in which case we get a round complexity of $O(\log\ell\log B^*/(\epsilon\log n)) \le O(\log\ell/\epsilon)$ and a total memory of $\widetilde{O}(mn^\epsilon\ell^4 + B^* n^\epsilon \ell)$. We can set ϵ to, for example, $1/\log\log n$, to get nearly optimal total space and $\widetilde{O}(\log\ell)$ round complexity.*

If we compare our results with previous works, our algorithm computes truly independent random walks as Łącki et al. (2020) does. This is in contrast with the algorithm of Bahmani et al. (2011), which introduces dependent constructs not independent walks. Our algorithm has significantly better total memory than Łącki et al. (2020), which would result in memory $\Omega(m\cdot B^*)$ for generating B^* walks out of a root node r. This comes at the cost of a slightly higher number of rounds ($\log_\lambda B^*$, a factor that in many applications can be considered constant).

The main idea is to preform multiple cycles of stitching algorithms, changing the initial distribution of the random walks adaptively. More precisely, in an initial cycle we construct only a few random walks, distributed according to the stationary distribution – this is known to be doable from previous work. Then, in each cycle we increase the number of walks that we build for node r by a factor of λ and we construct the walks only by activating other nodes in the graph that contribute actively in the construction of the random walks for r. In this way we obtain an algorithm that is significantly more work efficient in terms of total memory, compared with previous work.

Our second contribution is to present an application of our algorithm to estimating Personalized PageRank and to local graph clustering. To the best of our knowledge, our algorithm is the first local clustering algorithm that uses a number of parallel rounds that only have a logarithmic dependency on the length of the random walk used by the local clustering algorithm.

Theorem 4.1.4. *For any $\lambda > 1$, let $\alpha \in (0,1]$ be a constant and let C be a set satisfying that the conductance of C, $\Phi(C)$, is at most $\alpha/10$ and $\mathrm{Vol}(C) \le \frac{2}{3}\mathrm{Vol}(G)$. Then there is an MPC algorithm for local clustering that uses $O(\log\ell\cdot\log_\lambda B^*) = O(\log\log n\cdot\log_\lambda(\mathrm{Vol}(C)))$ rounds of communication and total memory $\widetilde{O}(m\lambda\ell^4 + B^*\lambda\ell) = \widetilde{O}(m\lambda + \lambda\mathrm{Vol}(C)^2)$, where $B^* := \frac{10^6\log^3 n}{\eta^2\alpha^2}$, $\ell := \frac{10\log n}{\alpha}$ and $\eta = \frac{1}{10\mathrm{Vol}(C)}$, and outputs a cluster with conductance $O(\sqrt{\alpha\log(\mathrm{Vol}(C))})$.*

Finally we present an experimental analysis of our results where we show that our algorithm to compute random walk is significantly more efficient than previous work Łącki et al. (2020), and that our technique scale to very large graphs.

Additional related works. Efficient parallel random walks algorithm have also been presented in distributed computing Das Sarma et al. (2013) and using multicores Shun et al. (2016). Although the algorithms in Das Sarma et al. (2013) require a number of rounds linear in the diameter of the graph. The results in Shun et al. (2016) are closer in spirit to our work here but from an algorithmic perspective the challenges in developing algorithms in multicore and MPC settings are quite different. In our setting, most of the difficulty is in the fact that there is no shared memory and coordination between machines. As a result, bounding communication between machines and number of rounds is the main focus of this line of research. From an experimental perspective an advantage of the MPC environment is that it can scale to larger instances, in contrast an advantage of the multicore approach is that it is usually faster in practice for medium size instances. In our case study, where one is interested in computing several local clusters from multiple nodes(for example to detect sybil attack(look at Alvisi et al. (2014) and following work for applications) the MPC approach is often more suitable. This is due to the fact that the computation of multiple Personalized PageRank vectors at the same time often requires a large amount of space.

Several parallel algorithm have also been presented for estimating PageRank or PersonalizedPageRank Das Sarma et al. (2011, 2015); Bahmani et al. (2011), but those algorithms either have higher round complexity or introduce dependencies between the PersonalizedPageRank vectors computed for different nodes. Finally there has been also some work in parallelize local clustering algorithm Chung and Simpson (2015), although all previously known methods have complexity linear in the number of steps executed by the random walk/process used in the algorithm(in fact, our method could potentially be used to speed-up this work as well).

Notation. We work on undirected unweighted graphs, which we usually denote by $G = (V, E)$, where V and E are the set of vertices and the set of edges, respectively. We also have $n := |V|$ and $m := |E|$, unless otherwise stated. We define matrix D as the diagonal matrix of degrees, i.e., $D_{i,i} = d(v_i)$. Also, we let A be the adjacency matrix, where $A_{i,j} = 1$ if and only if there is an edge joining v_i and v_j, and $A_{i,j} = 0$, otherwise.

4.2 MPC random walks

In this section, we present our main result to compute B^* random walks from a single root vertex r up to a length of ℓ, then we generalize it to multiple sources. As mentioned before, our main idea is to carefully stitch random walks adaptively to activate nodes locally. In the rest of the section, we start by presenting our main theorem and giving a brief overview of our algorithm. We then describe our stitching algorithm, and analyze the number of random walks we must start from each node so that our algorithms work. Finally, we present the extension of our result to the setting with multiple sources.

Theorem 4.1.1. *There exists a fully scalable MPC algorithm that, given a graph $G = (V, E)$ with n vertices and m edges, a root vertex r, and parameters B^*, ℓ and λ, can simulate B^* independent random walks on G from r of length ℓ with an arbitrarily low error, in $O(\log \ell \log_\lambda B^*)$ rounds and $\widetilde{O}(m\lambda\ell^4 + B^*\lambda\ell)$ total space.*

4.2.1 Overview of our Algorithm

Here, we explain the frameworks of stitching and budgeting, which are the two key tools making up our algorithm. For simplicity and without loss of generality, we assume that each vertex v has its own machine that stores its neighborhood, its budgets, and its corresponding random walks.[1]

Remark 4.2.1. *For ease of notation we assume that $\ell = 2^j$ for some integer j. One can see that this assumption is without loss of generality, because otherwise one can always round ℓ to the smallest power of 2 greater than ℓ, and solve the problem using the rounded ℓ. This affects the complexity bounds by at most a constant factor.*

Stitching. Here, we explain the framework of stitching, which is a key tool for our algorithm. At each point in time the machine corresponding to v stores sets of random walks of certain lengths, each starting in v. Each edge of each walk is labeled by a number from 1 to ℓ, denoting the position it will hold in the completed walk. Thus, a walk of length s could be labeled $(k+1,\dots,k+s)$ for some k. Initially each vertex generates a pre-determined number of random edges (or walks of length one) with each label from 1 to ℓ. Thus at this point, we would find walks labeled 1, 2, 3, ... on each machine. After the first round of stitching, these will be paired up into walks of length two, and so we will see walks labeled by $(1,2)$, $(3,4)$, $(5,6)$, ... on each machine. After the second round of stitching we will see walks of length 4, such as $(1,2,3,4)$, and so on. Finally, after the last round of stitching, each machine will contain some number of walks of length ℓ (labeled from 1 to ℓ), as desired.

At any given time let $W_k(v)$ denote the set of walks stored by v whose first label is k and $B(v,k)$ denotes their cardinality – in the future, we will refer to the function B as the budget. After the initial round of edge generation, $W_k(v)$ consists of $B(v,k)$ individual edges adjacent to v, for each v and k.

The rounds of communication proceed as follows: in the first round of stitching, for each edge (or length one walk) e in $W_k(v)$, for any *odd* k, v sends a request for the continuation of the walk to z, where z is the other endpoint of e. That is, v sends a request to z for an element of $W_{k+1}(z)$. Each vertex sends out all such requests simultaneously in a single MPC round. Following this each vertex replies to each request by sending a walk from the appropriate set. Crucially, each request must be answered with a *different, independent* walk. If the number of requests for $W_{k+1}(z)$ exceeds $|W_{k+1}(z)| = B(z,k+1)$, the vertex z declares failure and the algorithm terminates. Otherwise, all such requests are satisfied simultaneously in a single MPC round. Finally, each vertex v increases the length of each of its walks in $W_k(v)$ to two when k is odd, and deletes all remaining walks in $W_k(v)$ when k is even (see Fig. 4.1). For a more formal definition see Algorithm 12.

Budgeting. A crucial aspect of stitching is that the budgets $B(v,k)$ need to be carefully prescribed. If at any point in time a vertex receives more requests than it can serve, the entire algorithm fails. In the case where $B(\cdot,1)$ follows the stationary distribution, this can be done (see for instance Łącki et al. (2020)), since the number of requests – at least in expectation – will also follow the stationary distribution. In our setting however, when $B(\cdot,1)$ follows the indicator distribution of r, this is much more difficult. We should assign higher budgets to vertices nearer r; however, we have no knowledge of which vertices these are.

In other words, the main challenge in making stitching work with low space and few rounds is to set the vertex budgets $(B(v,k))$ accurately enough for stitching to succeed – this is the main technical contribution of this chapter.

[1]In reality multiple vertices may share a machine, if they have low degree; or if a vertex has a high degree it may be accommodated by multiple machines in a constant-depth tree structure.

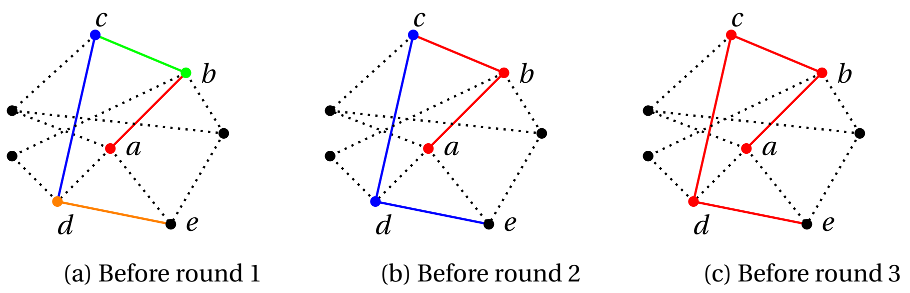


Figure 4.1 – Illustration of stitching algorithm for walk (a, b, c, d, e). The red, green, blue and orange walks in each figure correspond to walks in $W_1(a)$, $W_2(b)$, $W_3(c)$ and $W_4(d)$, respectively.

Our technique is to run multiple cycles of stitching sequentially. In the first cycle, we simply start from the stationary distribution. Then, with each cycle, we shift closer and closer to the desired distribution, in which the budget of r is much greater than the budgets of other vertices. We do this by augmenting $B(r, 1)$ by some parameter λ each cycle. This forces us to augment other budgets as well: For example, for u in the neighborhood of r we expect to have a significantly increased budget $B(u, 2)$. In order to estimate the demand on u (and all other vertices) we use data from the previous cycle.

Though initially only a few walks simulated by our algorithm start in r (we call these rooted walks), we are still able to derive some information from them. For instance, we can count the number of walks starting in r and reaching u as their second step. If κ rooted walks visited u as their second step in the previous cycle, we expect this number to increase to $\lambda \cdot \kappa$ in the following cycle. Hence, we can preemptively increase $B(u, 2)$ to approximately $\lambda \cdot \kappa$.

More precisely, we set the initial budget of each vertex to $\sim B_0 \cdot \deg(v)$ – an appropriately scaled version of the stationary distribution. This guarantees that the first round of stitching succeeds. Afterwards we set each budget $B(v, k)$ individually based on the information gathered from the previous cycle. We first count the number of rooted walks that ended up at v as their k^{th} step (Line 9). If this number is sufficiently large to be statistically significant (above some carefully chosen threshold θ in our case, Line 10), then we estimate the budget $B(v, k)$ to be approximately $\lambda \cdot \kappa$ in the following cycle (Line 11). On the other hand, if κ is deemed too small, it means that rooted random walks rarely reach v as their k^{th} step, and the budget $B(v, k)$ remains what it was before.

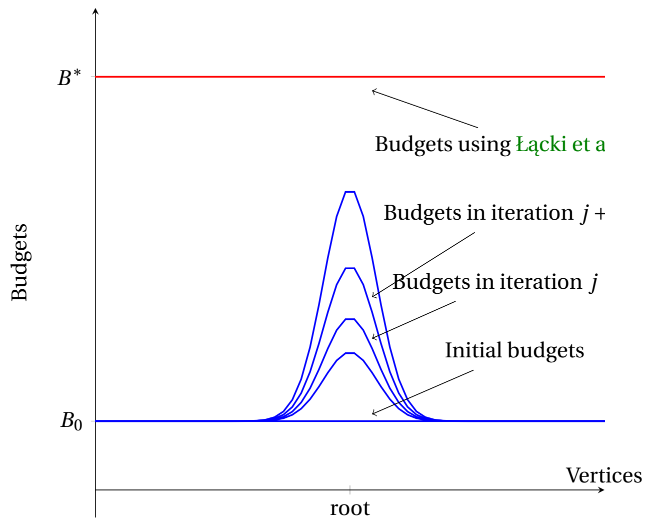


Figure 4.2 – Illustration of the evolution of $B(v,k)$ for all $v \in V$ for a fixed k on a line graph over the iterations of Algorithm 13 and comparing to budgets if one uses Łącki et al. (2020). Vertices are sorted by their order on the line and root is the middle vertex on the line.

Algorithm 12 Stitching algorithm

```
1:  procedure STITCH($G, B$)
2:      for $v \in V$ in parallel do
3:          for $k \in [\ell]$ do
4:              $W_k(v) \leftarrow$ a set of $B(v,k)$ independent uniformly random edges adjacent on $v$
5:      for $j = 1 \ldots \log_2 \ell$ do                                  ▷ See Remark 4.2.1
6:          for $v \in V$ in parallel do
7:              for $k \equiv 1 \ (mod\ 2^j)$ do
8:                  for walk $p \in W_k(v)$ do
9:                      send $(v,k)$ to $z$, where $z \leftarrow$ end vertex of $p$
10:         for $z \in V$ in parallel do
11:             for each message $(v,k)$ do
12:                 if $W_{k+2^{j-1}}(z) = \emptyset$ then
13:                     return Fail
14:                 send $(q,k,z)$ to $v$, where the walk $q \leftarrow$ a randomly chosen walk in $W_{k+2^{j-1}}(v)$
15:                 $W_{k+2^{j-1}}(z) \leftarrow W_{k+2^{j-1}}(z) \setminus \{q\}$
16:         for $v \in V$ in parallel do
17:             for each message $(q,k,z)$ do
18:                 $p \leftarrow$ any walk of length $2^{j-1}$ in $W_k(v)$ with end vertex $z$
19:                 $W_k(v) \leftarrow W_k(v) \setminus \{p\} \cup \{p+q\}$          ▷ $p+q$ is the concatenated walk
20:             for $k \equiv 2^{j-1} \ (mod\ 2^j)$ do
21:                 $W_k(v) \leftarrow \emptyset$
22:     return $W_1(v)$ for all $v \in V$
```

Algorithm 13 Main Algorithm (Budgeting)

```
1: procedure MAIN($G, r, \ell, B^*, \lambda$)
2:     $\theta \leftarrow 10C\ell^2 \log n$, $B_0 \leftarrow 30C\lambda\ell^3 \log n$, $\tau \leftarrow 1 + \sqrt{\frac{20C \log n}{\theta}}$        ▷ Parameter settings
3:     $\forall v \in V$, $B_0(v) \leftarrow B_0 \cdot \deg(v)$
4:     $\forall v \in V$, $\forall k \in [\ell]$: $B(v,k) \leftarrow B_0 \cdot \deg(v) \cdot \tau^{3k-3}$
5:     for $i = 1 \dots \lfloor \log_\lambda B^* \rfloor$ do
6:         $W_1 \leftarrow$ STITCH(G,B)
7:         $W \leftarrow W_1(r)$
8:         for $v \in V$, $k \in [\ell]$ do
9:             $\kappa \leftarrow |\{w \in W | w_k = v\}|$
10:            if $\kappa \geq \theta$ then
11:                $B(v,k) \leftarrow (B_0(v) + \lambda^i \cdot \frac{\kappa}{|W|}) \cdot \tau^{3k-3}$
12:            else
13:                $B(v,k) \leftarrow B_0(v) \cdot \tau^{3k-3}$
14:    $W_1 \leftarrow$ STITCH($G, B$)
15:    $W \leftarrow W_1(r)$
16:    return $W$
```

Remark 4.2.2. *In the above pseudocode (Algorithm 13) τ is a scaling parameter slightly greater then one. We augment all budgets $B(\cdot,k)$ by a factor τ^{3k-3}, to insure that there are always slightly more walks with higher labels, and ensure that stitching succeeds with high probability.*

Analysis. We are now ready to present the main properties of our algorithm:

Lemma 4.2.3 (Correctness and complexity)**.** *Algorithm 13 takes $O(\log \ell \cdot \log_\lambda B^*)$ rounds of MPC communication,* STITCH *terminates without failures with high probability, and the total amount of memory used for walks is $\sum_{v\in V} \sum_{k=1}^{\ell} B(v,k) = O(m\lambda\ell^4 \log n + B^* \lambda\ell)$.*

Proof. Let P^k be distribution of random walks of length k starting from r. That is, the probability that such a walk ends up in $v \in V$ after k steps is $P^k(v)$.

To bound the round complexity we note that each call of STITCH takes only $O(\log \ell)$ rounds of communication, and it is called $\lfloor \log_\lambda B^* \rfloor + 1$ times; this dominates the round complexity. Further rounds of communication are needed to update the budgets. However this can be done in parallel for each vertex, and thus takes only one round per iteration of the outer for-loop (Line 5).

To prove that the algorithm fails with low probability, we must show the following crucial claim about the budgets $B(V,K)$. Recall that the ideal budget in the i^{th} iteration would be $B(v,k) \approx B_0(v) + \lambda^i \cdot P^k(v)$. We show that in reality, the budgets do not deviate too much from this.

Claim 4.2.4. *After iteration i of the outer for-loop (Line 5) in Algorithm 13, with high probability B is set such that*

$$\forall v \in V,\ k \in [\ell]:\ B(v,k) \in \left[(B_0(v) + \lambda^i \cdot P^k(v)) \cdot \tau^{3k-4}, (B_0(v) + \lambda^i \cdot P^k(v)) \cdot \tau^{3k-2}\right].$$

Proof. We first note how $B(r,1)$ – the budget of walks starting at the root vertex – evolves. For $v = r$ and

$k=1$, κ is always equal to $|W|$ – since r is the root vertex – and greater than θ. Therefore, $B(r,1)$ is set in Line 11 of Algorithm 13 to $(B_0(r)+\lambda^i)$. This is important, because it means that when setting other budgets in iteration $i>1$, $|W|$ is always $(B_0(r)+\lambda^{i-1})$, the number of walks rooted from r in the previous round. The exception is the first iteration, when $|W|$ is simply $|B_0(r)|$. In both cases we may say that $|W|\geq\lambda^{i-1}$.

There are two options we have to consider: If after the i^{th} round of Stitch κ exceeded θ, in which case our empirical estimator $\kappa/|W|$ for $P^k(v)$ is deemed reliable. We then use this estimater to set the budget for the next round (see Line 11). Alternately, if κ did not exceed θ, the imperical estimator is deemed too unreliable; we then simply set $B(v,k)$ proportionally to $B_0(v)$ (see Line 13).

Case I ($\kappa<\theta$) then intuitively, κ is too small to provide an accurate estimator of $P^k(v)$. In this case we are forced to argue that the (predictable) term $B_0(v)$ dominates the (unknown) term $P^k(v)$. Since $\kappa<\theta$, $\mathbb{E}(\kappa)=P^k(v)\cdot|W|\leq 2\theta$. (The opposite happens with low probability[2] by Chernoff bounds, since $\theta\geq 10C\log n$.) Therefore,

$$B(v,k)=B_0(v)\cdot\tau^{3k-3}\leq(B_0(v)+\lambda^i\cdot P^k(v))\cdot\tau^{3k-2},$$

and

$$B(v,k)=B_0(v)\cdot\tau^{3k-3}=(B_0(v)+\lambda^i\cdot P^k(v))\cdot\left(1-\frac{\lambda^i\cdot P^k(v)}{B_0(v)+\lambda^i\cdot P^k(v)}\right)\cdot\tau^{3k-3}.$$

So, we need to prove that $1-\frac{\lambda^i\cdot P^k(v)}{B_0(v)+\lambda^i\cdot P^k(v)}\geq\tau^{-1}$. Now, by the above bound on $\mathbb{E}(\kappa)$ as well as the fact that $|W|\geq\lambda^{i-1}$, we have $2\theta\geq P^k(v)\cdot|W|\geq P^k(v)\cdot\lambda^{i-1}$, which results in $\lambda^i\cdot P^k(v)\leq 2\lambda\theta$. Consequently,

$$\left(1-\frac{\lambda^i\cdot P^k(v)}{B_0(v)+\lambda^i\cdot P^k(v)}\right)\geq\left(1-\frac{2\lambda\theta}{B_0(v)+2\lambda\theta}\right)\geq\tau^{-1}.$$

Here we used $B_0(v)\geq B_0\geq 3\lambda\theta\cdot(\sqrt{\theta/(20C\log n)})$, which holds by definition of $B_0(v)$ and our setting of parameters B_0 and θ.

Case II ($\kappa\geq\theta$), then intuitively κ is robust enough to provide a reliable estimator for $P^k(v)$. More precisely, $\kappa/|W|\in\left[\mathbb{E}(\kappa/|W|)\cdot\tau^{-1},\mathbb{E}(\kappa/|W|)\cdot\tau\right]$ with high probability – indeed τ is defined in terms of θ deliberately in exactly such a way that this is guaranteed by Chernoff bounds. $\mathbb{E}(\kappa/|W|)=P^k(v)$, therefore

$$\lambda^i\cdot\frac{\kappa}{|W|}\in\left[\lambda^i\cdot P^k(v)\cdot\tau^{3k-4},\lambda^i\cdot P^k(v)\cdot\tau^{3k-2}\right],$$

and

$$\begin{aligned}B(v,k)&=(B_0(v)+\lambda^i\cdot\frac{\kappa}{|W|})\cdot\tau^{3k-3}\\&\in\left[\left(B_0(v)+\lambda^i\cdot P^k(v)\right)\cdot\tau^{3k-4},(B_0(v)+\lambda^i\cdot P^k(v))\cdot\tau^{3k-2}\right].\end{aligned}$$

[2]Throughout the proof, we say 'low probability' to mean probability of $n^{-\Omega(C)}$ where C can be set arbitrarily high.

□

STITCH only reports failure if for some $v \in V$ and $k \in [2, \ell]$, vertex v receives more requests for walks in $W_k(v)$ than $|W_k(v)| = B(v,k)$. Number of such request is upper bounded by the number of edges ending in v generated by neighbors of v, say w at level $k-1$. That is, the number of requests for $W_k(v)$ is in expectation at most

$$\begin{aligned}\sum_{w\in\Gamma(v)} \frac{1}{d(w)} B(w,k-1) &\leq \sum_{w\in\Gamma(v)} \frac{1}{d(w)} (B_0(w) + \lambda^i \cdot P^{k-1}(w)) \cdot \tau^{3k-5} \\ &= \left(\sum_{w\in\Gamma(v)} \frac{1}{d(w)} B_0(w) + \lambda^i \cdot P^k(v)\right) \cdot \tau^{3k-5} \\ &= \left(d(v)B_0 + \lambda^i \cdot P^k(v)\right) \cdot \tau^{3k-5} \\ &= \left(B_0(v) + \lambda^i \cdot P^k(v)\right) \cdot \tau^{3k-5}.\end{aligned}$$

Since this is greater than θ, the actual number of requests is at most $(B_0(v) + \lambda^i \cdot P^k(v)) \cdot \tau^{3k-4} \leq B(v,k)$, with high probability by Chernoff. Therefore, STITCH indeed does not fail.

Finally, we prove the memory bound. By setting of parameter θ, τ^{3k-2} is at most a constant. Also by the setting of parameters we have,

$$B(v,k) \leq (B_0(v) + \lambda^{\lfloor \log_\lambda B^* \rfloor + 1} \cdot P^k(v)) \cdot \tau^{3k-2} = O(B_0(v) + B^* \lambda \cdot P^k(v)),$$

and $\sum_{v\in V} \sum_{k=1}^{\ell} B(v,k) = O(1) \cdot \sum_{v\in V} \sum_{k=1}^{\ell} (B_0(v) + B^* \lambda \cdot P^k(v)) = O(m\lambda\ell^4 \log n + B^* \lambda \ell)$. □

This gives us the proof of Theorem 4.1.1. Now, one can easily extend this result to the case when multiple sources for the starting vertex is considered. Theorem 4.1.2 is proven in Appendix C.1.

4.3 From random walks to local clustering

We now present two applications for our algorithm to compute random walks. In particular we show how to use it to compute PageRank vectors and how to use it to compute local clustering. In interest of space, we only state here our main results and we defer all the technical definition and proofs to the Appendix.

Approximating PageRank using MPC random walks Interestingly, we can show that we can use our algorithm as a primitive to compute PersonalizedPageRank for a node of for any input vector[3]

Theorem 4.3.1 (Approximating PersonalizedPageRank using MPC random walks)**.** *For any starting single vertex vector s (indicator vector), any $\alpha \in (0,1)$ and any η, there is a MPC algorithm that using $O(\log \ell \cdot \log_\lambda B^*)$ rounds of communication and the total amount of memory of $O(m\lambda\ell^4 \log n + B^* \lambda \ell)$, outputs a vector $\tilde{q}$, such that $\tilde{q}$ is a η-additive approximation to $\mathrm{pr}_\alpha(s)$, where $B^* := \frac{10^6 \log^3 n}{\eta^2 \alpha^2}$ and $\ell := \frac{10 \log n}{\alpha}$.*

[3]We note that Bahmani et al. (2011) also propose an algorithm to compute PersonalizedPageRank vector but with the limitation that this could be computed only for a single node and not for a vector.

The proof is deferred to Appendix C.3.

Using approximate PersonalizedPageRank vectors to find sparse cuts Now we can use the previous result on PersonalizedPageRank to find sets with relatively sparse cuts. Roughly speaking, we argue that for any set C of conductance $O(\alpha)$, for many vertices $v \in C$, if we calculate an approximate $\mathrm{pr}_\alpha(v)$ using our algorithms and perform a sweep cut over it, we can find a set of conductance $O(\sqrt{\alpha \log(\mathrm{Vol}(C))})$. This result is stated in Theorem 4.1.4. The proof of this result is very similar to the proofs of Section 5 of Andersen et al. (2006), however since our approximation guarantees are slightly different, we need to modify some parts of the proof for completeness. The full proof is presented in Appendix Our main result of this subsection is stated below.

4.4 Empirical Evaluation

In this Section we present empirical evaluations of our algorithms for random walk generation, as well as clustering. As our datasets, we use several real-world graphs form the Stanford Network Analysis Project Leskovec and Krevl (2014); Leskovec et al. (2007, 2008); Klimt and Yang (2004); Yang and Leskovec (2012). The graphs are undirected and come mostly (though not in all cases) from 'Networks with ground truth communities', where the clustering application is most relevant. In order to demonstrate the scalability of our main Algorithm we use graphs of varying sizes, as demonstrated in the table below.

Table 4.1 – Summary of the various graphs used in our empirical evaluations.

Name	Vertices	Edges	Description
ca-GrQc	5424	14,496	Collaboration network
email-Enron	36,692	183,831	Email communication network
com-DBLP	317,080	1,049,866	Collaboration network
com-Youtube	1,134,890	2,987,624	Online social network
com-LiveJournal	3,997,962	34,681,189	Online social network
com-Orkut	3,072,441	117,185,083	Online social network

The experiments were performed on Amazon's Elastic Map-Reduce system using the Apache Hadoop library. The clusters consisted of 30 machines, each of modest memory and computing power (Amazon's `m4.large` instance) so as to best adhere to the MPC setting. Each experiment described in this section was repeated 3 times to minimize the variance in performance inherent in distributed systems like this.

Practical considerations. In Section 4.2 we worked with the guarantee that no walks "fail" in the stitching phase. This assumption can be fulfilled at nearly no expense to the (asymptotic) guarantees in space and round complexity of Theorem 4.1.1, and make the proof much cleaner. In practice however, allowing some small fraction of the walks to fail allows for a more relaxed setting of the parameters, and thus better performance.

Each experiment is performed with 15 roots, selected uniformly at random. The main parameters defining the algorithm are as follows: ℓ — the length of a target random walk (16 and 32 in various experiments), C — The number of cycles (iterations of the for-loop in Line 5 of Algorithm 13) performed. , B_0 — the initial budget-per-degree of each vertex, λ — the approximate scaling of the budgets of the root vertices each cycle, τ — a parameter defining the amount of excess budget used in stitching. This is used somewhat differently here than in Algorithm 13. For more details see Appendix C.4.

4.4.1 Scalability

In this section we present the results of our experiments locally generating random walks simultaneously from multiple root vertices. We use the graphs COM-DBLP, COM-YOUTUBE, COM-LIVEJOURNAL, and COM-ORKUT, in order to observe how the runtime of Algorithm 13 scales with the size of the input graph. In each of these graphs, 15 root vertices have been randomly chosen. We ran three experiments with various settings of the parameters, of which one is presented below. For additional experiments see Appendix C.4. B_0 is set to be proportional to n/m – that is inverse proportional to the average degree – since the initial budget of each of each vertex is set to B_0 times the degree of the vertex.

We report the execution time in the Amazon Elastic Map-Reduce cluster, as well as the number of rooted walks generated. Finally, under 'Walk failure rate', we report the percentage of *rooted* walks that failed in the *last cycle* of stitching. This is the crucial quantity; earlier cycles are used only to calibrate the vertex budgets for the final cycle.[4]

Table 4.2 – Experiments with $\ell = 16$, $C = 3$, $B_0 = 6n/m$, $\lambda = 32$, $\tau = 1.4$.

GRAPH	TIME	B_0	ROOTED WALKS GENERATED	WALK FAILURE RATE
COM-DBLP	23 ± 7 **MINUTES**	1.812	$96,362 \pm 2597$	$14.6 \pm 0.6\%$
COM-YOUTUBE	34 ± 6 **MINUTES**	2.279	$53,076 \pm 1185$	$10.8 \pm 0.5\%$
COM-LIVEJOURNAL	76 ± 11 **MINUTES**	0.692	$184,246 \pm 756$	$7.9 \pm 0.1\%$
COM-ORKUT	64 ± 13 **MINUTES**	0.157	$200,924 \pm 1472$	$3.4 \pm 0.0\%$

We observe that Algorithm 13 successfully generates a large number of rooted walks – far more than the initial budgets of the root vertices. As predicted, execution time scales highly sublinearly with the size of the input (recall for example, that COM-ORKUT is more than a hundred times larger than COM-DBLP). The failure rate of walks decreases with the size of the graph in this dataset, with that of COM-ORKUT reaching as low as 3.4% on average; this may be due to the the higher average degree of our larger graphs, leading to the random walks spreading out more.

In Appendix C.4 we report the results of two more experiments, including one with longer walks.

[4] For completeness, we report the empirical standard deviation everywhere. Note, however, that due to the high resource requirement of these experiments, each one was only repeated three times.

4.4.2 Comparison

In this section we compare to the previous work of Łącki et al. (2020) for generating random walks in the MPC model. This work heavily relies upon generating random walks from all vertices simultaneously, with the number of walks starting from a given vertex v being proportional $d(v)$. In many applications, however, we are interested in computing random walks from a small number of root vertices. The only way to implement this using the methods of Łącki et al. (2020) is to start with an initial budget large enough to guarantee the desired number of walks from each vertex.

We perform similar experiments to those in the previous section – albeit on much smaller graphs. Each graph has 15 root vertices chosen randomly, from which we wish to sample random walks. In Table 4.3 we set B_0 to 1 and perform $C = 3$ cycles of Algorithm 13 with $\lambda = 10$, effectively augmenting the budget of root vertices by a factor 100 by the last cycle. Correspondingly, we implement the algorithm of Łącki et al. (2020) – which we call UNIFORM STITCHING – by simply setting the initial budget 100 times higher, and performing only a single cycle of stitching, ie.: $B_0 = 100$, $C = 1$.

Table 4.3 – Experiments with $\ell = 16$, $\lambda = 10$, $\tau = 1.3$. The row labeled Algorithm 13' corresponds to $B_0 = 1$, $C = 3$, while the row labeled 'Uniform Stitching' corresponds to $B_0 = 100$, $C = 1$.

ALGORITHM	CA-GRQC	EMAIL-ENRON	COM-DBLP
ALGORITHM 13	15 ± 1 MINUTES	19 ± 1 MINUTES	18 ± 1 MINUTES
UNIFORM STITCHING	7 ± 0 MINUTES	15 ± 0 MINUTES	66 ± 1 MINUTES

We observe that the running time of our Algorithm 13 barely differs across the three graphs, despite the nearly 100-factor difference between the sizes of CA-GRQC and COM-DBLP. At this small size, the execution time is dominated by setting up the 12 Map-Reduce rounds required to execute Algorithm 13 with these parameters. As expected, the baseline far outperforms our algorithm on the smallest graph; as the size of the input graph grows, however, its running time deteriorates quickly. For larger setting of the parameters UNIFORMSTITCHING can no longer complete on the cluster, due to the higher memory requirement, as we can see in Table C.3 in Appendix C.4.

Maximum Matching Part II

5 Randomized Composable Coresets for Maximum Matching

This chapter is based on joint work with Michael Kapralov and Gilbert Maystre. It has been accepted to the 4th ACM-SIAM Symposium on Simplicity in Algorithms (**SOSA**) 2021 Kapralov et al. (2021d) under the title

Communication Efficient Coresets for Maximum Matching.

5.1 Introduction

Composable coresets is a generic technique for the analysis of large data, that has been shown to be effective for a variety of problems. In the context of graphs, the idea is to partition the edges of the input graph into k parts, extract some small yet informative summary of each part, and recombine these summaries into a single graph without losing too much in the quality of the solution. (The formal definition is presented in Section 5.2.) The small summary of each part is called the composable coreset. This versatile technique translates simply to algorithms in both the MPC and the randomized streaming models (Section 1.1 in Assadi and Khanna (2017)).

The study of *randomized* composable coresets in the context of approximating maximum matching was initiated by Assadi and Khanna (2017) as the usefulness of *deterministic* composable coresets, where the initial partition of the input is arbitrary, was shown to be limited (see e.g. Assadi et al. (2016); Konrad (2015)). They proved that a maximum matching coreset, which contains $n/2$ edges, achieves nearly $1/9$ approximation, which was improved to nearly $1/3$ by Assadi et al. (2019a). This paper further showed that the approximation quality of the maximum matching coreset is at best $1/2$, and proposed an alternative: the EDCS coreset. EDCS's achieve a nearly $2/3$ approximation as randomized composable coresets for maximum matching; they are however significantly denser, with size $n \cdot \mathrm{poly}(\epsilon^{-1}) \cdot \mathrm{poly}(\log n)$ to achieve a $2/3-\epsilon$ approximation. More recently, the work of Assadi and Khanna (2017) gave a coreset of linear size in n that achieves an approximation ratio of $1/2-\epsilon$ for small $\epsilon > 0$, but at the expense of duplicating every edge $\Omega(1/\epsilon)$ times, increasing the communication accordingly. This is again prohibitively expensive for small ϵ.

As the main result of this chapter, we propose a small composable coreset of size at most $n-1$, which nonetheless achieves a $1/2$ approximation ratio, without the need for the duplication of edges.

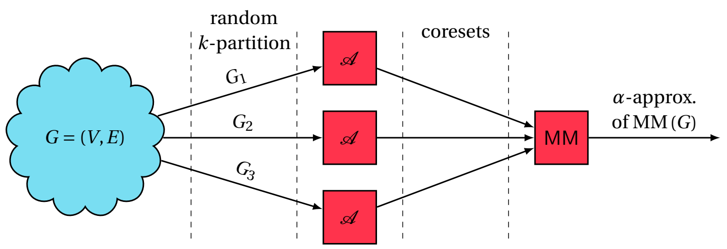


Figure 5.1 – A visual representation of randomized composable coresets. G is first partitioned into k parts randomly. Then each of those parts is reduced into coresets independently using an algorithm $\mathscr{A}$. A maximum matching is then computed amongst the recombination of all the coresets.

Theorem 5.1.1. *There exists a $1/2-o(1)$-approximate randomized composable coreset with size $n-1$ for bipartite maximum matching as long as $k=\omega(1)$, $MM(G)=\omega(k\log n)$.*

Intuition behind our construction. Our coreset is inspired by the previous best known 'small' randomized composable coreset, the maximum matching coreset. The main challenge in analyzing the performance of picking any maximum matching as a coreset lies in the fact that graphs that do not admit a perfect matching generally admits many different maximum matchings. To circumvent this, we propose to use *any matching skeleton* (a structure introduced by Goel et al. (2012), and later rediscovered by Bernstein et al. (2018)) as a coreset. This is essentially a carefully chosen 'canonical' fractional matching that matches vertices as uniformly as possible (see Section 5.4). Such a fractional matching can always be selected to be supported on a forest by a simple argument similar to the one that establishes the integrality of the bipartite matching polytope, meaning that the support size is never larger than $n-1$. The fact that the coreset is essentially an 'optimally spread out' maximum matching leads to a rather simple proof of the approximation ratio of $1/2-o(1)$, which we present in Section 5.5. In Section 5.6, we show that any matching skeleton does not provide a better than $2/3$ approximation to maximum matching, leaving some amount of room for improvement of our approximation ratio bound.

Previous results. Coresets have been studied in a variety of contexts Assadi and Khanna (2017); Badanidiyuru et al. (2014); Balcan et al. (2013); Bateni et al. (2014); Indyk et al. (2014); Mirrokni and Zadimoghaddam (2015); Mirzasoleiman et al. (2013) (also see e.g. Ahn et al. (2012a,b); Assadi et al. (2016); Bhattacharya et al. (2015); Bulteau et al. (2016); Chitnis et al. (2016); Kapralov et al. (2017a); Kapralov and Woodruff (2014); McGregor et al. (2015); Filtser et al. (2020) for the related work in the linear sketching context). Related to our problem, maximum matching approximation has been widely studied in low space regimes such as MPC Assadi et al. (2019a); Harvey et al. (2018); Czumaj et al. (2018); Lattanzi et al. (2011); Assadi and Khanna (2017) and streaming McGregor and Vorotnikova (2018); Goel et al. (2012); Louis (2015); Spielman and Srivastava (2011); Assadi and Khanna (2017); Assadi and Bernstein (2019). In particular Assadi et al. (2019a) achieves nearly $2/3$ approximate maximum matching in two MPC rounds and $\widetilde{O}(\sqrt{mn}+n)$ space per machine, using randomized composable coresets of size $O(n\log n)$.

5.2 Randomized Composable Coresets

Definition 5.2.1. *Let $G = (V, E)$ be a graph and $k \in \mathbb{N}$ and integer. A* **random k-partition** *of G is a set of k random subgraphs $\{G_i = (V, E_i)\}_{i\in[k]}$ of G, where each edge $e \in E$ is sent uniformly at random to exactly one of the E_i.*

Definition 5.2.2. *Mirrokni and Zadimoghaddam (2015) Let $\mathscr{A}$ be an algorithm that takes as input a graph H and returns a subgraph $\mathscr{A}(H) \subseteq H$. We say that $\mathscr{A}$ outputs an* **α-approximate randomized composable coreset** *for the maximum matching problem if given any graph $G = (V, E)$, any $k \in \mathbb{N}$ and a random k-partition of G we have*

$$\alpha \cdot MM(G) \leq \mathbb{E}MM(\mathscr{A}(G_1) \cup \cdots \cup \mathscr{A}(G_k))$$

where the expectation is taken over the randomness of the partition. The **size** *of the coreset is the number of edges returned by $\mathscr{A}$.*

Remark 5.2.3. *Throughout this chapter we will assume some natural bounds on the parameter k. Firstly, similarly to Assadi and Khanna (2017); Assadi et al. (2019a), we suppose that the maximum matching size of the input graph $MM(G) = \omega(k \log n)$. This allows us to argue concentration at various places in the analysis, and is a natural assumption: The regime where $MM(G)$ is smaller is handled in Chitnis et al. (2015). We will further make the natural assumption that $k = \omega(1)$, that is we parallelize over a superconstant number of machines.*

5.3 Preliminaries and Notation

Throughout the chapter we consider bipartite graphs, denoted by $G = (P, Q, E)$, where the vertex-sets P and Q are the two sides of the bipartition, and E is the edge-set. We let $n = |P \cup Q|$ denote the number of vertices in G and $m = |E|$ denote the number of edges. For a vertex $v \in P \cup Q$ of G we write $\Gamma_G(v)$ to denote the set of neighbors of v in G, or $\Gamma(v)$ if G is clear from context. Similarly, for a set $S \subseteq P \cup Q$ we write $\Gamma_G(S)$ or $\Gamma(S)$ to the denote the neighborhood of the set in G.

Definition 5.3.1. *A* **matching** *in a graph is a set of edges such that no two of them share an end point. The* **maximum matching size** *of a graph is the maximum possible size of a matching in it; we usually denote it $MM(G)$.*

Definition 5.3.2. *Given a graph $G = (P, Q, E)$, a* ***fractional matching*** *is a set of non-negative edge weights $\vec{x} : E \to [0, 1]$ such that no vertex has more than unit weight adjacent on it:*

$$\forall v \in P \cup Q : \sum_{w\in\Gamma(v)} x_{vw} \leq 1$$

The ***size*** *of a fractional matching is the sum of all edge-weights.*

Note that an integral fractional matching corresponds to the classical matching definition. We will also use the extended notion of α-matching of Goel et al. (2012), which are classical fractional matching with a changed constraint for one side of the bipartition.

Definition 5.3.3. *Given a graph $G = (P, Q, E)$, a α**-matching with respect to** P is a set of non-negative edge weights $\vec{x} : E \to [0,1]$ that saturates each vertex of P fractionally exactly α times and each vertex of Q at most once.*

Definition 5.3.4. *A vertex cover is a set of vertices $\Phi \subseteq P \cup Q$ such that all edges have at least one end point in Φ.*

The following theorem is a fundamental fact about bipartite graphs, on which we will be relying throughout the chapter.

Theorem 5.3.5. *For any bipartite graph, the size of the maximum matching, the size of the maximum fractional matching, and the size of the minimum vertex cover are equal.*

Corollary 5.3.6. *If a matching and a vertex cover have the same size, both are optimal.*

Furthermore, we will rely on the following concentration inequality.

Theorem 5.3.7 (Chernoff bound, see e.g. Alon and Spencer (2008))**.** *Let $Y = \sum_{i=1}^{n} X_i$ be the sum of n independent binary random variable each having $\Pr X_i = 1 = p_i$. Let $\mu_Y = \mathbb{E} Y = \sum_{i=1}^{n} p_i$. Then, for any $\epsilon \in (0,1)$, we have:*

$$\Pr X \notin (1 \pm \epsilon)\mu_Y \leq 2e^{-\frac{\epsilon^2 \mu_Y}{3}}$$

5.4 Our coreset: the matching skeleton

In this section, we recall the notion of matching skeleton, introduced by Goel et al. (2012) and later rediscovered by Bernstein et al. (2018). We simplify slightly the original definitions and results to suit our needs. We also introduce a new related object, the canonical vertex cover which is central to our proof.

We define a partition of the vertex set of G into subgraphs of varying vertex expansion as follows. For each $i = 1, \ldots$ we define a tuple (P_i, Q_i, α_i) iteratively as follows, starting with $G_0 = G$, $P_0 = P$:

1. Let $\alpha_i = \min_{\emptyset \neq S \subseteq P_{i-1}} \frac{|\Gamma_{G_{i-1}}(S)|}{|S|}$
2. Let $P_i =$ largest $S \subseteq P_{i-1}$ such that $\frac{|\Gamma_{G_{i-1}}(S)|}{|S|} = \alpha_i$
3. Let $Q_i = \Gamma_{G_{i-1}}(P_i)$
4. $G_i = G_{i-1} \setminus (P_i \cup Q_i)$

This process continues until G_i is empty.

Definition 5.4.1. *We call each (P_i, Q_i, α_i) a **block** and α_i its **expansion level**, which is carried over to the vertices of the block using the notation $\alpha(v) := \alpha_i$ for $v \in P_i \cup Q_i$.*

We call the collection $\{(P_i, Q_i, \alpha_i)\}_{i\in[k]}$ *the* **block decomposition** *of* G.

Remark 5.4.2. *A practical way to find* α_1 *is to solve several max-flow instances. For some* $\alpha \in \mathbb{R}_+$, *let* G_α *be a copy of* G *where edges are directed from* P *to* Q *with an infinite weight. Also part of* G_α *is a source vertex* s *which has edges directed toward each* $p \in P$ *with weight* α *and a sink vertex* t *with unit-weight edges incoming from each* $q \in Q$. *Observe that* $\alpha_1 = \inf\{\alpha \in \mathbb{R}_+ : |\mathsf{MC}(\alpha)| > 1\}$ *where* $\mathsf{MC}(\alpha)$ *is the min-cut containing* s *in* G_α. *Finding* α_1 *thus reduces to solving max-flow instances with increasing* α *until a non-trivial min-cut is found. This cut actually consists of* P_1 *and* Q_1 *together with* s. *The remaining of the partition is obtained by repeating this argument.*

We now recall the main properties of the block decomposition of G.

Lemma 5.4.3 (Section 3 in Goel et al. (2012))**.** *Let* $\{(P_i, Q_i, \alpha_i)\}_{i\in[k]}$ *be the block partition of* G. *The sequence* $(\alpha_i)_{i\in[k]}$ *is strictly increasing and such that* $\alpha_i = |Q_i|/|P_i|$. *Also, for any* $i \in [k]$:

$$\Gamma(P_i) \subseteq \bigcup_{j\le i} Q_j$$

Intuitively, each block $P_i \cup Q_i$ is associated with a certain expansion of the P_i side, namely α_i. The expansion of the block cannot be greater than α_i, as $|Q_i| = \alpha_i |P_i|$. However, it is also no less than α_i, as the entire block admits of an α_i-matching with respect to P_i.

Lemma 5.4.4 (Section 3 in Goel et al. (2012), Lemma 12 in Bernstein et al. (2018))**.** *Let* $G = (P, Q, E)$ *be a graph together with its block decomposition* $\{(P_i, Q_i, \alpha_i)\}_{i\in[k]}$. *For each* $i \in [k]$ *there is an* α_i*-matching of* $P_i \cup Q_i$ *with respect to* P_i.

Remark 5.4.5 (Section 3 in Goel et al. (2012))**.** *The above* α*-matchings can easily be made to have cycle-free supports, by eliminating cycles through standard techniques.*

Now that the block decompositon of a graph is introduced, we can define matching skeletons which are simply the union of the above introduced cycle-free α-matching for each block.

Definition 5.4.6 (Matching skeleton Goel et al. (2012))**.** *Let* $G = (P, Q, E)$ *be a graph together with its block decomposition* $\{(P_i, Q_i, \alpha_i)\}_{i\in[k]}$. *For each* $i \in [k]$, *let* $\vec{x}_i : (P_i \times Q_i) \cap E \to [0, 1]$ *be a cycle-free* α_i*-matching. We call*

$$H = \bigcup_{i\in[k]} \operatorname{supp}(\vec{x}_i)$$

a matching skeleton of G. *See Fig. 5.1 for a visual example.*

Remark 5.4.7. *The matching skeleton coreset has size at most* $|C| + |S| - 1$, *as it is always a forest.*

We now describe a special kind of vertex cover, related with the block partition of a graph, which will be a crucial tool in analyzing the quality of our coreset.

Definition 5.4.8 (Canonical vertex cover)**.** *Given a graph* $G = (P, Q, E)$ *together with its block decomposition, we call*

$$\Phi = \{q \in Q | \alpha(q) < 1\} \cup \{p \in P | \alpha(p) \ge 1\}$$

the **canonical vertex cover** *of* G. *That is, in each bock we take the smaller side of the bipartition.*

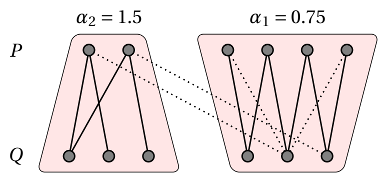


Figure 5.2 – A graph $G = (P, Q, E)$ and its block partition. A cycle-free matching skeleton is shown with solid edges. Due to the construction of the block partition, edges between P_2 and Q_1 cannot be part of any matching skeleton. Also, from Lemma 5.4.3, no edge can exist between Q_2 and P_1.

Lemma 5.4.9. *The canonical vertex cover is a minimum vertex cover.*

Proof. First we show that Φ is indeed a vertex cover. Suppose there exists an edge $\{p, q\}$ not adjacent on Φ and let $p \in P_i$ and $q \in Q_j$. By definition of the canonical vertex cover, this means that $\alpha_j \geq 1 > \alpha_i$ and hence $i < j$ using monotonicity of the expansion levels (Lemma 5.4.3). In turn, this implies that $\Gamma(p) \not\subseteq \bigcup_{\ell \leq i} Q_\ell$: a contradiction with Lemma 5.4.3.

We now proceed to show that Φ is minimum, by showing that there exists a fractional matching of size $|\Phi|$ (See Corollary 5.3.6). We define this fractional matching block-by-block: Consider the block (P_i, Q_i, α_i).

- If $\alpha_i < 1$, then $\Phi \cap (P_i \cup Q_i)$ is exactly Q_i. In this case, we can take the α_i-matching with respect to P_i as our fractional matching. This will have size $\alpha_i |P_i| = |Q_i|$, exactly as desired.
- On the other hand, if $\alpha_i \geq 1$, then $\Phi \cap (P_i \cup Q_i)$ is exactly P_i. Then, an α_i-matching with respect to P_i *scaled down by a factor of* α_i is a valid fractional matching of the block, and has size $|P_i|$.

□

The above deduction also shows that any matching skeleton contains a maximum matching. Therefore, the matching skeleton coreset performs at least as well as the maximum matching coreset of Assadi and Khanna (2017). In particular, this directly yields a lower bound of $1/3$ on the approximation ratio of our coreset. However, a matching skeleton retains more information from the input graph, as the entire block partition can be recovered from it. This allows for a better approximation ratio as Section 5.5 demonstrates.

Remark 5.4.10. *Let us draw the parallels between the server flows of Bernstein et al. (2018) and the notion of matching skeleton of Goel et al. (2012). In the context of Bernstein et al. (2018), the support of a realization of the balanced server flow is simply a matching skeleton. The balancedness condition corresponds to the neighboring property of Lemma 5.4.3. Finally, the server flow values of Bernstein et al. (2018) are exactly the expansion levels.*

Finally, we prove a structural result about the robustness of the block decomposition under changes to the edge-set of G. This will be crucial to our proofs in both Sections 5.5 and 5.6.

Lemma 5.4.11. *Let $G = (P, Q, E)$ be a graph together with its block decomposition $\{(P_i, Q_i, \alpha_i)\}_{i\in[k]}$. let H be a matching skeleton of G. Now, let $G' = (P, Q, E')$ be a modification of G with edges added and removed in a specific way: $E' = (E \cup E^+) \setminus E^-$ such that*

- *Each edge $\{p, q\} \in E^+$ obeys $\alpha(p) \geq \alpha(q)$,*
- *and $E^- \cup H = \emptyset$.*

Then the block decomposition of G' is still $\{(P_i, Q_i, \alpha_i)\}_{i\in[k]}$, and therefore H remains a valid matching skeleton of G'.

Proof. We will use $\Gamma(S)$, $\Gamma'(S)$ and $\Gamma_H(S)$ to denote the neighborhood of some set S in the graphs G, G' and H respectively. Consider now the first step of the block decomposition of G'. We first prove that no set $S \subseteq P$ has lower expansion than α_1 in G'. Consider any set $S \subseteq P$. We can lower bound the size of $\Gamma'(S)$ by $\Gamma_H(S)$ since $H \subseteq G'$. Moreover, we note that H contains the support of an α_i-matching with respect to P_i, in block $(P_i \cup Q_i)$, for each i. Therefore, the expansion of any subset in P_i is at least α_i and

$$|\Gamma_H(S \cap P_i) \cap Q_i| = |\Gamma_H(S \cap P_i)| \geq \alpha_i |S \cap P_i|.$$

The equality comes from the fact that a matching skeleton contains no edge crossing two blocks. Using this, we have:

$$|\Gamma'(S)| \geq |\Gamma_H(S)| \geq \sum_{i=1}^{k} |\Gamma_H(S \cap P_i) \cap Q_i| = \sum_{i=1}^{k} |\Gamma_H(S \cap P_i)| \geq \sum_{i=1}^{k} \alpha_i |S \cap P_i| \geq \alpha_1 |S|$$

Note that the statement is true with strict inequality when $S \not\subseteq P_1$. On the other hand, the expansion of P_1 in G' is exactly α_1, as $\Gamma'(P_1) = Q_1$. This is because E^+ cannot have any edge between P_1 and $Q \setminus Q_1$.

We thus have proven that the first block in the decomposition of G' is (P_1, Q_1, α_1). One can then proceed by induction on i to prove that the same is true for the i^{th} block. The argument is identical to the base case by observing that since E^+ cannot have edges between P_i and $\bigcup_{j=i+1}^{k} Q_j$, it does not increase the expansion of P_i. □

5.5 Main Result

Having defined the matching skeleton coreset, we now prove a lower bound of nearly 1/2 on its effectiveness. This improves upon any known lower bound for a randomized composable coreset of size $O(n)$ for the maximum matching problem.

Theorem 5.5.1. $\mathsf{MatchingSkeleton}(G)$ *constitutes a $(1/2 - o(1))$-approximate randomized composable coreset for maximum matching in any bipartite graph $G = (P, Q, E)$ where $k = \omega(1)$ and the maximum matching size $MM(G) = \omega(k \log n)$.*

Proof. Our analysis is inspired by the maximum matching coreset analysis of Assadi et al. (2019a), however, we achieve a better approximation ratio using more subtle techniques. Let μ denote $\mathrm{MM}(G)$. Recall that by the definition of randomized composable coresets (Definition 5.2.2) we must randomly edge-partition $G = (P, Q, E)$ into k subgraphs $G_1, \dots, G_k$, and show that the union of each coresets,

$$\widetilde{G} = \bigcup_{i=1}^{k} \mathsf{MatchingSkeleton}(G_i), \tag{5.1}$$

has an expected maximum matching size of $\mu \cdot (1/2 - o(1))$, over the randomness of the k-partition.

We begin by choosing an arbitrary maximum matching M^* of G. We separate G in two parts: M^* and $G^- := G \backslash M^*$ for the purposes of analysis, and for every $i = 1, \dots, k$, let

$$G_i^- := G_i \cap G^-.$$

We will show the stronger statement that even under **adversarial partitioning** of G^-, Eq. (5.1) holds, as long as M^* is partitioned randomly. From now on we will assume that the partition into $G_1^-, \dots, G_k^-$ is fixed arbitrarily; we will show that either at least one of G_i^- contains a large matching or $M^* \cap \widetilde{G}$ is large.

Consider an arbitrary k-partitioning of G^- into $G_1^-, \dots, G_k^-$ and let the maximum matching size of G_i^- be μ_i^-. If even one of μ_i^- is at least $\mu/2$, we are done. Indeed, following Lemma 5.4.9, any matching skeleton of G_i will contain a maximum matching, that is a matching of size $\mathrm{MM}(G_i) \ge \mu_i^- \ge \mu/2$, and hence so will $\widetilde{G}$. Therefore, we can focus on the case where $\max_{i=1}^{k} \mu_i^- \le \mu/2$ and use the following lemma, which is our main technical contribution:

Lemma 5.5.2 (Main lemma)**.** *Consider an arbitrary partitioning of G^- where $\max_{i=1}^{k} \mu_i^- < \mu/2$. Let e be a uniformly random element of M^*. Then*

$$\Pr e \in \widetilde{G} \ge 1/2 - o(1),$$

where probability is taken over the randomness of the partitioning of M^* as well as the randomness of the choice of e.

The above lemma relies on a subtle probabilistic argument, and is formulated in terms of a uniformly random edge of M^* for technical reasons. However, an immediate consequence of it is that at least nearly half of the edges of M^* will be taken in $\widetilde{G}$. This follows by linearity of expectation:

$$\mathbb{E}\left|M^* \cap \widetilde{G}\right| = \mathbb{E} \sum_{e \in M^*} \mathbb{1}(e \in \widetilde{G}) = \sum_{e \in M^*} \Pr e \in \widetilde{G} \ge \mu \cdot (1/2 - o(1)).$$

where the last inequality follows by Lemma 5.5.2. We have proven that Eq. (5.1) holds under adversarial partitioning of G^- both when $\max_{i=1}^{k} \mu_i^- \ge \mu/2$ and when $\max_{i=1}^{k} \mu_i^- < \mu/2$, which implies the statement of the theorem. □

We conclude the analysis of the MatchingSkeleton coreset by proving Lemma 5.5.2.

Proof of Lemma 5.5.2: Without loss of generality we may assume that $e \in G_1$. We know that the maximum matching size of G_1^- is at most $\mu/2$. Consider now adding to G_1^- all edges of $M^* \cap G_1$ *except for* e. Since the size of $M^* \cap G_1 \setminus \{e\}$ is at most $2\mu/k$ with high probability by Theorem 5.3.7, the maximum matching size does not increase by more than $2\mu/k$.

We base our analysis on fixing the outcome of the random graph $G_1 \setminus \{e\}$ to be some fixed H. We refer to this event that $G_1 \setminus \{e\} = H$ as $\mathscr{E}(H)$. Suppose that indeed the maximum matching size of H is at most $\mu \cdot (1/2 + 2/k)$, and hence that H has a canonical vertex cover Φ of this size. Recall from Definition 5.4.8 that the canonical vertex cover contains exactly the vertices of Q with α-value *strictly less* than one and the vertices of P with α-values *at least* one. Therefore, any new edge added to H that is *not* adjacent on Φ *must be* included in any matching skeleton, as we show in the following paragraph.

Indeed, consider $e = \{p, q\}$ to be such an edge, and suppose that there exists some matching skeleton H of G_1 where e is not included. This means, by Lemma 5.4.11 with $E^- = \{e\}$ and $E^+ = \emptyset$ that the block decompositions of G_1 and H are identical. However, by definition of the canonical vertex cover Φ for H and because $p, q \notin \Phi$, we have $\alpha_H(p) < 1 \le \alpha_H(q)$. This implies that in the block partition of G_1, $p \in P_i$ and $q \in Q_j$ with $i < j$, which is a contradiction of Lemma 5.4.3.

Consequently, if e is not adjacent on Φ, it must be taken into MatchingSkeleton(G_1). The last important observation is that the distribution of e, when conditioned on $\mathscr{E}(H)$, is uniform on $M^* \setminus H$. Indeed, this conditioning in no way breaks the symmetry between the unsampled edges $M^* \setminus H$, and e is equally likely to be any of them. Therefore, e is uniformly distributed among at least $\mu \cdot (1 - 2/k)$ edges among which at most $\mu \cdot (1/2 + 2/k)$ are adjacent on Φ: Conditioned on $\mathscr{E}(H)$, where $\mathrm{MM}(H) \le \mu \cdot (1/2 + 2/k)$, the probability that e is not adjacent on Φ and therefore $e \in$ MatchingSkeleton(G_1) is at least $1/2 - o(1)$.

The above deduction was made with the assumption that $\mathrm{MM}(H) \le \mu \cdot (1/2 + 2/k)$. However, recall that this happens with high probability by Theorem 5.3.7, therefore we can extend the result to full generality. Consider the possible outcomes of $G_1 \setminus \{e\}$ to form the family $\mathscr{H}$. We can split $\mathscr{H}$ into the disjoint union of $\mathscr{H}_0$ and $\mathscr{H}^*$, where $\mathscr{H}^*$ comprises the anomalous outcomes where the maximum matching size of H is greater than $\mu \cdot (1/2 + 2/k)$. Then,

$$\begin{aligned}
\Pr e \in \mathsf{MatchingSkeleton}(G_1) &= \sum_{H \in \mathscr{H}} \Pr \mathscr{E}(H) \Pr e \in \mathsf{MatchingSkeleton}(G_1) | \mathscr{E}(H) \\
&\ge \sum_{H \in \mathscr{H}_0} \Pr \mathscr{E}(H) \Pr e \in \mathsf{MatchingSkeleton}(G_1) | \mathscr{E}(H) \\
&\ge \sum_{H \in \mathscr{H}_0} \Pr \mathscr{E}(H) \cdot (1/2 - o(1)) \\
&= \Pr G_1 \setminus \{e\} \notin \mathscr{H}^* \cdot (1/2 - o(1)) \\
&\ge 1/2 - o(1),
\end{aligned}$$

as desired. □

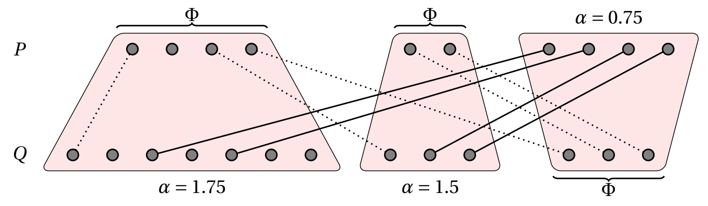


Figure 5.3 – A visual representation of the block decomposition of H. For clarity, edges of H are not shown but the edges of $M^\star \setminus H$ are. If e turns out to be one of the solid edges it *must* be taken into $\mathsf{MatchingSkeleton}(G_1)$; however, if it is one of the dotted edges, it might not be.

5.6 Limitations of the server flow coreset

In this section we show the limits of the server flow forest as a randomized composable coreset for maximum matching by constructing a pathological bipartite graph on which it only preserves the maximum matching size to a factor of $2/3$.

Theorem 5.6.1. *For large enough n and k such that $k = O(n/\log n)$, $k = \omega(1)$, there exists a bipartite graph G on n vertices with maximum matching size μ, for which the maximum matching size of*

$$\widetilde{G} = \bigcup_{i=1}^{k} \mathsf{MatchingSkeleton}(G_i)$$

is at most $\mu \cdot (2/3 + o(1))$ with high probability.

Remark 5.6.2. *Note that here the high probability is over the randomness of the partition. The choice of the matching skeleton is considered to be adversarial in each subgraph, among the multiple possible valid choices.*

We begin by defining the graph $G = (P, Q, E)$. The construction follows the ideas used inAssadi et al. (2019a) to prove an upper bound on the performance of the maximum matching coreset. Let the vertex-set of G consist of six parts: P_1, P_2, and P_3 make up P on one side of the bipartition and Q_1, Q_2, and Q_3 make up Q on the other side. Let the sizes of P_1, P_2, Q_2, and Q_3 be r, and let the sizes of Q_1 and P_3 be $r + 2r/k$, where r is some parameter such that $6r + 4r/k = n$. The edge-set E is comprised of the following:

- A perfect matching between all of P_1 and a subset of Q_1,
- a complete bipartite graph between Q_1 and P_2,
- a perfect matching between P_2 and Q_2,
- a complete bipartite graph between Q_2 and P_3,
- and a perfect matching between a subset of P_3 and all of Q_3.

The graph is pictured in Fig. 5.4.

The analysis of the behavior of MatchingSkeleton on this graph relies on the observation that in a typical subsampled version, $P_1 \cup Q_1 \cup P_2$ forms a region of α-value at least 1 while $Q_2 \cup P_3 \cup Q_3$ forms a region of α-value at most 1. This means that the edges sampled between P_2 and Q_2 need not be taken into the server flow, which further implies that $\widetilde{G}$ can be missing the entire (P_2, Q_2) matching.

In order to prove this we will need the following basic property of expansion levels. One side of the lemma has been previously shown in Bernstein et al. (2018).

Lemma 5.6.3. *Consider a bipartite graph $G = (P, Q, E)$.*

- *If P can be perfectly matched to Q, then* $\min \alpha \geq 1$.
- *Conversely, if Q can be perfectly matched to P, then* $\max \alpha \leq 1$.

Proof. By optimality of the canonical vertex cover (Lemma 5.4.9), and by Theorem 5.3.5 we have that the size of the maximum matching is

$$\sum_{i=1}^{k} \begin{cases} |P_i| \text{ if } \alpha_i \geq 1 \\ |Q_i| \text{ if } \alpha_i < 1 \end{cases} .$$

In the first case of the lemma, $\mathrm{MM}(G) = |P| = \sum_{i=1}^{k} |P_i|$, therefore α_i must always be at least 1. In the second case, $\mathrm{MM}(G) = |Q| = \sum_{i=1}^{k} |Q_i|$, therefore α_i must always be at most 1. □

Finally, we state a result on perfect matchings in random bipartite graphs. This is a simplification, and direct result of Corollary 7.13 from Bollobás and Frieze (1985).

Theorem 5.6.4. *Let H be a random bipartite graph on $n + n$ vertices, where each of the n^2 possible edges appears independently with probability $p = \Omega(\log n/n)$. Then H contains a perfect matching with high probability.*

We are ready to prove Theorem 5.6.1.

Proof of Theorem 5.6.1:

Consider $G_i = (P, Q, E_i)$, the graph G sub-sampled at rate $1/k$. We claim that with high probability the non-isolated vertices of $P_1 \cup P_2$ can be perfectly matched to Q_1. Indeed, of r edges of $P_1 \times Q_1$, r/k are expected to appear in G_i and with high probability no more than $2r/k$ do (see Theorem 5.3.7). In this case, at least r unmatched vertices of Q_1 remain, which we will call Q_1'. Note that the graph between Q_1' and P_2 follows the same distribution as the random graph described in Theorem 5.6.4, with $p = 1/k = \Omega(\log n/n)$. Therefore, $(Q_1' \times P_2) \cap E_i$ contains a perfect matching with high probability.

By Lemma 5.6.3, this means that the subgraph induced by $P_1 \cup Q_1 \cup P_2$ in G_i has block decomposition with all $\alpha \geq 1$. By similar reasoning we can show that the non-isolated vertices of $Q_2 \cup Q_3$ can be perfectly matched to P_3. Hence, by Lemma 5.6.3, the induced subgraph of $Q_2 \cup P_3 \cup Q_3$ in G_i has a block decomposition with all $\alpha \leq 1$.

Simply taking the disjoint union of these two induced subgraphs does not change the expansion levels. Hence the graph G_i^-, consisting of all edges of G_i *except those between* P_2 *and* Q_2, has block decomposition with the α values of P_1, Q_1, and P_2 being at least 1, and the α values of Q_2, P_3 and Q_3 being at most 1. Let H be a matching skeleton of G_i^-. By applying Lemma 5.4.11 with $E^- = \emptyset$ and $E^+ = E_i \cap P_2 \times Q_2$, we get that H is still a matching skeleton of G_i. Therefore, there exists a matching skeleton of G_i which contains no edges from $P_2 \times Q_2$.

In conclusion, it is possible that each coreset selects a matching skeleton of its sub-graph containing no edges from $Q_2 \times P_2$. In such case, the maximum matching of $\widetilde{G}$ has size at most $2r + 4r/k$, whereas that of G was $3r$. $\square$

Remark 5.6.5. *With a simple alteration to the proof, it can be shown that this upper bound holds even when the individual matching skeletons are selected arbitrarily.*

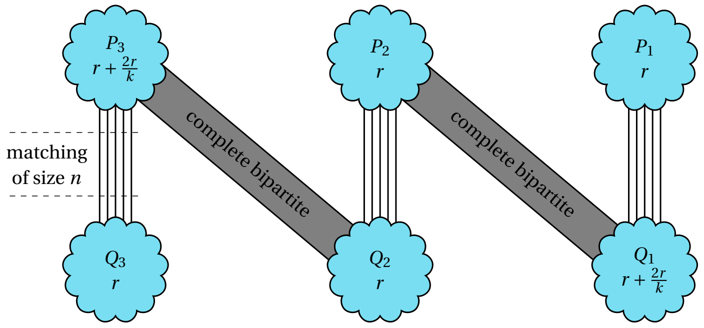


Figure 5.4 – A typical sub-sampling of the graph has a matching skeleton that does not contain any edges of $Q_2 \times P_2$.

6 Highly Space-Efficient Maximum Matching

This chapter is based on joint work with Michael Kapralov, Ashkan Norouzi-Fard, and Slobodan Mitrović. It has been accepted to the 31st ACM-SIAM Symposium on Discrete Algorithms (**SODA**) 2020 Kapralov et al. (2020a) under the title

Space Efficient Approximation to Maximum Matching Size from Uniform Edge Samples.

6.1 Introduction

Large datasets are prevalent in modern data analysis, and processing them requires algorithms with a memory footprint much smaller than the size of the input, i.e. sublinear space. The streaming model of computation, which captures this setting, has received a lot of attention in the literature recently. In this model the edges of the graph are presented to the algorithm as a stream in some order. It has recently been shown that randomly ordered streams allow for surprisingly space efficient estimation of graph parameters by nontrivial memory vs sample complexity tradeoffs (see, e.g. Kapralov et al. (2014); Monemizadeh et al. (2017); Peng and Sohler (2018) for approximating matching size and other graph properties such as number of connected components, weight of MST and independent set size). Memory vs sample complexity tradeoffs for learning problems have also recently received a lot of attention in literature Raz (2016, 2017); Kol et al. (2017); Garg et al. (2018); Beame et al. (2018). In this chapter we study the following question:

How many iid samples of edges of a graph G are necessary and sufficient for estimating the size of the maximum matching in G to within a constant factor?
Can such an estimate be computed using small space?

We give nearly optimal bounds for both questions, developing a collection of new techniques for efficient simulation of matching algorithms by random sampling. Our main result is

Theorem 6.1.1. *There exists an algorithm (Algorithm 15) that, given access to iid edge-samples of a graph $G = (V, E)$ with n vertices and m edges produces a constant factor approximation to maximum matching* *size in G using $O(\log^2 n)$ bits of memory and at most m samples with probability at least* $4/5$.

The sample complexity of Algorithm 15 is essentially optimal: for every constant C, every m between $n^{1+o(1)}$ and $\Omega(n^2)$ it is information theoretically impossible to compute a C-approximation to maximum matching size in a graph with high constant probability using fewer than $m^{1-o(1)}$ iid samples from the edge set of G, even if the algorithm is not space bounded.

The proof of the upper bound part in Theorem 6.1.1 is given in Section 6.4 (more precisely, Section 6.4.2). The proof of the lower bound part of Theorem 6.1.1 follows from Theorem 6.8.1 in Section 6.8.

The core algorithmic tool underlying Theorem 6.1.1 is a general method for implementing peeling type algorithms efficiently using sampling. In particular, our approach almost directly yields a constant factor approximate local computation algorithm (LCA) for maximum matching in a graph G with degrees bounded by d using $O(d \log n)$ queries and $O(\log d)$ exploration depth, whose analysis is deferred to Section 6.6.

Theorem 6.1.2. *Let G be a graph with n vertices and maximum degree d. Then there exists a random matching M, such that $\mathbb{E}[|M|] = \Theta(MM(G))$, and an algorithm that with high probability:*

- *Given an edge e of G, the algorithm reports whether e is in M or not by using $O(d \log n)$ queries.*
- *Given a vertex v of G, the algorithm reports whether v is in M or not by using $O(d \log n)$ queries.*

Moreover, this algorithm can be executed by using $O(d \log^3 n)$ bits of memory.

We note that the most efficient LCA's for matching Levi et al. (2017) require $d^4 \log^{O(1)} n$ exploration to achieve a constant factor approximation, and are based on the idea of simulating the randomized greedy algorithm locally. Indeed, this runtime complexity follows from the beautiful result of Yoshida et al. Yoshida et al. (2009) or Onak et al. Onak et al. (2012) that shows the size of the query tree of the randomized greedy is $O(d)$ *in expectation over the starting edge.* Applying a Markov bound to discard vertices/edges on which the exploration takes too long leads to the desired complexity. Moreover, in a degree d bounded graph matching size could be as small as n/d, with only a $1/d$ fraction of vertices and edges involved in the matching. Therefore, a multiplicative (as opposed to multiplicative-additive) constant factor approximation based on average case results of Yoshida et al. (2009); Onak et al. (2012) must use a Markov bound with a loss of at least a factor of d in the size of the query tree with respect to the average, and hence cannot lead to a better than $O(d^2)$ runtime overall. At the same time, Theorem 6.1.2 yields the near-optimal runtime of $O(d \log n)$, going well beyond what is achievable with the above mentioned average case results.

At this point it is natural to wonder if the average case analysis of Yoshida et al. (2009); Onak et al. (2012) can be improved to show that the size of the query tree is $O(d)$ in expectation *for any given edge* as opposed to for a random one. Surprisingly, we show in Section 6.7 that this is not possible:

Theorem 6.1.3. *There exists an absolute constant $b > 0$ such that for every n, $d \in [5, \exp(b\sqrt{\log n})]$ and $\epsilon \in [1/d, 1/2]$ there exists a graph G with n vertices and maximum degree $d+1$, and an edge e such that* *running* YYI-MAXIMAL-MATCHING(e,π) *from Algorithm 23 results in an exploration tree of size at least*

$$\frac{1}{8}\cdot\epsilon\cdot\left(\frac{d}{2}\right)^{2-\epsilon},$$

in expectation.

In Section 6.9 we prove that our algorithm is robust to correlations that result from replacing the iid stream of edge-samples with a random permutation stream. Formally, we prove

Theorem 6.1.4. *There exists an algorithm that given access to a random permutation edge stream of a graph $G=(V,E)$, with n vertices and $m\geq 3n$ edges, produces an $O(\log^2 n)$ factor approximation to maximum matching size in G using $O(\log^2 n)$ bits of memory in a single pass over the stream with probability at least* $3/4$.

Theorem 6.1.4 improves upon the previous best known $\log^{O(1)} n$-approximation due to Kapralov et al. (2014), where the power of the logarithm was quite large (we estimate that it is at least 8). The proof is given in Section 6.9.

Let us now provide a brief overview of some of the techniques we use in this chapter.

Our techniques: approximating matching size from iid samples. As mentioned above, our approach consists of efficiently simulating a simple peeling-type algorithm, using a small amount of space and samples. This approach was previously used in Kapralov et al. (2014) to achieve a polylogarithmic approximation to maximum matching size in polylogarithmic space. As we show below, major new ideas are needed to go from a polylogarithmic approximation to a constant factor approximation. The reason for this is the fact that any matching size approximation algorithm needs to perform very deep (logarithmic depth) exploration in the graph, leading to $\omega(1)$ long chains of dependencies (i.e., recursive calls) in any such simulation. Indeed, both the work of Kapralov et al. (2014) and our result use a peeling type algorithm that performs a logarithmic number of rounds of peeling as a starting point. The work of Kapralov et al. (2014) simulated this process by random sampling, oversampling various tests by a logarithmic factor to ensure a high degree of concentration, and losing extra polylogarithmic factors in the approximation to mitigate the effect of this on sample complexity of the recursive tests. Thus, it was crucial for the analysis of Kapralov et al. (2014) that the approximation factor is much larger than the depth of exploration of the algorithm that is being simulated. In our case such an approach is provably impossible: a constant factor approximation requires access to $\Omega(\log n)$ depth neighborhoods by known lower bounds (see Kuhn et al. (2016) and Section 6.8). A method for circumventing this problem is exactly the main contribution of our work – we show how to increase sampling rates in deeper explorations of the graph, improving confidence of statistical estimation and thereby avoiding a union bound over the depth, while at the same time keeping the number of samples low. A detailed description of the algorithm and the analysis are presented in Section 6.3.

Our techniques: tight sample complexity lower bound and tight instances for peeling algorithms. As the second result in Theorem 6.1.1 shows, the sample complexity of our algorithm is essentially best possible even among algorithms that are not space constrained. The lower bound (see Section 6.8) is based on a construction of two graphs G and H on n vertices such that for a parameter k **(a)** matching size in G is smaller than matching size in H by a factor of $n^{\Omega(1)/k}$ but **(b)** there exists a bijection from vertices of G to vertices of H that preserves k-depth neighborhoods up to isomorphism. To the best of our knowledge, this construction is novel. Related constructions have been shown in the literature (e.g. cluster trees of Kuhn et al. (2016)), but these constructions would not suffice for our lower bound, since they do not provide a property as strong as **(b)** above.

Our construction proceeds in two steps. We first construct two graphs G' and H' that have identical k-level degrees (see Section 6.8.3). These two graphs are indistinguishable based on k-level degrees and their maximum matching size differs by an $n^{\Omega(1/k)}$ factor, but their neighborhoods are not isomorphic due to cycles. G' and H' have $n^{2-O(1/k)}$ edges and provide nearly tight instances for peeling algorithms that we hope may be useful in other contexts. The second step of our construction is a lifting map (see Theorem 6.8.22 in Section 6.8) that relies on high girth Cayley graphs and allows us to convert graphs with identical k-level vertex degrees to graphs with isomorphic depth-k neighborhoods without changing matching size by much. The details are provided in Section 6.8.4.

Finally, the proof of the sampling lower bound proceeds as follows. To rule out factor C approximation in $m^{1-o(1)}$ space, take a pair of constant (rather, $m^{o(1)}$) size graphs G and H such that **(a)** matching size in G is smaller than matching size in H by a factor of C and **(b)** for some large k one has that k-depth neighborhoods in G are isomorphic to k-depth neighborhoods in H. Then the actual hard input distribution consists of a large number of disjoint copies of G in the **NO** case and a large number of copies of H in the **YES** case, possibly with a small disjoint clique added in both cases to increase the number of edges appropriately. Since the vertices are assigned uniformly random labels in both cases, the only way to distinguish between the **YES** and the **NO** case is to ensure that at least k edge-samples land in one of the small copies of H or G. Since k is small, the result follows. The details can be found in Section 6.8.

Our techniques: random permutation streams.

As mentioned above, in Section 6.9 (see Theorem 6.1.4) we extend our result to a stream of edges in random order which improves upon the previous best known $\log^{O(1)} n$-approximation due to Kapralov et al. (2014), where the power of the logarithm was quite large (we estimate that it is at least 8). In order to establish Theorem 6.1.4 we exhibit a coupling between the distribution of the state of the algorithm when run on an iid stream and when run on a permutation stream. The approach is similar in spirit to that of Kapralov et al. (2014), but carrying it out for our algorithm requires several new techniques. Our approach consists of bounding the KL divergence between the distribution of the state of the algorithm in the iid and random permutation settings by induction on the level of the tests performed in the algorithm. In order to make this approach work, we develop a restricted version of triangle inequality for KL divergence that may be of independent interest (see Lemma 6.9.7 in Section 6.9).

6.1.1 Related Work

Over the past decade, matchings have been extensively studied in the context of streaming and related settings. The prior work closest to ours is by Kapralov et al. Kapralov et al. (2014). They design an algorithm that estimates the maximum matching size up to $\text{poly}(\log) n$ factors by using at most m iid edge-samples. Their algorithm requires $O(\log^2 n)$ bits of memory. As they prove, this algorithm is also applicable to the scenario in which the edges are provided as a random permutation stream. The problem of approximating matching size using $o(n)$ space has received a lot of attention in the literature, including random order streams Cormode et al. (2017); Monemizadeh et al. (2017) and worst case streams McGregor and Vorotnikova (2018); Bury et al. (2019); McGregor and Vorotnikova (2016); Chitnis et al. (2016); Esfandiari et al. (2015, 2016); Bury and Schwiegelshohn (2015); Assadi et al. (2017). The former, i.e., Monemizadeh et al. (2017); Cormode et al. (2017), are the closest to our setting since both of these works consider random streams of edges. However, the results mostly apply to bounded degree graphs due to an (at least) exponential dependence of the space on the degree. The latter consider worst case edge arrivals, but operate under a bounded arboricity assumption on the input graph. Very recently constant space algorithms for approximating some graph parameters (such as number of connected components and weight of the minimum spanning tree) from random order streams were obtained in Peng and Sohler (2018) (see also Monemizadeh et al. (2017)).

An extensive line of work focused on computing approximate maximum matchings by using $\tilde{O}(n)$ memory. The standard greedy algorithm provide a 2 approximation. It is known that no single-pass streaming algorithm (possibly randomized) can achieve better than $1-1/e$ approximation while using $\tilde{O}(n)$ memory Kapralov (2013); Goel et al. (2012). For both unweighted and weighted graphs, better results are known when a stream is randomly ordered. In this scenario, it was shown how to break the barrier of 2 approximation and obtain a $2-\epsilon$ approximation, for some small constant $\epsilon > 0$ Konrad et al. (2012); Konrad (2018); Assadi et al. (2016); Assadi and Khanna (2017); Paz and Schwartzman (2017); Assadi et al. (2019a); Assadi and Bernstein (2019); Ghaffari and Wajc (2019); Gamlath et al. (2018). If multiple passes over a stream are allowed, a line of work Feigenbaum et al. (2005); Eggert et al. (2009); Ahn and Guha (2011b) culminated in an algorithm that in $O(\text{poly}(1/\epsilon))$ passes and $\tilde{O}(n)$ memory outputs a $(1+\epsilon)$-approximate maximum matching in bipartite graphs. In the case of general graphs, $(1+\epsilon)$-approximate weighted matching can be computed in $(1/\epsilon)^{O(1/\epsilon)}$ passes and also $\tilde{O}(n)$ memory McGregor (2005); Epstein et al. (2011); Zelke (2012); Ahn and Guha (2011a,b) and Gamlath et al. (2018).

In addition to the LCA results we pointed to above, close to our work are Rubinfeld et al. (2011); Alon et al. (2012); Levi et al. (2017); Ghaffari (2016); Ghaffari and Uitto (2019) who also study the worst-case oracle behavior. In particular, Ghaffari and Uitto (2019) showed that there exists an oracle that given an arbitrary chosen vertex v outputs whether v is in some fixed maximal independent set or not by performing $d^{O(\log\log d)} \cdot \text{poly}(\log) n$ queries. When this oracle is applied to the line graph, then it reports whether a given edge is in a fixed maximal matching or not. We provide further comments on LCA related work in Section 6.6.2.

6.2 Preliminaries

Graphs In this chapter we consider unweighted undirected graphs $G = (V, E)$ where V is the vertex set and E is the edge set of the graph. We denote $|V|$ by n and $|E|$ by m. Furthermore, we use d to signify the maximum degree of G or an upper bound on the maximum degree.

Matching In this chapter, we are concerned with estimating the size of a maximum matching. Given a graph $G = (V, E)$, a *matching* $M \subseteq E$ of G is a set of pairwise disjoint edges, i.e., no two edges share a common vertex. If M is a matching of the maximum cardinality, then M is also said to be *a maximum matching*. We use $\mathrm{MM}(G)$ to refer to the cardinality of a maximum matching of G. A *fractional matching* $M_f : E \to \mathbb{R}^+$ is a function assigning weights to the edges such that the summation of weights assigned to the edges connected to each vertex is at most one, i.e., $\sum_{e \in E : v \in e} M_f(e) \le 1$, for each vertex v. The size of a fractional matching (denoted by $|M_f|$) is defined to be the summation of the weights assigned to the edges. Notice that any matching can also be seen as a fractional matching with weights in $\{0, 1\}$. It is well-known that

$$\mathrm{MM}(G) \le \max_{\text{fractional matching } M_f \text{ of } G} |M_f| \le \frac{3}{2}\mathrm{MM}(G),$$

hence,

$$\mathrm{MM}(G) = \max_{\text{fractional matching } M_f \text{ of } G} \Theta(|M_f|).$$

Therefore, to estimate the cardinality of a maximum matching, it suffice to estimates the size of a fractional maximum matching. In this work, we show that the estimate returned by our algorithm is by a constant factor smaller than the size of a fractional maximum matching, which implies that it is also by a constant factor smaller than $\mathrm{MM}(G)$.

Vertex cover We also lower-bound the estimate returned by our algorithm. To achieve that, we prove that there is a vertex cover in G of the size within a constant factor of the output of our algorithm. Given a graph $G = (V, E)$, a set $C \subseteq V$ is a *vertex cover* if each edge of the graph is incident to at least one vertex of the set C. It is folklore that the size of any vertex cover is at least the size of any matching.

Vertex neighborhood Given a graph $G = (V, E)$, we use $N(v)$ to denote the vertex neighborhood of v. That is, $N(v) = \{w : \{v, w\} \in E\}$.

6.3 Our Algorithm and Technical Overview

In this section we present our algorithm for estimating the matching size from at most m iid edge-samples. We begin by providing an algorithm that estimates the matching size while having full access to the graph (see Algorithm 14) and then, in Section 6.3.2, we show how to simulate Algorithm 14 on iid edge-samples. We point out that Algorithm 14 uses $O(m)$ memory, while our simulation uses $O(\log^2 n)$ bits of memory.

6.3.1 Offline Algorithm

First we introduce a simple peeling algorithm which we will be simulating locally. This algorithm constructs a fractional matching M and a vertex cover C which are within a constant factor of each other. As noted in Section 6.2, by duality theory this implies that both M and C are within a constant factor of the optimum. Algorithm 14 begins with both M and C being empty sets and augments them simultaneously

Algorithm 14 Offline peeling algorithm for constructing a constant factor approximate maximum fractional matching and a constant factor approximate minimum vertex cover

```
1: procedure GLOBAL-PEELING($G = (V, E), \delta, c$)        ▷ $\delta$ and $c$ are two constants that we fix later
2:     $A \leftarrow V$                                      ▷ Set of active vertices
3:     $C \leftarrow \emptyset$                              ▷ $C$ is the vertex cover that the algorithm constructs
4:     $M : E \to \mathbb{R}$
5:     $M(e) \leftarrow 0 \quad \forall e \in E$             ▷ Initially, the fractional matching for all the edges is zero
6:     for $i = 1$ to $J + 1$ do                             ▷ Represents the rounds of peeling
7:         for $e \in E \cap A \times A$ do                  ▷ Remaining edges
8:             $M(e) \leftarrow M(e) + c^{i-1}/d$            ▷ Increases weight on edges
9:         for $v \in A$ do
10:            if $M(v) \geq \delta$ then                    ▷ Recall that $M(v) = \sum_{w \in N(v)} M((v, w))$
11:                $A \leftarrow A \backslash \{v\}$         ▷ Remove the vertex
12:                $C \leftarrow C \cup \{v\}$               ▷ Add the vertex to the vertex cover
13:    return $(M, C)$
```

in rounds. When the weight of a vertex[1] v is higher than a threshold δ, the algorithm adds v to the vertex cover C and removes v along with all its incident edges. When this happens, we say that v and its edges have been *peeled off*. In the i^{th} round the weight of the matching on each edge that has not yet been peeled off is increased by c^{i-1}/d. For any $v \in V$, we denote the total weight of M adjacent to v by

$$M(v) = \sum_{w \in N(v)} M((v, w)).$$

This process continues for $J + 1$ rounds before all the edges (but not necessarily all the vertices) are peeled off from the graph. Note that at the last ($J + 1^{\text{st}}$) iteration the amount of weight assigned to each of the non-peeled-off edges is at least c^J/d and hence it suffices to set $J = \Theta(\log d)$ for all the vertices v with non-zero degree to be peeled off. Therefore, at this point C is indeed a vertex cover.

At the termination of Algorithm 14, any vertex v that has been added to C at some point must have at least δ matching adjacent to it. More precisely,

$$\sum_{e \in E} M(e) \geq \delta |C| \geq \delta \mathrm{MM}(G). \tag{6.1}$$

However, since the rate at which M is increased on each edge only increases by a multiplicative factor of c per round, the weight adjacent to any vertex cannot be much higher. Indeed, if v is peeled off in round $i + 1$, then $M(v)$ is at most δ in round i. In the intervening one round $M(v)$ could increase by at most $c\delta$, therefore $M(v)$ is at most $(c + 1)\delta$ at the point when v is peeled off. Naturally, if a vertex is peeled off in the first round, $M(v)$ is at most 1. In conclusion $M(v)$ is at most $\max((c + 1)\delta, 1)$; thus scaling down M by a factor of $\max((c + 1)\delta, 1)$ produces a (valid) fractional matching which is within a $\max(c + 1, 1/\delta)$ factor of

[1]Remark that we use the term *weight of an edge* to refer to the fractional matching assigned to that edge, i.e., the value computed by EDGE-LEVEL-TESTAlgorithm 16. Similarly, we use the term *weight of a vertex* as the summation of the weights of the edges connected to it.

C. Therefore,

$$\sum_{e\in E} M(e) \le \max((c+1)\delta, 1)\frac{3}{2}\text{MM}(G), \tag{6.2}$$

where the factor 3/2 is the result of the gap between a fractional matching and the maximum matching explained in preliminaries. Combining Eq. (6.1) and Eq. (6.2) we get a constant approximation guarantee.

Main challenges in simulating Algorithm 14 and comparison to Kapralov et al. (2014). The approach of starting with a peeling type algorithm and performing a small space simulation of that algorithm by using edge-samples of the input graph has been used in Kapralov et al. (2014). However, the peeling algorithm that was used is significantly weaker than Algorithm 14 and hence much easier to simulate, as we now explain. Specifically, the algorithm of Kapralov et al. (2014) repeatedly peels off vertices of sufficiently high degree, thereby partitioning edges of the input graph into classes, and assigns *uniform weights* on edges of every class (vertices of degree $\approx c^i$ in the residual graph are assigned weight $\approx c^{i-1}/n$). This fact that the weights are uniform simplifies simulation dramatically, at the expense of only an $O(\log n)$ approximation. Indeed, in the algorithm of Kapralov et al. (2014) the weight on an edge is equivalent to the level at which the edge disappears from the graph, and only depends on the number of neighbors that the endpoints have one level lower. In our case the weight of an edge at level i is composed of contributions from **all levels smaller than** i. As we show below, estimating such weights is much more challenging due to the possibility to errors accumulating across the chain of $\Omega(\log d)$ (possibly $\Omega(\log n)$) levels. In fact, techniques for coping with this issue, i.e., avoiding a union bound over $\log n$ levels, are a major contribution of our work.

6.3.2 IID Edge Stream Version of Algorithm 14

In this section we describe our simulation of Algorithm 14 in the regime in which an algorithm can learn about the underlying graph only by accessing iid edges from the graph.

Remark 6.3.1. *In the following we will assume that $m \ge n$ for simplicity. In the case where this is not true we can simply modify the graph by adding a new vertex v_0 to V and adding an n-star centered at v_0 to the input graph G to get G'. This increases the matching size by at most 1. Also, knowing m and V we can simply simlulate sampling an iid edge G'. Indeed, whenever we need to sample an edge of G' with probability $\frac{m}{n+m}$ we sample an iid edge of G and with probability $\frac{n}{n+m}$ we sample uniformly from $v_0 \times V$. For simplicity we will omit this subroutine from our algorithms and assume that $m \ge n$ in the input graph G.*

Remark 6.3.2. *Recall that d is an upper bound on the maximum degree of G. For the purposes of the iid sampling algorithm, we can simply set d to be n, as it can be any upper bound on the maximum degree. In this model of computation the runtime will actually not depend on d. We still carry d throughout the analysis, as it will be crucial in the LCA implementation (Section 6.6). However, the reader is encouraged to consider $d = n$ for simplicity.*

Algorithm 15 is a local version of Algorithm 14 which can be implemented by using iid edge-samples. Notice that for the sake of simplicity in both presentation of the algorithms and analysis, we referred to the iid edge oracle as stream of random edges. Instead of running the peeling algorithm on the entire

Algorithm 15 IID edge sampling algorithm approximating the maximum matching size of G, $\mathrm{MM}(G)$.

```
1:  procedure IID-PEELING($G = (V, E)$)
2:      $M' \leftarrow 0$                      ▷ The value of estimated matching
3:      $t \leftarrow 1$                       ▷ The number of iid edges that we run EDGE-LEVEL-TEST on
4:      while samples lasts do
5:          $M' \leftarrow$ SAMPLE($G, t$)
6:          $t \leftarrow 2t$
7:      return $M'$                            ▷ The estimated size of the matching that our algorithm returns
8:
9:  procedure SAMPLE($G = (V, E), t$)
10:     $M' \leftarrow 0$                      ▷ Starting fractional matching
11:     for $k = 1$ to $t$ do
12:         $e \leftarrow$ iid edge from the stream
13:         $M' \leftarrow M' +$ EDGE-LEVEL-TEST($e$)   ▷ The fractional matching assigned to this edge
14:     return $m \cdot \frac{M'}{t}$
```

Algorithm 16 Given an edge e, this algorithm returns a fractional matching-weight of e.

```
1:  procedure EDGE-LEVEL-TEST($e = (u, v)$)
2:      $w \leftarrow 1/d$                     ▷ By definition, every edge gets the weight $1/d$
3:      for $i = 1$ to $J + 1$ do              ▷ Recall that $J = \lfloor \log_c d \rfloor - 1$
4:          if LEVEL-$i$-TEST($u$) and LEVEL-$i$-TEST($v$) then   ▷ Check if both endpoints pass the $i$-th test
5:              $w \leftarrow w + c^i/d$       ▷ Increase the weight of the edge
6:          else
7:              return $w$
8:      return $w$
```

Algorithm 17 Given a vertex v that has passed the first j rounds of peeling, this algorithm returns whether or not it passes the $j + 1$st round.

```
1:  procedure LEVEL-($j+1$)-TEST($v$)
2:      $S \leftarrow 0$                       ▷ The estimate of the weight of this vertex
3:      for $k = 1$ to $c^j \cdot \frac{m}{d}$ do
4:          $e \leftarrow$ next edge in the stream     ▷ Equivalent of sampling and iid edge
5:          if $e$ is adjacent to $v$ then
6:              $w \leftarrow$ the other endpoint of $e$
7:              $i \leftarrow 0$               ▷ Represents the last level that $w$ passes
8:              while $i \le j$ and LEVEL-$i$-TEST($w$) do   ▷ LEVEL-0-TEST returns TRUEby definition
9:                  $S \leftarrow S + c^{i-j}$   ▷ The contribution that edge $e$ gives with respect to the current sampling
    rate
10:                 if $S \ge \delta$ then     ▷ If yes, then the weight of the vertex is high
11:                     return FALSE
12:                 $i \leftarrow i + 1$
13:     return TRUE
```

graph, we select a uniformly random edge e and estimate what the value of M in Algorithm 14 would be on this edge. This is done by the procedure EDGE-LEVEL-TEST(e) (Algorithm 16). The procedures IID-PEELINGand SAMPLEare then used to achieve the desired variance and will be discussed in Section 6.4; they are not necessary for now, to understand the intuition behind Algorithm 15.

Notice that in Algorithm 14 if the two endpoints of $e = \{u, v\}$ get peeled off at rounds i_1 and i_2 respectively, then we can determine the value of $M(e)$ to be $\sum_{i=1}^{\min(i_1,i_2)} c^{i-1}/d$. Therefore, when evaluating EDGE-LEVEL-TEST(e) we need only to estimate what round u and v make it to. To this end we use the procedure LEVEL-j-TEST(v) which estimates whether a vertex survives the j^{th} round of Algorithm 14. However, instead of calculating $M(v)$ exactly, by looking recursively at all neighbors of v, we only sample the neighborhood of v at some appropriate rate to get an unbiased estimator of $M(v)$.

Additionally, both EDGE-LEVEL-TEST and LEVEL-j-TEST determinate early if the output becomes clear. In EDGE-LEVEL-TEST(e), if either endpoint returns FALSEin one of the tests we may stop, since the value of $M(e)$ depends only on the endpoint that is peeled off earlier. In LEVEL-j-TEST, if the variable 'S', which is used as an accumulator, reaches the threshold δ we may stop and return FALSEeven if the designated chunk of the stream has not yet been exhausted. This speed-up will be crucial in the sample complexity analysis that we provide in Section 6.4.

Main challenges that Algorithm 16 resolves and comparison to Kapralov et al. (2014). We now outline the major ideas behind our constant factor approximation algorithm and compare them to the techniques used by the polylogarithmic approximation of Kapralov et al. (2014).

Precision of level estimates despite lack of concentration. As discussed above, the crucial feature of our algorithm is the fact that the total weight of a vertex that is assigned to level j (i.e., fails LEVEL-j-TEST) is contributed by vertices at all lower levels, in contrast to the work of Kapralov et al. (2014), where only contributions from the previous level were important. Intuitively, a test is always terminated as soon as it would need to consume more samples than expected. In fact, one can show that without this even a single call of LEVEL-j-TEST may run for $\omega(m)$ samples in expectation, in graphs similar to the hard instances for greedy, constructed in Section 6.7. As we will see in Section 6.4 the early stopping rule lends itself to extremely short and elegant analysis of sample complexity. However, the correctness of Algorithm 15 is much less straightforward. For example, note that we are approximately computing the weight of a vertex at level j by sampling, and need such estimates to be precise. The approach of Kapralov et al. (2014) to this problem was simple: $C \log n$ neighbors on the previous level were observed for a large constant factor C to ensure that all estimates concentrate well around their expectation. This lead to a blowup in sample complexity, which was reduced by imposing an aggressive pruning threshold on the contribution of vertices from the previous level, at the expense of losing another logarithmic factor in the approximation quality. The work of Kapralov et al. (2014) could afford such an approach because the approximation factor was much larger than the depth of the recursive tests (approximation factor was polylogarithmic, whereas the depth merely logarithmic). Since we are aiming for a constant factor approximation, we cannot afford this approach.

The core of our algorithm is LEVEL-j-TEST of Algorithm 17. Indeed, once the level of two vertices,

u and v, have been determined by repeated use of LEVEL-j-TEST to be $\widehat{L}(u)$ and $\widehat{L}(v)$ respectively, the connecting edge has fractional weight of exactly

$$\widehat{M}(u,v) \equiv \sum_{i=0}^{\min\{\widehat{L}(u),\widehat{L}(v)\}} c^i/d. \tag{6.3}$$

This means that the size of the matching constructed by the algorithm is $\sum_{e=(u,v)\in E}\widehat{M}(u,v)$, and our matching size estimation algorithm (Algorithm 15) simply samples enough edges to approximate the sum to a constant factor multiplicatively. Thus, in order to establish correctness of our algorithm it suffices to show that $\sum_{e=(u,v)\in E}\widehat{M}(u,v)$ is a constant factor approximation to the size of the maximum matching in G. We provide the formal analysis in Section 6.5, and give an outline here for convenience of the reader.

Our goal will be to prove that the total weight of $\widehat{M}$ on all the edges is approximately equal to the maximum matching size of G. Consider by analogy the edge weighting M of Algorithm 14. Here, M is designed to be such that the weight adjacent to any vertex (that is the sum of the weight of adjacent edges) is about a constant, with the exception of the vertices that make it to the last ($T+1^{\text{st}}$) peeling level. This is sufficient to prove correctness. $\widehat{M}$ is designed similarly, however, classifications of some vertices may be inaccurate.

First of all, if a vertex is misclassified to a peeling level much higher than where it should be, it may have a superconstant amount of weight adjacent to it. Since among n vertices some few are bound to be hugely misclassified we cannot put a meaningful upper bound on the weight adjacent to a worst case vertex; we cannot simply say that $\widehat{M}$, when normalized by a constant, is a fractional matching. Instead, we choose some large constant threshold λ and discard all vertices whose adjacent weight is higher than λ (we call these bad vertieces), then normalize the remaining matching by λ. By analyzing the concentration properties of $\widehat{M}$ in Corollary 6.5.9, we get that $\widehat{M}$ concentrates quadratically around its expectation, that is

$$\mathbb{P}\left[\widehat{M}(v) > x\right] = O\left(\mathbb{E}\left[\widehat{M}(v)\right]/x^2\right).$$

Using this we can show that discarding bad vertices will not significantly change the total weight of the graph. For details see Section 6.5.2 and particularly Corollary 6.5.10, which states that only a small fraction of the total weight is adjacent to bad vertices, in expectation:

$$\mathbb{E}\left[\widehat{M}(v)\cdot \mathbb{1}\left[\widehat{M}(v)\ge\lambda\right]\right] \le \frac{1}{4}\mathbb{E}\left[\widehat{M}(v)\right].$$

A more difficult problem to deal with is that vertices may be misclassified to lower peeling levels than where they should be. Indeed, the early stopping rule, (see Line 10 of Algorithm 17) means that vertices have a lot of chances to fail early. For example, a vertex v which *should* reside at some high ($\Theta(\log n)$) level must survive $\Theta(\log n)$ inaccurate tests to get there. One might reasonably think that some vertices are misclassified to lower levels even in the typical case; this however turns out to not be true. The key realization is that the LEVEL-$(j+1)$-TEST(v) behaves very similarly to the LEVEL-j-TEST(v) with one of the crucial differences being that LEVEL-$(j+1)$-TEST(v) samples the neighborhood of v c-times more aggressively. A multiplicative increase in sampling rate translates to a multiplicative decrease in the error probability of the test, as formalized by Lemma 6.3.3 from Section 6.5.3:

Lemma 6.3.3 (Oversampling lemma)**.** *For sufficiently small $\delta > 0$ and large enough c the following holds. Let $X = \sum_{k=1}^{K} Y_k$ be a sum of independent random variables Y_k taking values in $[0,1]$, and $\overline{X} \equiv \frac{1}{c}\sum_{i=1}^{c} X_i$ where X_i are iid copies of X. If $\mathbb{E}[X] \le \delta/3$ and $\mathbb{P}[X \ge \delta] = p$, then $\mathbb{P}\left[\overline{X} \ge \delta\right] \le p/2$.*

More formally, consider some vertex v whose peeling level *should* be about $j^* = \Theta(\log n)$. When running LEVEL-j-TEST(v) for some $j \ll j^*$ the variable S in Algorithm 17 is an unbiased estimator of the weight currently adjacent on v, and $\mathbb{E}[S] \ll \delta$. However, we have no bound on the variance of S and so it is entirely possible that with as much as constant probability S exceeds δ and the test exits in Line 10 to return false. Over a logarithmic number of levels we cannot use union bound to bound the error probability. Instead, let S' be the same variable in the subsequent run of LEVEL-$(j+1)$-TEST(v). S' can be broken into contributions from neighbors at levels below j (denoted by A below), and contributions from the neighbors at level j (denoted by B below). It can be shown that

$$S' = A + B$$

and

$$\mathbb{P}\left[S' \ge \delta\right] \le \mathbb{P}[A \ge \delta] + \mathbb{P}[B \ge 1],$$

due to the integrality of B. However, A is simply an average taken over c iid copies of S, and so we can apply the oversampling lemma, with $X = S$. (Note that S satisfies the condition of being the independent sum of bounded variables, as different iterations of the for loop in Line 3 are independent and the contribution of each to S is small.) As a result we get

$$\mathbb{P}[A \ge \delta] \ge \frac{1}{2} \cdot \mathbb{P}[S \ge \delta]$$

which ultimately allows us to get a good bound on the total error probability, summing over all j^* levels.

Sample complexity analysis of tests. Note that as Algorithm 17, in order to test whether a vertex v is peeled off at iteration at most j it suffices to sample a c^{j-1}/d fraction of the stream and run recursive tests on neighbors of v found in that random sample. It is crucial for our analysis that the tests are sample efficient, as otherwise our algorithm would not gather sufficient information to approximate matching size from only m samples (or, from a single pass over a randomly ordered permutation stream – see Section 6.9). One might hope that the sample complexity of LEVEL-j-TEST is dominated by the samples that it accesses explicitly (as opposed to the ones contributed by recursive calls), and this turns out to be the case. The proof follows rather directly by induction, essentially exactly because lower level tests, say LEVEL-i-TEST for $i < j$, by the inductive hypothesis use a c^{i-1}/d fraction of the stream, and contribute c^{i-j-1} to the counter S maintained by the algorithm (see Algorithm 17). Since the algorithm terminates as soon as the counter reaches a small constant δ, the claim essentially follows, and holds deterministically for any stream of edges – the proof is given in Section 6.4 below (see Lemma 6.4.1).

Sample complexity of approximating matching size. A question that remains to be addressed is how to actually use the level tests to approximate the maximum matching size. We employ the following natural approach: keep sampling edges of the graph using the stream of iid samples and testing the received edges (the way we process these samples is novel). In Kapralov et al. (2014), where a polylogarithmic approximation was achieved, the algorithm only needed to ascertain that at least one of the logarithmic classes (similar to the ones defined by our peeling algorithm) contains nontrivial edge mass, and could discard the others.

Since we aim to obtain a constant factor approximation, this would not be sufficient. Instead, our Line 9 maintains a counter that it updates with *estimated weights* of the edges sampled from the stream. Specifically, an edge $e = (u, v)$ is declared to be a level j edge if both u and v pass all LEVEL-i-TEST tests with $i < j$ and at least one of them fails LEVEL-j-TEST – we refer to this procedure as EDGE-LEVEL-TEST(e) (see Algorithm 16). Such an edge contributes approximately c^{j-1}/d to a counter M' that estimates matching size. It turns out to be very helpful to think of approximating matching size **as a fraction of the stream length**, i.e., have $M' \in [0,1]$ (see Line 9 for the actual test and Line 13 for the application inside Line 9). Our estimate is then the average of the weights of all sampled edges. We then need to argue that the variance of our estimate is small enough to ensure that we can get a constant multiplicative approximation without consuming more than m samples. This turns out to be a very natural variance calculation: the crucial observation is that whenever an edge e sampled from the stream is assigned weight c^{i-1}/d due to the outcomes of LEVEL-j-TEST on the endpoints (see Line 13 of Line 9), the cost of testing it (in terms of samples consumed by recursive calls) is comparable to its contribution to the estimate. This implies that at most m samples are sufficient – the details are given in the proof of Theorem 6.4.4

6.4 Sample Complexity of Algorithm 15

We begin by analyzing the sample complexity of a single LEVEL-j-TEST test. After that, in Theorem 6.4.4, we prove that this sample complexity suffices to estimate the matching size by using no more than m iid edge-samples.

6.4.1 Sample Complexity of Level Tests

Lemma 6.4.1. *For every $c \geq 1$, $\delta \leq 1/2$, and graph $G = (V, E)$, let τ_j be the maximum possible number of samples required by* LEVEL-j-TEST *defined in Algorithm 16, for $j \in [1, J+1]$. Then, with probability one we have:*

$$\tau_j \leq 2c^{j-1} \cdot \frac{m}{d}. \tag{6.4}$$

Proof. We prove this lemma by induction that Eq. (6.4) holds for each j.

Base of induction For $j = 1$ the bound Eq. (6.4) holds directly as

$$\tau_1 \leq \frac{m}{d},$$

by the definition of the algorithm.

Inductive step. Assume that Eq. (6.4) holds for j. We now show that Eq. (6.4) holds for $j+1$ as well.

Consider any vertex $v \in V$ and LEVEL-$(j+1)$-TEST(v). Let α_i be the number of recursive LEVEL-i-TEST calls invoked during a worst case run of LEVEL-$(j+1)$-TEST(v) (that is when LEVEL-$(j+1)$-TEST(v) consumes τ_{j+1} samples). Then, τ_{j+1} can be upper-bounded as

$$\tau_{j+1} \leq c^j \cdot \frac{m}{d} + \sum_{i=1}^{j} \alpha_i \tau_i.$$

Moreover, from Eq. (6.4) and our inductive hypothesis, it holds

$$\begin{aligned} \tau_{j+1} &\leq c^j \cdot \frac{m}{d} + \sum_{i=1}^{j} \alpha_i \cdot 2c^{i-1} \cdot \frac{m}{d} \\ &= c^j \cdot \frac{m}{d} \cdot \left(1 + 2\sum_{i=1}^{j} c^{i-1-j} \alpha_i\right). \end{aligned} \tag{6.5}$$

Our goal now is to upper-bound $\sum_{i \leq j} c^{i-1-j} \alpha_i$. During the execution, LEVEL-$(j+1)$-TEST maintains the variable S. Consider any time during LEVEL-$(j+1)$-TEST when a recursive call is made to LEVEL-i-TEST, for $i \geq 1$, in Line 8 of Algorithm 17. Immediately preceding this, in Line 9 of the previous iteration of the loop, the variable S would have been incremented by c^{i-1-j}. This happens α_i times for every $i \geq 1$. Consider now the state of the algorithm just before the *last* recursive call is made to a lower level test. By the time the last recursive call is made, S has been increased by $\sum_{i \leq j} c^{i-1-j} \alpha_i$ in total. However, just before the last recursive test is made, the algorithm *does not* exit to return FALSEin Line 10, meaning that $S < \delta$ at this point. Hence

$$\sum_{i \leq j} c^{i-1-j} \alpha_i \leq \delta.$$

This together with Eq. (6.5) leads to

$$\tau_{j+1} \leq c^j \cdot \frac{m}{d} \cdot (1 + 2\delta) \leq 2c^j \cdot \frac{m}{d},$$

as desired, where the last inequality follows by the assumption that $\delta \leq 1/2$. □

6.4.2 Sample Complexity of IID-PEELING

Lemma 6.4.2. *For every $c \geq 2$, $0 < \delta \leq 1/2$, and graph $G = (V, E)$, for any edge $e = (u, v) \in E$, if M_e is the output of an invocation of* EDGE-LEVEL-TEST(e)*, then with probability one this invocation used at most $4M_e \cdot m$ edge-samples.*

Proof. As before, let τ_j be the maximum possible number samples required by LEVEL-j-TEST. From Lemma 6.4.1 we have $\tau_j \leq 2c^{j-1} \cdot \frac{m}{d}$.

Let I be the last value of i, at which the algorithm EDGE-LEVEL-TEST(e) exits the while loop in Line 7 of Algorithm 16. Alternately if the algorithm exits in Line 8, let $I = J$. This means that the variable w has been incremented for all values of i from 1 to $I - 1$, making w equal to $\sum_{i=0}^{I-1} c^i / d$. On the other hand, in the worst case scenario, LEVEL-i-TEST has been called on both u and v for values of i from 1 to I. Therefore, the number of samples used by this invocation of EDGE-LEVEL-TEST(e) is at most

$$2\sum_{i=1}^{I}\tau_i \le 2\sum_{i=1}^{I}\frac{2c^{i-1}\cdot m}{d} = 4m\cdot\sum_{i=0}^{I-1}\frac{c^i}{d} = 4M_e\cdot m.$$

□

In Section 6.5, we show the following theorem.

Theorem 6.4.3. *For sufficiently small $\delta > 0$ and large enough c the following holds. For a graph $G = (V, E)$ and an edge $e \in E$, let M_e denote the value returned by* EDGE-LEVEL-TEST(e) *(Algorithm 16). Then,*

$$\sum_{e\in E}\mathbb{E}[M_e] = \Theta(MM(G)).$$

Given this, we now prove that our main algorithm outputs a constant factor approximation of the maximum matching size.

Theorem 6.4.4. *For sufficiently small $\delta > 0$ and large enough c the following holds. For a graph $G = (V, E)$,* IID-PEELING *(Algorithm 15) wit probability $4/5$ outputs a constant factor approximation of $MM(G)$ by using at most m iid edge-samples.*

We now give the proof of the main algorithmic result of the chapter:

Proof of Theorem 6.1.1 (first part): The proof of the first part of Theorem 6.1.1 now follows from Theorem 6.4.4 and the fact that IID-PEELING has recursion depth of $O(\log n)$, where each procedure in the recursion maintains $O(1)$ variables, hence requiring $O(\log n)$ bits of space. Therefore, the total memory is $O(\log^2 n)$. □

Proof of Theorem 6.4.4: We first derive an upper bound on the number of edges that SAMPLE (see Algorithm 15) needs to test in order to obtain a constant factor approximation to maximum matching size with probability at least $9/10$, and then show that the number of iid samples that the corresponding edge tests use is bounded by m, as required. We define

$$\mu \equiv \mathbb{E}_{e\sim U(E)}[M_e]$$

for convenience, where $U(E)$ is the uniform distribution on E. Now we have $\mu\cdot m = \Theta(MM(G))$ by Theorem 6.4.3. We now show that our algorithm obtains a multiplicative approximation to μ using at most m samples.

Upper bounding number of edge tests that suffice for multiplicative approximation of μ. We now analyze the number of edge-samples used by Algorithm 15. We first analyze the sample-complexity of method SAMPLE, and later of method IID-PEELING.

Let $E_t = \{e_1, \ldots, e_t\}$ be a list of t iid edge-samples taken by SAMPLE(G, t) (Line 11 of Algorithm 15), where

t is a parameter passed on Line 5 of IID-PEELING. We now show that if $t \geq 160/\mu$, then

$$\mathbb{P}\left[\left|\frac{1}{t}\sum_{i=1}^{t} M_{e_i} - \mu\right| > \mu/2\right] < 1/10, \tag{6.6}$$

where the probability is over the choice of E_t as well as over the randomness involved in sampling M_{e_t} for $i = 1, \ldots, t$.

We prove Eq. (6.6) by Chebyshev's inequality. For $e \sim U(E)$ we have that $M_e \leq \sum_{i=0}^{J} c^i/d \leq 2c^J/d \leq 2/c$, since $J = \lfloor \log_c d \rfloor - 1$.

$$\mathrm{Var}\left[M_e\right] \leq \mathbb{E}\left[M_e^2\right] \leq 2/c \cdot \mathbb{E}\left[M_e\right] = 2\mu/c.$$

We thus get by Chebyshev's inequality, using the fact that $\mathrm{Var}\left[\frac{1}{t}\sum_{i=1}^{t} M_{e_i}\right] = \frac{1}{t}\mathrm{Var}\left[M_e\right]$, that

$$\mathbb{P}\left[\left|\frac{1}{t}\sum_{i=1}^{t} M_{e_i} - \mu\right| \geq \mu/2\right] \leq \mathrm{Var}\left[M_e\right]/(t\cdot(\mu/2)^2) \leq \frac{8}{tc\mu},$$

and hence Eq. (6.6) holds for any $t \geq 160/c\mu$, as required.

Upper bounding total sample complexity of edge tests. It remains to upper bound the total number of iid edge-samples consumed by the edge tests. For an edge $e \in E$ let Z_e denote the fraction of our overall budget of m samples needed to finish the invocation EDGE-LEVEL-TEST(e) (equivalently, let $m \cdot Z_e$ be the number of samples taken). By Lemma 6.4.2 we have

$$Z_e \leq 4M_e. \tag{6.7}$$

Since $e_1, \ldots, e_t$ are uniform samples from the edges of the graph, we have

$$\mathbb{E}\left[\sum_{i=1}^{t} Z_{e_i}\right] = \sum_{i=1}^{t} \mathbb{E}\left[Z_{e_i}\right] \leq \sum_{i=1}^{t} 4\mathbb{E}\left[M_{e_i}\right] = 4\mu t.$$

Hence, t executions of EDGE-LEVEL-TEST(e_i) in expectation require at most $4\mu t \cdot m$ edges. Each of these invocations of EDGE-LEVEL-TEST is performed by SAMPLE(G, t). In addition to invoking EDGE-LEVEL-TEST, SAMPLE(G, t) samples t edges on Line 12 to obtain $e_1, \ldots, e_t$. Therefore, the total sample complexity of SAMPLE(G, t) in expectation is

$$4t\mu \cdot m + t \leq 5t\mu \cdot m.$$

In the last inequality we upper-bounded t by $t\mu \cdot m$. This upper-bound holds as $M_e \geq 1/d$ (Line 2 of Algorithm 16), so $\mu \geq 1/d \geq 1/n$ and hence $\mu \cdot m \geq 1$. Also, here we used that $m \geq n$ by Remark 6.3.1.

Upper bounding sample complexity of IID-PEELING. Finally, we bound the expected sample complexity of IID-PEELINGfrom Algorithm 15. IID-PEELINGterminates only when it runs out of samples, but we consider it to have had enough samples if it completes the call of SAMPLE with a parameter $t \geq 160/c\mu$. (In particular let t^* be the smallest power of 2 greater than $160\mu/c$; this is the value of t we aim for, as all values of t are powers of 2. $t^* \leq 320/c\mu$.) The expected number of iid samples required is at most

$$5\mu \cdot m + 10\mu \cdot m + \cdots + 5t^* \mu \cdot m \leq 10t^* \mu \cdot m \leq 3200m/c$$

By Markov's inequality we thus have that IID-PEELINGthe call of SAMPLE for $t = t^*$ within $32000m/c \leq m$ samples with probability at least $9/10$. (Let $c \geq 32000$.) Conditioned on this, the algorithm succeeds with probability at least $9/10$, therefore it succeeds with overall probability at least $4/5$, as claimed.

□

6.5 Correctness of Algorithm 15 (proof of Theorem 6.4.3)

The main result of this section is the following theorem.

Theorem 6.4.3. *For sufficiently small $\delta > 0$ and large enough c the following holds. For a graph $G = (V, E)$ and an edge $e \in E$, let M_e denote the value returned by* EDGE-LEVEL-TEST(e) *(Algorithm 16). Then,*

$$\sum_{e \in E} \mathbb{E}[M_e] = \Theta(MM(G)).$$

Overview of techniques. The main challenge in proving this claim is that the function LEVEL-j-TEST (Algorithm 16) potentially returns different outputs for the same vertex on different runs. Moreover, the outputs of two independent invocations could differ with constant probability. The propagation of such unstable estimates over $\Theta(\log d)$ (possibly $\Theta(\log n)$) peeling steps could potentially result in a significant error in the estimation of MM(G). The previous works avoid this issue by loosening the approximation factor to $O(\text{poly}(\log) n)$, which allows a union bound over all vertices of the graph. We cannot afford this. Instead, we control error propagation by showing that contributions of lower levels to higher level tests have progressively smaller variance, and hence the total error stays bounded. This is nontrivial to show, however, since none of the involved random variables concentrate particularly well around their expectation – our main tool for dealing with this is the Oversampling Lemma (Lemma 6.3.3) below.

An orthogonal source of difficulty stems from the fact that without polylogarithimic oversampling LEVEL-j-TEST's are inherently noisy, and might misclassify nontrivial fractions of vertices across $\omega(1)$ levels. Informally, to cope with this issue we charge the cost of the vertices that the LEVEL-j-TEST's misclassify to the rest of the vertices. More precisely, we show that although the algorithm might misclassify the layer that a vertex belongs to for a constant fraction of the vertices, the amount of resulting error of the edges adjacent to such vertices in the estimation of the matching is low compared to the contribution of the rest of the edges. This enables us to bound the total error cost by a small constant with respect to the size of the maximum matching.

6.5.1 Definitions and Preliminaries

In order to establish Theorem 6.4.3 we analyze the propagation of the error introduced by misclassification in a new setting, not strictly corresponding to any of our algorithms. We first consider a hypothetical set of tests. Namely, for each vertex $v \in V$, we consider executions of LEVEL-i-TEST(v) for $i = 1, 2, \ldots$

until a test fails. Then we use these outcomes to categorize all the vertices of G into levels; these levels loosely corresponding to those of Algorithm 14. We then define a set of edge weights based on these levels, $\widehat{M} : E \to \mathbb{R}$. Most of the section is then devoted to showing that this is close to a maximum fractional matching in the sense that

$$\sum_{e \in E} \mathbb{E}\left[\widehat{M}(e)\right] = \Theta(\mathrm{MM}(G)).$$

Defining a (nearly optimal) vertex cover. Our main tool in upper bounding the size of the maximum matching in G is a carefully defined (random) nested sequence of $V = \widehat{V}_0 \supseteq \widehat{V}_1 \supseteq \ldots \supseteq \widehat{V}_{J+1}$ As we show later in Section 6.5.3 (see the proof of Lemma 6.5.11), the set

$$C \equiv \left\{ v \in V \middle| \mathbb{P}\left[v \in \widehat{V}_{T+1}\right] \le 1 - \frac{1}{3c^2} \right\} \tag{6.8}$$

turns out to be a nearly optimal deterministic vertex cover in G.

Since our algorithm is essentially an approximate and randomized version of a peeling algorithm for approximating matching size (Algorithm 14), our analysis is naturally guided by a sequence of random subsets $\widehat{V}_j$ of the vertex set V. These subsets loosely correspond to the set of vertices in V that survived j rounds of the peeling process. The sets are defined by the following process performed **independently** by all vertices of G. Every $v \in V$ keeps running $\textsc{Level-}j\textsc{-Test}(v)$ (Algorithm 16) for $j = 0, 1, 2, \ldots$, while the tests return $\textsc{true}$. Let $\widehat{L}(v)$ denote the largest j such that the corresponding test returned $\textsc{true}$. We then let

$$\widehat{V}_j \equiv \left\{ v \in V : \widehat{L}(v) \ge j \right\}, \text{ for } j = 0, \ldots, J+1. \tag{6.9}$$

Note that the variables $\widehat{L}(v)$ are independent, and $V = \widehat{V}_0 \supseteq \widehat{V}_1 \supseteq \widehat{V}_2 \supseteq \ldots \supseteq \widehat{V}_{J+1}$ with probability 1. Also, $\widehat{L}(v)$ values can be defined via the sets $\widehat{V}_j$ as

$$\widehat{L}(v) \equiv \text{maximum } i \text{ such that } v \in \widehat{V}_i. \tag{6.10}$$

Recall that $\widehat{V}_0$ equals V, so for any v there is at least one i such that $v \in \widehat{V}_i$, and hence the definition Eq. (6.10) is valid.

Our proof crucially relies on a delicate analysis of the probability of a given vertex v belonging to $\widehat{V}_{j+1}$ conditioned on v belonging to $\widehat{V}_j$ for various $j = 0, \ldots, J$. To analyze such events we define, for every $v \in V$, the random variable $S_j(v)$ as follows. Let $r = c^j m/d$ and let $e_1, \ldots, e_r$ be i.i.d. uniform samples (with repetition) from the edge set E of G. Let $i_1 \le \ldots \le i_Q$, where $Q \le r$, be the subset of indices corresponding to edges in the sample that are incident on v, i.e., $e_{i_a} = (v, w_a)$ for every $a = 1, \ldots, Q$ (note that Q is a random variable). For every $a = 1, \ldots, Q$ let $L_a \sim \widehat{L}(w_a)$ be independent samples from the distribution $\widehat{L}(w_a)$ defined above. We now let

$$S_j(v) \equiv \sum_{a=1}^{Q} \sum_{i=0}^{\min\{L_a, j\}} c^{i-j}. \tag{6.11}$$

In other words, $S_j(v)$ is simply the value of the variable S during the call $\textsc{Level-}(j+1)\textsc{-Test}$ (Algorithm 16). In particular, we have

$$\widehat{V}_{j+1} = \left\{ v \in \widehat{V}_j \middle| S_j(v) < \delta \right\}. \tag{6.12}$$

Remark 6.5.1. *Note that Eq. (6.11) and Eq. (6.12), together with $\widehat{V}_0 \equiv V$ define $\widehat{V}_j$ recursively (and in a non-cyclic manner). Indeed, $\widehat{V}_{j+1}$ depends on $S_j(v)$ which depends on $\min\{L_a, j\}$ whose distribution is defined by $\widehat{V}_i, i \le j$ only.*

We also define $S_j(v)$ in a compact way and use that definition in the proofs extensively

$$S_j(v) \equiv \sum_{\substack{k=1 \\ e_k \sim U(E)}}^{c^j m/d} \mathbb{1}[v \in e_k] \sum_{i=0}^{\min(L(e_k \setminus v), j)} c^{i-j}. \tag{6.13}$$

Remark 6.5.2. *Note that here $L(w)$ is a random variable independently sampled from the distribution of $\widehat{L}(w)$, similarly to L_a in Eq. (6.11).*

Remark 6.5.3. *Here $U(E)$ denotes the uniform distribution over E. Also we denote by $v \in e$ the fact that e is adjacent to v, where v is a vertex and e is an edge. In this case we denote the other endpoint of e by $e \setminus v$. We use this notation heavily throughout the analysis.*

Defining the fractional pseudo-matching $\widehat{M}$. We now define a (random) fractional assignment $\widehat{M}$ of mass to the edges of G that our analysis will be based on: we will later show (see Lemma 6.5.7 in Section 6.5.2) that $\widehat{M}$ is close to some matching of G. For every edge $e = (u, v) \in E$ we let

$$\widehat{M}(u, v) \equiv \sum_{i=0}^{\min\{\widehat{L}(u), \widehat{L}(v)\}} c^i/d, \tag{6.14}$$

and define the size $|\widehat{M}|$ of the pseudo-matching $\widehat{M}$ to be the summation of its fractional matching mass along all the edges:

$$|\widehat{M}| \equiv \sum_{e \in E} \widehat{M}(e). \tag{6.15}$$

We note that $\widehat{M}$ is a random variable, and we show later in Section 6.5.2 (see Lemmas 6.5.7 and 6.5.11) that $\mathbb{E}[|\widehat{M}|]$ is a constant factor approximation to the size of a maximum matching in G. The following natural auxiliary definitions will be useful.

For every vertex $v \in V$, we let

$$\widehat{M}(v) \equiv \sum_{w \in N(v)} \sum_{i=0}^{\min\{\widehat{L}(v), \widehat{L}(w)\}} c^i/d = \sum_{w \in N(v)} \widehat{M}(v, w), \tag{6.16}$$

denote amount of fractional mass incident to v. Similarly, we let

$$\widehat{M}_j(v) \equiv \sum_{w \in N(v)} \sum_{i=0}^{\min\{\widehat{L}(w), j\}} c^i/d, \tag{6.17}$$

denote the amount of fractional mass contributed to v by its neighbors conditioned on $\widehat{L}(v) = j$.

In the upcoming section we will prove upper and lower bounds on $\widehat{M}$ to prove that it is within a constant factor of the matching number of G. Supposing we have this result, it is easy to deduce Theorem 6.4.3.

Proof of Theorem 6.4.3. Consider the marginal distribution of $\widehat{M}(e)$ for some edge $e=(u,v)$. $\widehat{M}(e)$ is essentially the result of running independent LEVEL-i-TEST's on u and v to determine at which level either vertex is peeled off, and then updating the edge weight in accordance with EDGE-LEVEL-TEST. Thus the marginal distribution of $\widehat{M}(e)$ is identical to that of EDGE-LEVEL-TEST(e). Therefore

$$\sum_{e\in E}\mathbb{E}[M_e]=\sum_{e\in E}\mathbb{E}\left[\widehat{M}(e)\right]=\mathbb{E}\left[|\widehat{M}|\right]=\Theta(|M|),$$

thus proving the theorem.

□

Next we state two observations that follow directly from the definitions above.

Observation 6.5.4. *For any vertex v and any j the following holds:*

(a) $\widehat{M}_j(v)$ *depends only on vertices other than v, therefore it is independent of* $\widehat{L}(v)$.

(b) $\widehat{M}(v)=\widehat{M}_{\widehat{L}(v)}(v)$.

(c) $\mathbb{E}\left[\widehat{M}_j(v)\right]=\mathbb{E}\left[S_j(v)\right]=\sum_{w\in N(v)}\sum_{i=0}^{j}\mathbb{P}\left[w\in\widehat{V}_i\right]\cdot c^i/d$.

Observation 6.5.5. *For any vertex v and any j, it holds that*

$$(c+1)\cdot\mathbb{E}\left[S_j(v)\right]\geq\mathbb{E}\left[S_{j+1}(v)\right],$$

where $S_j(v)$ is defined in Eq. (6.13).

Proof. This follows directly from the formula of $\mathbb{E}\left[S_j(v)\right]$ in Observation 6.5.4 Item (c):

$$\begin{aligned}\mathbb{E}\left[S_{j+1}(v)\right]-\mathbb{E}\left[S_j(v)\right]&=\sum_{w\in N(v)}\mathbb{P}\left[v\in\widehat{V}_{j+1}\right]\cdot c^{j+1}/d\\&\leq\sum_{w\in N(v)}\mathbb{P}\left[v\in\widehat{V}_j\right]\cdot c^{j+1}/d\\&\leq c\cdot\sum_{i=0}^{j}\sum_{w\in N(v)}\mathbb{P}\left[w\in\widehat{V}_i\right]\cdot c^i/d\\&=c\cdot\mathbb{E}\left[S_j(v)\right],\end{aligned}$$

which implies the observation. □

We will use the following well-known concentration inequalities.

Theorem 6.5.6 (Chernoff bound)**.** *Let $X_1,\ldots,X_k$ be independent random variables taking values in* $[0,a]$. *Let $X\equiv\sum_{i=1}^{k}X_i$. Then, the following inequalities hold:*

(a) For any $\delta \in [0,1]$ if $\mathbb{E}[X] \le U$ we have

$$\mathbb{P}[X \ge (1+\delta)U] \le \exp\left(-\delta^2 U/(3a)\right).$$

(b) For any $\delta > 1$ if $\mathbb{E}[X] \le U$ we have

$$\mathbb{P}[X \ge (1+\delta)U] \le \exp\left(-(\delta+1)\log(\delta+1)U/3a\right) \le \exp(-\delta U/(3a)).$$

(c) For any $\delta > 0$ if $\mathbb{E}[X] \ge U$ we have

$$\mathbb{P}[X \le (1-\delta)U] \le \exp\left(-\delta^2 U/(2a)\right).$$

6.5.2 Lower Bound: Constructing a Near-Optimal Matching from $\widehat{M}$

As the main result of this section, we prove a lower bound on $|\widehat{M}|$. Namely, we show that the mass defined by $\widehat{M}$ is at most a constant factor larger than the size of a maximum matching.

Lemma 6.5.7 (Lower-bound on $|\widehat{M}|$)**.** *Let $c \ge 20$ and $0 < \delta \le 1$. For any graph $G = (V, E)$, there exists a feasible fractional matching M such that $\mathbb{E}\left[|\widehat{M}|\right] = O(|M|)$, where $\widehat{M}$ is defined in Eq. (6.15).*

The following lemma, which is the main technical result that we use in the proof of Lemma 6.5.7, shows that if the pseudo-matching weight $\widehat{M}(v)$ is large at some level, then it is very likely that v does not pass the next vertex tests. This lemma is crucial in proving an upper-bound on the estimated size of the matching.

Lemma 6.5.8 (Concentration on $\widehat{M}(v)$)**.** *For any vertex v, any $j > 0$, and constants c and x such that $c \ge 20$ and $x \ge 100c\log c$, we have:*

$$\mathbb{P}\left[\widehat{M}(v) \ge x \wedge \widehat{L}(v) = j+1\right] \le \frac{10c^2}{x^2} \cdot \mathbb{E}\left[\widehat{M}(v) \cdot \mathbb{1}\left[\widehat{L}(v) = j\right]\right]. \tag{6.18}$$

Where $\widehat{L}(v)$ and $\widehat{M}(v)$ are defined in Eq. (6.10) and Eq. (6.16), respectively.

The full proof of Lemma 6.5.8 is deferred to Appendix D.1. Next, we show certain basic properties of $\widehat{M}$ that are derived from Lemma 6.5.8.

Corollary 6.5.9. *For any vertex v, $c \ge 20$, and $x \ge 100c\log c$, we have:*

$$\mathbb{P}\left[\widehat{M}(v) \ge x\right] \le \frac{10c^2}{x^2} \cdot \mathbb{E}\left[\widehat{M}(v)\right],$$

where $\widehat{M}(v)$ is defined in Eq. (6.16).

Proof. We simply sum Eq. (6.18) from Lemma 6.5.8 over $j = 1, \ldots, J$. The term corresponding to $\widehat{L}(v) = J+1$ is missing from the right hand side, but this only makes the inequality stronger. The term corresponding to $\widehat{L}(v) = 1$ is missing from the left hand side, but the corresponding probability is in fact 0. Indeed, if $\widehat{L}(v) = 1$, each edge adjacent to v has only $1/d$ weight on it, and there are at most d such edges. $\widehat{M}(v) \le 1 < x$. □

The following corollary enables us to bound the contribution of the high degree vertices to the fractional matching. This is a key ingredient to proving a lower bound on the estimated matching size. Namely, in the proof of Lemma 6.5.7 we ignore all the vertices such that $\widehat{M}(v) \geq \lambda$, where we think of λ being some large constant. Ignoring those vertices reduces the matching mass contained in $\widehat{M}$. The following corollary essentially bounds the matching mass lost in this process.

Corollary 6.5.10. *For any vertex v, $c \geq 20$, and $\lambda \geq 100c^2$:*

$$\mathbb{E}\left[\widehat{M}(v) \cdot \mathbb{1}\left[\widehat{M}(v) \geq \lambda\right]\right] \leq \frac{1}{4}\mathbb{E}\left[\widehat{M}(v)\right],$$

where $\widehat{M}(v)$ is defined in Eq. (6.16).

Proof. We prove this corollary by applying Corollary 6.5.9 for different values of x.

$$\begin{aligned}
\mathbb{E}\left[\widehat{M}(v) \cdot \mathbb{1}\left[\widehat{M}(v) \geq \lambda\right]\right] &\leq \sum_{i=0}^{\infty} \lambda 2^{i+1} \mathbb{P}\left[\widehat{M}(v) \geq \lambda 2^i\right] \\
&\overset{\text{by Corollary 6.5.9}}{\leq} \sum_{i=0}^{\infty} \lambda 2^{i+1} \cdot \frac{10c^2}{\lambda^2 2^{2i}} \mathbb{E}\left[\widehat{M}(v)\right] \\
&= \frac{20c^2 \mathbb{E}\left[\widehat{M}(v)\right]}{\lambda} \sum_{i=0}^{\infty} 2^{-i} \\
&\leq \frac{1}{4}\mathbb{E}\left[\widehat{M}(v)\right]
\end{aligned}$$

Since $\lambda > 80c^2$. □

We now have all necessary tools to prove Lemma 6.5.7.

Proof of Lemma 6.5.7. We prove this lemma by constructing such a matching, M. To that end, let $\lambda \equiv 100c^2$. If $\widehat{M}(v) \geq \lambda$ we say that v is a *violating* vertex. Similarly, any edge adjacent to at least one violating vertex we also call violating. We add all the non-violating edges of $\widehat{M}$ to M. Moreover, we reduce the weight of each edge in fractional matching M by the factor $1/\lambda$. Then, M is a feasible fractional matching since the summation of the weights of the edges connected to each vertex is at most one.

We now compute the expected weight of M. Recall that, in any fractional matching, the summation of the weights of the edges is half the summation of the weights of the vertices, since each edge has two endpoints. Therefore

$$\begin{aligned}
\mathbb{E}[|M|] &= \frac{1}{2} \sum_{v \in V} \mathbb{E}[|M(v)|] \\
&= \frac{1}{2\lambda} \sum_{v \in V} \left(\mathbb{E}\left[\widehat{M}(v)\right] - 2\mathbb{E}\left[\widehat{M}(v) \cdot \mathbb{1}\left[\widehat{M}(v) \geq \lambda\right]\right]\right) \\
&\overset{\text{by Corollary 6.5.10}}{\geq} \frac{1}{2\lambda} \sum_{v \in V} \frac{1}{2}\mathbb{E}\left[\widehat{M}(v)\right] \\
&\geq \frac{1}{4\lambda}\mathbb{E}\left[\widehat{M}\right].
\end{aligned}$$

Since λ is a constant by definition, this completes the proof. $\square$

6.5.3 Upper bound: Constructing a Near-Optimal Vertex Cover

Lemma 6.5.7 essentially states that the size of the pseudo-matching $\widehat{M}$ that our algorithm (implicitly) constructs does not exceed by more than a constant factor the size of a maximum matching. By the next lemma, we also provide a lower-bound on the size of $\widehat{M}$. Namely, we show that the size of $\widehat{M}$ is only by a constant factor smaller than a vertex cover of the input graph. Since from duality theory the size of a vertex cover is an upper-bound on the size of a maximum matching, the next lemma together with Lemma 6.5.7 shows that $\widehat{M}$ is a $\Theta(1)$-approximate maximum matching.

Lemma 6.5.11 (Upper-bound on $|\widehat{M}|$). *For sufficiently small $\delta > 0$ and large enough c the following holds. For any graph $G = (V, E)$, there exists a feasible vertex cover C such that $\mathbb{E}\left[|\widehat{M}|\right] = \Omega(|C|)$, where $|\widehat{M}|$ is defined in Eq. (6.15).*

In our proof, we choose C to be the set of vertices that with constant probability do not pass to the very last level, i.e., to the level $T+1$. (C corresponds to the set that we defined in Eq. (6.8).) Then, the high-level approach in our proof of Lemma 6.5.11 is to choose a vertex $v \in C$ and show that with constant probability the matching incident to v is sufficiently large.

Observe that v is added to C if the algorithm estimates that at some level the matching weight of v is at least δ. So, in light of our approach, we aim to show that if v has expected matching at least δ then it is unlikely that its *actual* matching weight is much smaller than δ in a realization. So, for every j independently we first upper-bound the probability that v gets added to C at level j if its actual matching mass by level j is much smaller than δ. To provide this type of upper-bound across all the levels simultaneously, one standard approach would be to take a union bound over all the levels. Unfortunately, the union bound would result in a loose upper-bound for our needs.

Instead we will show that over the levels the algorithm's estimates of the weight adjacent to a specific vertex becomes more and more accurate (since it takes more and more samples). This allows us to show that the likelihood of misclassifying a vertex by peeling it too early is small even across potentially $\Omega(\log n)$ levels. The matching weight that the algorithm estimates while testing level j can be decomposed into two parts:

- A_j – the weight coming from the neighbors up to level $j-1$.
- B_j – the weight coming from the neighbors that pass to level j.

Now, compared to the weight estimate while performing the test for level $j-1$, to obtain A_j the algorithm performs c times more tests and takes their average. This means that A_j is a more precise version of a test the algorithm has already done. Since this is the case, we can amortize the error coming from estimating A_j to the tests that the algorithms has already applied, and bound only the error coming from estimating B_j. This observation enables us to provide a more precise analysis than just applying a union bound.

The next lemma makes formal our discussion of relating A_j and the weight estimate the algorithm performs while testing level $j-1$. In this lemma, it is instructive to think of $\overline{X}$ as of A_j, of each X as a single instance of level $j-1$ testing (that corresponds to $A_{j-1}+B_{j-1}$), and of Y_k as the test performed on a single sampled edge.

Lemma 6.3.3 (Oversampling lemma)**.** *For sufficiently small $\delta > 0$ and large enough c the following holds. Let $X = \sum_{k=1}^{K} Y_k$ be a sum of independent random variables Y_k taking values in $[0,1]$, and $\overline{X} \equiv \frac{1}{c}\sum_{i=1}^{c} X_i$ where X_i are iid copies of X. If $\mathbb{E}[X] \le \delta/3$ and $\mathbb{P}[X \ge \delta] = p$, then $\mathbb{P}\left[\overline{X} \ge \delta\right] \le p/2$.*

This lemma formalizes the intuition that given a random variable X, taking the average of many independent copies gives a more accurate estimate of the mean then X itself, and it should therefore be less likely to exceed a threshold significantly above the mean. In the general case however, this is not true. Indeed consider a variable X such that $X > c\delta$ with some extremely small probability and $X < \delta$ the rest of the time. In this case if even one instance of the independent samples exceeds δ then the average of all the samples ($\overline{X}$) will as well; therefore the probability of exceeding δ actually increases. To be able to prove the lemma we need to use an additional characteristic of X: namely that it is the independent sum of bounded variables, and therefore it concentrates reasonably well around its expectation. This characteristic will hold for the particular random variables to which we want to apply the oversampling lemma in the proof of Lemma 6.5.11, specifically A_j.

We present the proof of this lemma in Appendix D.2. We are now ready to prove Lemma 6.5.11.

Proof of Lemma 6.5.11. Define the following set of vertices

$$C \equiv \left\{ v \in V \,\middle|\, \mathbb{P}\left[v \in \widehat{V}_{J+1}\right] \le 1 - \frac{1}{3c^2} \right\}. \tag{6.19}$$

Notice that this set is deterministic and does not depend on the outcome of $\widehat{V}_{J+1}$, only its distribution.

First we prove that C is indeed a vertex cover. Suppose toward contradiction that both $u, v \notin C$ for some edge $e = (u,v)$. Suppose u has already made it to $\widehat{V}_J$. Then in the course of deciding whether u makes it further into $\widehat{V}_{J+1}$ we take $mc^J/n \ge m/c^2$ iid edge-samples. Hence the probability of sampling the edge e at least one of those times is

$$1-\left(1-\frac{1}{m}\right)^{mc^J/d} \ge 1-\left(1-\frac{1}{m}\right)^{m/c^2} \ge \frac{1}{2c^2},$$

for $c \ge 1$. If e is sampled, then the probability that v makes it into $\widehat{V}_J$ is at least $1-1/(3c^2)$ which is strictly greater than $2/3$, since $v \notin C$. This means that with more than $1/(3c^2)$ probability u would fail at level J even if it made it that far and therefore must be in C. By contradiction C is a vertex cover.

Consider a vertex $v \in C$. Our goal is to show that the fractional matching adjacent to v is small with small probability. To that end, we upper-bound the probability that the matching adjacent to v is less than γ, for some positive constant $\gamma \ll \delta$. Indeed we will see that $\gamma \le 1/(12c^2)$ works.

Let $j^\star$ be the largest $j \in [0, J+1]$ such that $\mathbb{E}\left[S_{j^\star}(v)\right] < \gamma$. We prove that the combined probability of v failing any test up to $j^\star$ is at most $1/(6c^2)$. Before proving this, let us explain the rest of the proof, assuming we get such guarantee.

Note that the random variable $\widehat{L}(v)$ is independent from the sequence of random variables $\widehat{M}_j(v)$ and $\widehat{M}(v) = \widehat{M}_{\widehat{L}(v)}(v)$ by Item (a) and Item (b) of Observation 6.5.4. Thus the expected size of $\widehat{M}$ is sufficiently large (at least γ) conditioned on the event that $\widehat{L} > j^\star$. This does not happen in two cases. Either v fails a test at a level lower than or equal to $j^\star$, or v fails no test, but $j^\star = T+1$, so v reaches the last level but expected size of the incident matching $\widehat{M}(v)$ is too small in expectation regardless. The first event is bounded by $1/(6c^2)$ by the above guarantee; the second event is bounded by $1-1/(3c^2)$ since $v \in C$. Therefore v must reach a level greater than $j^\star$ with probability at least $1/(6c^2)$. Note that this also shows that $j^\star < T+1$, which is not clear from definition. Hence,

$$\begin{aligned}
\mathbb{E}\left[\widehat{M}(v)\right] &\geq \sum_{j=0}^{T+1} \mathbb{P}\left[\widehat{L}(v) = j\right] \mathbb{E}\left[\widehat{M}_j(v)\right] \\
&\geq \sum_{j=j^\star+1}^{T+1} \mathbb{P}\left[\widehat{L}(v) = j\right] \mathbb{E}\left[\widehat{M}_{j^\star+1}(v)\right] \\
&\geq \mathbb{P}\left[\widehat{L}(v) > j^\star\right] \cdot \gamma > \frac{\gamma}{6c^2},
\end{aligned}$$

and consequently by linearity of expectation follows $\mathbb{E}\left[|\widehat{M}|\right] \geq \frac{\gamma}{6c^2} \cdot |C|$.

In the rest of the proof we upper-bound the probability that v fails before or at the ${j^\star}^{\text{th}}$ level.

$$\begin{aligned}
\mathbb{P}\left[\widehat{L}(v) \leq j^\star\right] &= \sum_{j=0}^{j^\star} \mathbb{P}\left[\widehat{L}(v) = j\right] \\
&\leq \sum_{j=0}^{j^\star} \mathbb{P}\left[\widehat{L}(v) = j | v \in \widehat{V}_j\right] \qquad (6.20) \\
&= \sum_{j=0}^{j^\star} \mathbb{P}\left[S_j(v) \geq \delta\right]. \qquad (6.21)
\end{aligned}$$

Eq. (6.20) follows from the fact that $\widehat{L}(v) = j$ implies that $v \in \widehat{V}_j$ (while the other direction does not necessarily hold). We next rewrite $\mathbb{P}\left[S_j(v) \geq \delta\right]$. By definition Eq. (6.13), we have

$$\mathbb{P}\left[S_j(v) \geq \delta\right] = \mathbb{P}\left[\sum_{\substack{k=1 \\ e_k \sim U(E)}}^{mc^j/d} \mathbb{1}\left[v \in e_k\right] \sum_{i=0}^{\min(L(e\setminus v),j)} c^{i-j} \geq \delta\right]$$

We split the contribution to $S_j(v)$ into two parts: the weight coming from the sampled edges incident to v up to level $j-1$ (defined as the sum A_j below); and, to contribution coming from the sampled edges incident to v that passed to level j (corresponding to the sum B_j below). More precisely, for each j, the sums A_j and B_j are defined as follows

$$A_j \equiv \sum_{\substack{k=1 \\ e_k \sim U_E}}^{mc^j/d} \mathbb{1}\left[v \in e\right] \sum_{i=0}^{\min(L(e \setminus v),(j-1))} c^{i-j},$$

$$B_j \equiv \sum_{\substack{k=1 \\ e_k \sim U_E}}^{mc^j/d} \mathbb{1}\left[v \in e\right] \mathbb{1}\left[L(e \setminus v) \geq j\right],$$

where we use the notation $v \in e$ for v being an endpoint of e; in this case $e \setminus v$ denotes the other endpoint.

Observe that if $S_j(v) \geq \delta$, then either $A_j \geq \delta$, or $A_j < \delta$ and $B_j \geq \delta - A_j > 0$. If $B_j > 0$, it means that at least one edge incident to v was sampled and it passed to level j. This sample contributes 1 to B_j, and hence if $B_j > 0$ it implies $B_j \geq 1$. Then we can write

$$\mathbb{P}\left[S_j(v) \geq \delta\right] = \mathbb{P}\left[A_j + B_j \geq \delta\right] = \mathbb{P}\left[A_j \geq \delta \vee B_j \geq 1\right] \leq \alpha_j + \beta_j, \tag{6.22}$$

where we define $\alpha_j \equiv \mathbb{P}\left[A_j \geq \delta\right]$ and $\beta_j \equiv \mathbb{P}\left[B_j \geq 1\right]$. To upper-bound $\mathbb{P}\left[S_j(v) \geq \delta\right]$, we upper-bound α_j's and β_j's separately.

Upper-bounding α_j and β_j. We first upper-bound α_j by $(\alpha_{j-1} + \beta_{j-1})/2$ by applying Lemma 6.3.3. We begin by defining X, Y and $\overline{X}$ that correspond to the setup of Lemma 6.3.3. Let Y_k be the following random variable

$$Y = \mathbb{1}\left[v \in e\right] \sum_{i=0}^{\min(L(e \setminus v),(j-1))} c^{i-(j-1)},$$

where e is an edge sampled uniformly at random. Then, $A_{j-1} + B_{j-1} = \sum_{k=1}^{mc^{j-1}/n} Y_k$, where Y_k is a copy of Y. Let $X = A_{j-1} + B_{j-1}$ and $\overline{X} = A_j$. Observe that $\overline{X} = \sum_{i=1}^{c} X_i / c$. Then, for $j > 0$, it holds

$$\begin{aligned} \alpha_j &= \mathbb{P}\left[A_j \geq \delta\right] \\ &\overset{\text{by Lemma 6.3.3}}{\leq} \mathbb{P}\left[A_{j-1} + B_{j-1} \geq \delta\right]/2 \\ &\overset{\text{by Eq. (6.22)}}{\leq} (\alpha_{j-1} + \beta_{j-1})/2. \end{aligned} \tag{6.23}$$

In the case of $j = 0$, we have $\alpha_0 = 0$.

Applying Section 6.5.3 recursively, we derive

$$\alpha_j \leq \sum_{i=0}^{j-1} \frac{1}{2^{j-i}} \beta_i. \tag{6.24}$$

Now we upper-bound the sum of β_j's by using Markov's inequality: since for every $j = 0, \dots, j^\star$

$$\mathbb{E}\left[B_j\right] = \sum_{\substack{k=1 \\ e_k \sim U_E}}^{mc^j/d} \mathbb{E}\left[\mathbb{1}\left[v \in e\right] \mathbb{1}\left[L(e \setminus v) \geq j\right]\right] = \mathbb{E}\left[|N_j(v)|\right] c^j/d,$$

we get

$$\begin{aligned}
\sum_{j=0}^{j^\star} \beta_j &\leq \sum_{j=0}^{j^\star} \mathbb{E}\left[|N(v) \cap \widehat{V}_j|\right] \cdot c^j/d \\
&= \sum_{w \in N(v)} \sum_{j=0}^{j^\star} \mathbb{P}\left[w \in \widehat{V}_j\right] \cdot c^j/d \\
&= \mathbb{E}\left[S_{j^\star}(v)\right] \\
&\leq \gamma
\end{aligned} \tag{6.25}$$

Finalizing the proof. Combining the above inequalities together, we derive

$$\begin{aligned}
\mathbb{P}\left[\widehat{L}(v) \leq j^\star\right] &\overset{\text{from Eq. (6.21) and Eq. (6.22)}}{\leq} \sum_{j=0}^{j^\star} (\alpha_j + \beta_j) \\
&\overset{\text{from Eq. (6.24)}}{=} \sum_{j=0}^{j^\star} \left(\beta_j + \sum_{i=0}^{j-1} \frac{1}{2^{j-i}} \beta_i\right) \\
&\leq \sum_{j=0}^{j^\star} 2\beta_j \\
&\overset{\text{from Eq. (6.25)}}{\leq} 2\gamma \\
&\leq 1/(6c^2),
\end{aligned}$$

for $\gamma \leq 1/(12c^2)$, as desired □

6.6 LCA

Local computational algorithms (or LCA's) have been introduced in Rubinfeld et al. (2011) and have since been studied extensively, particularly in context of graph algorithms Alon et al. (2012); Mansour et al. (2012); Mansour and Vardi (2013); Even et al. (2014); Levi et al. (2017); Ghaffari and Uitto (2019). The LCA model is designed to deal with algorithms on massive data, such that both the input *and the output* are too large to store in memory. Instead we deal with both via query access. In the setting of graphs, we have access to a graph G via queries which can return the neighbors of a particular vertex. We must then construct our output (in our case a constant factor maximum matching) implicitly, such that we can answer queries about it consistently. That is, for any edge we must be able to say whether or not it is in the matching and for any vertex we must be able to say whether or not any edge adjacent to it is in the matching.

In this section we show that our approach, detailed in the previous sections, can be implemented in the LCA model as well. Specifically, we prove Theorem 6.1.2.

Theorem 6.1.2. *Let G be a graph with n vertices and maximum degree d. Then there exists a random matching M, such that $\mathbb{E}[|M|] = \Theta(MM(G))$, and an algorithm that with high probability:*

- *Given an edge e of G, the algorithm reports whether e is in M or not by using $O(d \log n)$ queries.*
- *Given a vertex v of G, the algorithm reports whether v is in M or not by using $O(d \log n)$ queries.*

Moreover, this algorithm can be executed by using $O(d \log^3 n)$ bits of memory.

Remark 6.6.1. *It can also be shown with more careful analysis that if $d = O((n/\log n)^{1/4})$ then $|M| = \Theta(MM(G))$ with high probability. A proof sketch of this claim can be found in Section 6.6.7.*

The proof of this theorem is organized as follows. First, in Section 6.6.3, we state our LCA algorithms. Then, in Section 6.6.4 we analyze the query complexity of the provided algorithms, essentially proving the two bullets of Theorem 6.1.2. In Section 6.6.5 we show that the matching fixed by our algorithms is $\Theta(1)$ approximation of MM(G). In Section 6.6.6 we discuss about the memory requirement of our approach and the implementation of consistent randomness. These conclusions are combined in Section 6.6.7 into a proof of Theorem 6.1.2.

6.6.1 Overview of Our Approach

Our main LCA algorithm simulates Algorithm 16, i.e., it simulates methods LEVEL-$(j+1)$-TEST and EDGE-LEVEL-TESTprovided in Section 6.3.2. However, instead of taking $c^j m/d$ random edge-samples from the entire graph (as done on Line 4 of LEVEL-j-TEST(v)), we first sample the number of edges D incident to a given vertex v, where D is drawn from binomial distribution $B(c^j m/d, d(v)/m)$. Then, we query D random neighbors of v. This simulation is given as Algorithm 18.

In our analysis, we tie the fractional matching weight of an edge to the query complexity. Essentially, we show that a matching weight w of an edge is computed by performing $O(w \cdot d)$ queries. As we will see, this allows us to transform a fractional to an integral matching by using only $O(d \log n)$ queries per an edge.

From fractional to integral matching. Given an edge $e = \{u, v\}$, LCA-EDGE-LEVEL-TESToutputs the fractional matching weight w_e of e. However, our goal is to implement an oracle corresponding to an integral matching. To that end, we round those fractional to $0/1$ weights as follows. First, each edge e is marked with probability $w_e/10\lambda$, for some large constant λ, specifically the constant from Corollary 6.5.10, the result of which we will be relying on heavily. Then, each edge that is the only one marked in its neighborhood is added to the matching. We show that in expectation this rounding procedure outputs a $\Theta(1)$-approximate maximum matching.

Consistency of the oracles. Our oracles are randomized. Nevertheless, they are designed in such a way that if the oracle is invoked on an edge e multiple times, each time it provides the same output. We first present our algorithms by ignoring this property, and then in Section 6.6.6 describe how to obtain these consistent outputs.

6.6.2 Related Work

Parnas and Ron Parnas and Ron (2007) initiated the question of estimating the minimum vertex cover and the maximum matching size in sublinear time. First, they propose a general reduction scheme that takes a k-round distributed algorithm and design an algorithm that for graphs of maximum degree d has $O(d^k)$ query complexity. Second, they show how to instantiate this reduction with some known distributed algorithms to estimate the maximum matching size with a constant multiplicative and ϵn additive factor with $d^{O(\log(d/\epsilon))}$ queries. An algorithm with better dependence on ϵ, but worse dependence on d, was developed by Nguyen and Onak Nguyen and Onak (2008) who showed how to obtain the same approximation result by using $2^{O(d)}/\epsilon^2$ queries. Significantly stronger query complexity, i.e., $O(d^4/\epsilon^2)$, was obtained by Yoshida et al. Yoshida et al. (2009). Both Nguyen and Onak (2008) and Yoshida et al. (2009) analyze the following randomized greedy algorithm: choose a random permutation π of the edges; visit the edges sequentially in the order as given by π; add the current edge to matching if none of its incident edge is already in the matching. We analyze this algorithm in great detail in Section 6.7. As their main result, assuming that the edges are sorted with respect to π, Yoshida et al. (2009) show that this randomized greedy algorithm in expectation requires $O(d)$ queries to output whether a given edge is in the matching fixed by π or not. When the edges are not sorted, their algorithm in expectation requires $O(d^2)$ queries to simulate the randomized greedy algorithm.

The result of Yoshida et al. (2009) was improved by Onak el al. Onak et al. (2012), who showed how to estimate the maximum matching size by using $\tilde{O}(\bar{d} \cdot \text{poly}(() 1/\epsilon)$ queries, where $\bar{d}$ is the average degree of the graph. Instead of querying randomly chosen edges, Onak et al. (2012) query randomly chosen vertices, which in turn allows them to choose a sample of $\Theta(1/\epsilon^2)$ vertices rather than a sample of $\Theta(d^2/\epsilon^2)$ edges. Then, given a vertex v, the approach of Onak et al. (2012) calls the randomized greedy algorithm on (some of) the edges incident to v. By adapting the analysis of Yoshida et al. (2009), Onak et al. (2012) are able to show that the expected vertex-query complexity of their algorithm is $O(d)$. As noted, these results estimate the maximum matching size up to a constant multiplicative and ϵn additive factor. Hence, assuming that the graph does not have isolated vertices, to turn this additive to a constant multiplicative factor it suffices to set $\epsilon = 1/d$.

To approximate the maximum matching size, the aforementioned results design oracles that given an edge e outputs whether e is in some fixed $\Theta(1)$-approximate maximum matching, e.g., a maximal matching, or not. Then, they query a small number of edges chosen randomly, and use the oracle-outputs on those edges to estimate the matching size. Concerning the query complexity, the usual strategy here is to show that running the oracle on most of the edges requires a "small" number of queries, leading to the desired query complexity in expectation. When those oracles are queried on arbitrary chosen edge, their query complexity might be significantly higher than the complexity needed to estimate the maximum matching size. We devote Section 6.7 to analyzing the randomized greedy algorithm mentioned above, and show that in some cases it requires at least $\Omega(d^{2-\epsilon})$ queries, for arbitrary small constant ϵ. This is in stark contrast with the expected query complexity of $O(d)$.

Recently, Ghaffari and Uitto (2019) showed that there exists an oracle that given an arbitrary chosen vertex v outputs whether v is in some fixed maximal independent set or not by performing $d^{O(\log\log d)} \cdot \operatorname{poly}(\log) n$ queries, which improves on the prior work obtaining $d^{O(\operatorname{poly}(\log) d)} \cdot \operatorname{poly}(\log) n$ complexity Rubinfeld et al. (2011); Alon et al. (2012); Levi et al. (2017); Ghaffari (2016). When this oracle is applied to the line graph, then it reports whether a given edge is in a fixed maximal matching or not.

6.6.3 Algorithms

Algorithm 18 is an LCA simulation of Algorithm 16. In LCA-LEVEL-$(j+1)$-TEST, that is an LCA simulation of LEVEL-$(j+1)$-TEST, we can not choose a random edge-sample from the entire graph as it is done on Line 4 of LEVEL-$(j+1)$-TEST. So, instead, we first sample from the distribution corresponding to how many of those random edge-samples will be incident to a given vertex v. In this way we obtain a number D (see Line 3 of LCA-LEVEL-$(j+1)$-TEST). Then, our algorithm samples D random edges incident to v and performs computation on them. Note that this is equivalent to the iid variant, as we would ignore all edges not adjacent to v.

Algorithm 18 This is an LCA simulation of Algorithm 16 for a graph of maximum degree d. Given an edge e, this algorithm returns a fractional matching-weight of e.

```
1: procedure LCA-EDGE-LEVEL-TEST($e = (u, v)$)
2:     for $i = 1$ to $J + 1$ do
3:         if LCA-LEVEL-$i$-TEST($u$) and LCA-LEVEL-$i$-TEST($v$) then
4:             $w \leftarrow w + c^i / n$
5:         else
6:             return $w$
7:     return $w$
```

Algorithm 19 This is an LCA simulation of Algorithm 17. Given a vertex v, this algorithm returns true if it belongs to level $j+1$ and false otherwise.

```
1: procedure LCA-LEVEL-$(j+1)$-TEST($v$)
2:     $S \leftarrow 0$
3:     Sample $D$ from binomial distribution $B\left(c^j \cdot \frac{m}{d}, d(v)/m\right)$
4:     for $k = 1$ to $D$ do
5:         Let $e = \{v, w\}$ be a random edge incident to $v$.
6:         $i \leftarrow 0$
7:         while $i \le j$ and LCA-LEVEL-$i$-TEST($w$) do    ▷ LCA-LEVEL-0-TEST($w$) returns TRUE by definition.
8:             $S \leftarrow S + c^{i-j}$
9:             if $S \ge \delta$ then
10:                return FALSE
11:            $i \leftarrow i + 1$
12:    return TRUE
```

We now build on LCA-EDGE-LEVEL-TEST and LCA-LEVEL-$(j+1)$-TEST to design an oracle that for some fixed $\Theta(1)$-approximate maximum matching returns whether a given edge is in this matching or not.

Our final LCA algorithm is ORACLE-EDGE(see Algorithm 21). Given an edge $e = \{u, v\}$, this algorithm reports whether e is in some fixed matching M or not. This matching M is fixed for all the queries. ORACLE-EDGEperforms rounding of a fractional matching as outlined above. As a helper method, it uses MATCHING-CANDIDATE that as a parameter gets an edge e, and returns 0 and 1 randomly chosen with respect to LCA-EDGE-LEVEL-TEST(e). We note that if MATCHING-CANDIDATE(e) returns 1 that it *does not* necessarily mean that e is included in M. It only means that e is a candidate for being added to M. In fact, e is added to M if e is the only matching candidate in its 1-hop neighborhood.

Algorithm 20 Given an edge e, this method rounds fractional matching mass returned by LCA-EDGE-LEVEL-TEST(e) (that is first scaled by 10λ) to an integral one.

```
1: procedure MATCHING-CANDIDATE($e$)
2:     Let $\lambda$ be the constant from Corollary 6.5.10.
3:     Let $X_e$ be 1 with probability LCA-EDGE-LEVEL-TEST($e$)/($10\lambda$), and be 0 otherwise.
4:     return $X_e$
```

Algorithm 21 An oracle that returns TRUEif a given edge e is in the matching, and returns FALSEotherwise.

```
1: procedure ORACLE-EDGE($e = (u, v)$)
2:     $X_e \leftarrow$ MATCHING-CANDIDATE($e$)
3:     if $X_e = 0$ then
4:         return FALSE
5:     for each edge $e'$ incident to $e$ do
6:         $X_{e'} \leftarrow$ MATCHING-CANDIDATE($e'$)
7:         if $X_{e'} = 1$ then
8:             return FALSE        ▷ $e$ is in the matching only if it is the only candidate in its neighborhood
9:     return TRUE
```

We also design a vertex-oracle (see Algorithm 22) that for a given vertex v reports whether an edge incident to v is in the matching M or not. This oracle iterates over all the edges incident to v, and on each invokes MATCHING-CANDIDATE. If any invocation of MATCHING-CANDIDATE(e) returns 1 it means that there is at least one matching candidate in the neighborhood of v. Recall that having two or more matching candidates in the neighborhood of v would result in none of those candidates being added to M. Hence, v is in M only if ORACLE-EDGE(e) return TRUE. Otherwise, v is not in M.

Algorithm 22 An oracle that returns TRUEif a given vertex v is in the matching, and returns FALSEotherwise.

```
1: procedure ORACLE-VERTEX($v$)
2:     for each edge $e$ incident to $v$ do
3:         $X_e \leftarrow$ MATCHING-CANDIDATE($e$)
4:         if $X_e = 1$ then
5:             return ORACLE-EDGE($e$)
6:     return FALSE        ▷ None of the edges incident to $v$ is a matching candidate.
```

6.6.4 Query Complexity

We begin by analyzing the query complexity of LCA-LEVEL-j-TEST. Statement of the next lemma is an adapted version of Lemma 6.4.1 to LCA.

Lemma 6.6.2. *For every $c \ge 2$, $0 < \delta \le 1/2$, and graph $G = (V, E)$ with the maximum degree at most d, let τ_j be the maximum query complexity of* LCA-LEVEL-j-TEST *defined in Algorithm 18, for $j \in [1, J+1]$. Then, with probability one we have:*

$$\tau_j \le 2c^{j-1}. \tag{6.26}$$

Proof. We prove this lemma by induction that Eq. (6.26) holds for each j. This proof follows the lines of Lemma 6.4.1.

Base of induction For $j = 1$ the bound Eq. (6.26) holds directly as

$$\tau_1 \le 1$$

by definition of the algorithm. Indeed, as soon as we sample a single neighbor the algorithm returns FALSE.

Inductive step. Assume that Eq. (6.26) holds for j. We now show that Eq. (6.26) holds for $j+1$ as well.

Consider any vertex $v \in V$ and LCA-LEVEL-$(j+1)$-TEST(v). Let α_i be the number of recursive LCA-LEVEL-i-TEST calls invoked. Then, τ_{j+1} can be upper-bounded as

$$\tau_{j+1} \le \alpha_0 + \sum_{i=1}^{j} \alpha_i \tau_i.$$

For $\delta < 1$, we have $\alpha_0 \le c^j$. Moreover, from Eq. (6.26) and our inductive hypothesis, it holds

$$\begin{aligned}\tau_{j+1} &\le c^j + \sum_{i=1}^{j} \alpha_i \cdot 2c^{i-1} \\ &= c^j \cdot \left(1 + 2\sum_{i=1}^{j} c^{i-1-j} \alpha_i\right).\end{aligned}$$

The rest of the proof now follows in the same way as in Lemma 6.4.1. □

The next claim is an LCA variant of Lemma 6.4.2. The proof is almost identical.

Lemma 6.6.3. *For every $c \ge 2$, $0 < \delta \le 1/2$, and graph $G = (V, E)$, for any edge $e = (u, v) \in E$, if M_e is the output of an invocation of* LCA-EDGE-LEVEL-TEST(e)*, then with probability one this invocation used at most $4M_e \cdot d$ queries.*

Proof. As before, let τ_j be the maximum possible number of samples required by LCA-LEVEL-j-TEST. From Lemma 6.6.2 we have $\tau_j \le 2c^{j-1}$.

Let I be the last value of i, at which the algorithm LCA-EDGE-LEVEL-TEST(e) exits the while loop in Line 6. Alternately if the algorithm exits in Line 7, let $I = J$. This means that the variable w has been incremented for all values of i from 0 to $I-1$, making w equal to $\sum_{i=0}^{I-1} c^i / n$. On the other hand, in the worst case scenario, LCA-LEVEL-i-TEST has been called on both u and v for values of i from 1 to I. Therefore, the number of queries used by this invocation of LCA-EDGE-LEVEL-TEST(e) is at most

$$2\sum_{i=1}^{I} \tau_i \le 2\sum_{i=1}^{I} 2c^{i-1} = 4\sum_{i=0}^{I-1} c^i = 4M_e \cdot d,$$

where we used the fact that $I \le T$. □

We are now ready to prove the query complexity of our oracles, and at the same time prove the query complexity part of Theorem 6.1.2.

Lemma 6.6.4. *For every $c \ge 2$, $0 < \delta \le 1/2$ and $G = (V, E)$ be a graph of maximum degree at most d. Then, for any edge $e \in E$ and with probability at least $1 - n^{-5}$,* ORACLE-EDGE(e) *requires $O(d \log n)$ queries.*

Proof. Each loop of ORACLE-EDGE$(e = \{u, v\})$ queries one of the edges incident to u or v. Hence, obtaining these incident edges takes $O(d)$ queries.

Let E' be the set of edges on which LCA-EDGE-LEVEL-TEST is called from MATCHING-CANDIDATE as a result of running ORACLE-EDGE(e) of Line 6. Let W be the sum of outputs of LCA-EDGE-LEVEL-TEST on these edges. First, by Lemma 6.6.3, the query complexity of all the tests performed on Line 6 is $O(W \cdot d)$. Second, following that $X_{e'}$ is obtained by rounding LCA-EDGE-LEVEL-TEST$(e')/(10\lambda)$ in MATCHING-CANDIDATE, we have that

$$\mathbb{E}\left[\sum_{e' \in E'} X_{e'}\right] = \frac{W}{10\lambda}.$$

As soon as $X_{e'} = 1$ for any $e' \in E'$, the algorithm terminates and returns FALSE on Line 8. Since $X_{e'}$s are independent random variables, by Chernoff bound (Theorem 6.5.6 Item (b)), with probability at least $1 - n^{-5}$ we have $W/(10\lambda) \le 20 \log n$. Hence, we conclude that the loop of ORACLE-EDGE requires $O(d \log n)$ queries with probability at least $1 - n^{-5}$. □

Lemma 6.6.5. *For every $c \ge 2$, $0 < \delta \le 1/2$ and $G = (V, E)$ be a graph of maximum degree at most d. Then, for any vertex $v \in V$, with high probability* ORACLE-VERTEX(v) *requires $O(d \log n)$ queries.*

Proof. Each loop of ORACLE-VERTEX(v) queries one of the edges incident to v. Hence, obtaining these incident edges takes $O(d)$ queries.

Following the same arguments as in Lemma 6.6.4 we have that with probability at least $1 - n^{-5}$ the total query complexity of all invocations of MATCHING-CANDIDATE on Line 3 is $O(d \log n)$. In addition, the algorithm invokes ORACLE-EDGE at most once. Hence, from Lemma 6.6.4, the total query complexity of ORACLE-VERTEX is $O(d \log n)$ with high probability. □

6.6.5 Approximation Guarantee

We now prove that outputs of ORACLE-EDGE correspond to a $\Theta(1)$-approximate maximum matching of G.

Lemma 6.6.6. *For sufficiently small $\delta > 0$ and large enough c the following holds. Let $G = (V, E)$ be a graph whose maximum degree is at most d. Let M be the set of edges for which* ORACLE-EDGE *outputs* TRUE*. Then, M is a matching and $\mathbb{E}[|M|] = \Theta(MM(G))$.*

Proof. We first argue that M is a matching. For an edge e, let X_e be the output of MATCHING-CANDIDATE(e). The edge e is a candidate to be a matching edge (but e will not necessarily be added to the matching M) only if $X_e = 1$. Hence, if $X_e = 0$, then the ORACLE-EDGE(e) returns FALSEon Line 4. Otherwise, Line 7 verifies whether $X_{e'} = 1$ for any e' incident to e. If for at least one such edge $X_{e'} = 1$, then e and e' are both candidates to be matching edges. However, adding both e and e' would lead to a collision and hence not a valid matching. This collision is resolved by adding neither e nor e' to M, implying that the set of edges added to M indeed forms a matching.

We now argue that $\mathbb{E}[M] = \Theta(\text{MM}(G))$. We use the fact that LCA-LEVEL-j-TEST and LCA-EDGE-LEVEL-TESTare perfect simulations of LEVEL-j-TEST and EDGE-LEVEL-TESTrespectively. Therefore, the fractional pseudo-matching defined by the return values of LCA-EDGE-LEVEL-TESTis identical to $\widehat{M}$ defined in Eq. (6.16) of Section 6.5, and obeys the same properties.

If a matching weight incident to a vertex v is at least λ, (recall that λ is the constant from Corollary 6.5.10), then we say v is *heavy*. An edge is incident to a heavy vertex if at least one of its endpoints is heavy. Hence, by Corollary 6.5.10, at most $1/2$ of the matching mass is incident to heavy vertices in expectation. Let e be an edge neither of whose endpoints are heavy. The edge e is in the matching if $X_e = 1$ and $X_{e'} = 0$ for each other edge e' incident to e. Recall that $X_{e'} = 1$ with probability LCA-EDGE-LEVEL-TEST$(e')/(10\lambda)$. Since no endpoint of e is heavy, it implies that $X_{e'} = 0$ for every e' incident to e with probability

$$\begin{aligned} & \prod_{e' \in \delta(e)} \left(1 - \frac{\text{LCA-EDGE-LEVEL-TEST}(e')}{10\lambda}\right) \\ \geq \quad & 1 - \sum_{e' \in \delta(e)} \frac{\text{LCA-EDGE-LEVEL-TEST}(e')}{10\lambda} \\ \geq \quad & 1 - \frac{2\lambda}{10\lambda} = \frac{4}{5}. \end{aligned}$$

Hence, if e is not incident to a heavy vertex, ORACLE-EDGE(e) returns TRUE and hence adds e to M with probability at least $\frac{4}{5}$LCA-EDGE-LEVEL-TEST$(e)/(10\lambda)$. Also, by Theorem 6.4.3 and our discussion about the matching mass of the edges incident to heavy vertices, we have that

$$\sum_{e = \{u,v\} : u \text{ and } v \text{ are not heavy}} \text{LCA-EDGE-LEVEL-TEST}(e) = \Theta(\text{MM}(G)).$$

This together with the fact that λ is a constant implies that $\mathbb{E}[|M|] = \Theta(\text{MM}(G))$, as desired. □

6.6.6 Memory Complexity and Consistent Oracles

In this section we discuss about the memory requirement of our LCA algorithms, and also describe how to obtain oracles that provide consistent outputs.

Each of our methods maintains $O(1)$ variables. Observe that by definition the depth of our recursive method LCA-LEVEL-$(j+1)$-TEST is $O(\log d)$. Hence, the total number of variables that our algorithms have to maintain at any point of execution is $O(\log d)$ requiring $O(\log d \cdot \log n)$ bits.

The way we described ORACLE-EDGE above, when it is invoked with an edge e two times, it could potentially provide different outputs. This is the case for two reasons: D on Line 3 and e on Line 5 of LCA-LEVEL-$(j+1)$-TEST are chosen randomly, and this choice may vary from iteration to iteration; and the random variable X_e drawn by MATCHING-CANDIDATE may be different in different invocations of MATCHING-CANDIDATE(e). It is tempting to resolve this by memorizing the output of ORACLE-EDGE(e) and the corresponding invocations of LCA-LEVEL-$(j+1)$-TEST (as we will see shortly, not all the invocations of LCA-LEVEL-$(j+1)$-TEST should be memorized). Then, when ORACLE-EDGE(e) is invoked the next time we simply output the stored value. Unfortunately, in this way our algorithm would potentially require $\Theta(Q)$ memory to execute Q oracle queries, while our goal is to implement the oracles using $O(d \cdot \text{poly}(\log) n)$ memory. To that end, instead of memorizing outputs, we will use k-wise independent hash functions.

Lemma 6.6.7. *For $k, b, N \in \mathbb{N}$, there is a hash family $\mathcal{H}$ of k-wise independent hash functions such that all $h \in \mathcal{H}$ maps $\{0,1\}^N$ to $\{0,1\}^b$. Any hash function in the family $\mathcal{H}$ can be stored using $O(k \cdot (N+b))$ bits of space.*

Lemma 6.6.8 (Nisan's PRG, Nisan (1990))**.** *For every $s, R > 0$, there exists a PRG that given a seed of $s \log R$ truly random bits can produce $\Omega(R)$ pseudo random bits such that any algorithm of space at most s requiring $O(R)$ random bits will succeed using the pseudo random bits with probability at least $2^{-\Omega(s)}$. Each bit can be extracted in $O(s \log R)$ time.*

Randomness and consistency in the algorithms. Before we explain how to apply Lemmas 6.6.7 and 6.6.8 to obtain consistency of our methods, we recall a part of our analysis and recall how LCA-LEVEL-j-TEST is used in our algorithms. Our analysis crucially depends on Corollary 6.5.10 (see Lemma 6.6.6) that provides a statement about heavy vertices, i.e., about vertices whose incident edges have the matching mass of at least λ. Heavy vertices are defined with respect to $\widehat{M}$, while $\widehat{M}$ variables are defined with respect to $\widehat{L}$ (see Eqs. (6.14) and (6.16)). Finally, $\widehat{L}(v)$ is defined/sampled by repeatedly invoking LCA-LEVEL-j-TEST(v) for $j = 0, 1, 2, \ldots$ while the tests return TRUE (see the discussion above Eq. (6.9)). LCA-EDGE-LEVEL-TEST(e) effectively samples $\widehat{L}$ for its endpoints by Line 3. Since we provide our analysis by assuming that $\widehat{L}(v)$ is defined consistently, the invocations of LCA-LEVEL-j-TEST directly from LCA-EDGE-LEVEL-TEST have to be consistent (Line 3). We call this invocation as *the top* invocation of LCA-LEVEL-j-TEST.

Based on this discussion, the top invocation of LCA-LEVEL-j-TEST(v) will use a fixed sequence $B(v)$ of random bits to execute this call (including all the recursive invocations of LCA-LEVEL-j-TEST performed therein). We emphasize that the output of the top invocation of LCA-LEVEL-j-TEST(v) does not have to be the same as the output of LCA-LEVEL-j-TEST(v) invoked recursively via some other top invocation. That is, for example, if a top level invocation LCA-LEVEL-j-TEST(v) recursively calls LCA-LEVEL-i-TEST(w)

(for some w neighbor of v and some $i < j$) then the recursive call LCA-LEVEL-i-TEST(w) continues to use $B(v)$ for its randomness.

Next, recall that by Lemma 6.6.2 LCA-LEVEL-j-TEST has query complexity $O(d)$ even at the highest level. Each query is a random edge-sample. Also, recursively via Line 7, each query requires sampling $O(d)$ times variable D on Line 3 of LCA-LEVEL-j-TEST. Hence, the total number of random bits required for the execution of LCA-LEVEL-(j)-TEST is $O(d\log^2 n)$. However, the test itself uses only $\log(d)\cdot\log(n)$ space, therefore, by Lemma 6.6.8, a seed of $O(\log^3 n)$ truly random bits suffice, that is $|B(v)| = O(\log^3 n)$. Furthermore, our runtime is only increased by a factor of $\log^3 n$ from using Nisan's PRG.

The rounding of the fractional matching by MATCHING-CANDIDATE(e) should also be consistent across queries and should never depend on where the call to MATCHING-CANDIDATE(e) came from, (unlike with LCA-LEVEL-j-TEST). To that end, each edge e should have its own random seed $B(e)$ to use in MATCHING-CANDIDATE(e). Here $|B(e)| = O(\log n)$ suffices.

Independence. We have shown that our LCA algorithm would work correctly if all seeds, $B(v)$ and $B(e)$ for $v \in V$ and $e \in E$, were truly independent. However, storing $\Omega(m)$ random seeds would be extremely inefficient. Instead we will use an k-wise independent hash family, $\mathscr{H}$, mapping $V \cup E$ to $\{0,1\}^{O(\log^3 n)}$. That is we will sample a hash function $h \in \mathscr{H}$ up front and calculate $B(v) = h(v)$ during queries. Consider an edge $e \in E$. Note that whether or not e is in the integral matching depends only on the levels of vertices in the 1-hop neighborhood of e, as well as the rounding of edges in its 1-hop neighborhood. This is $O(d)$ vertices and edges in total, and so an $O(d)$-wise independent hash family mapping $V \cup E$ to $\{0,1\}^{O(\log^3 n)}$ with uniform marginal distributions on each vertex and edge would suffice to guarantee that e is in the integral matching with exactly the same probability as in the truly independent case. This shows that $\mathbb{E}[|M|] = \mathrm{MM}(G)$ would still hold. By Lemma 6.6.7 such a family exists, and any element of it can be stored in space $O(d\log^3 n)$ space.

6.6.7 Proof of Theorem 6.1.2

Proof of Theorem 6.1.2. We are now ready to prove the main result of this section. In Section 6.6.3 we provided two oracles, ORACLE-EDGE and ORACLE-VERTEX. Lemmas 6.6.4 and 6.6.5 show that these oracles have the desired query complexity. Lemma 6.6.6 proves that ORACLE-EDGE outputs a $\Theta(1)$-approximate maximum matching. Observe that ORACLE-VERTEX is consistent with ORACLE-EDGE. That is, for any vertex v, ORACLE-VERTEX(v) output TRUE iff there is an edge incident to v for which ORACLE-EDGE(e) outputs true. This implies that the outputs of ORACLE-VERTEX also correspond to a $\Theta(1)$-approximate maximum matching. Finally, in Section 6.6.6 we discussed how these oracles can be implemented by using space $O(d\log^3 n)$. □

Proof sketch of Remark 6.6.1 . We observe that the random variable $|M|$ concentrates around its expectation since it is the sum of many bounded variables:

$$|M| = \sum_{e\in E} \mathbb{1}(e \in M).$$

Although these variables are not independent, their dependence graph has bounded degree, as observed in Section 6.6.6 under the heading **Independence**. Indeed, for any specific edge $e = (u, v) \in E$, the variable $\mathbb{1}(e \in M)$ depends on the levels of the vertices in the one-hop neighborhood of e, as well as the level and rounding of edges in the one-hope neighborhood of e. Overall, random bits that influence the rounding of e include $B(w)$ and $B(f)$ for all vertices w and edges f in the neighborhood of e. If e and e' are at least distance 3 away, they are completely independent (disregarding the dependences introduced by the hash functions we use), therefore the dependence graph of the variables $\mathbb{1}(e \in M)$ has maximum degree $d' = O(d^3)$.

Theorem 1. of Pemmaraju (2001) states that the when the independence graph of Bernoulli variables X_i has degree bounded by d', then

$$\mathbb{P}\left[\sum_i X_i \geq (1-\epsilon)\mu\right] \leq \frac{4(d'+1)}{\epsilon} \exp\left(-\mu\epsilon^2/2(d'+1)\right),$$

where $\mu = \mathbb{E}\left[\sum_i X_i\right]$. By applying this with $\epsilon = 1/2$, $\mu = O(\mathrm{MM}(G)) = O(n/d)$, $d' = O(d^3)$ and $d = O((n/\log n)^{1/4})$, we get that indeed $|M|$ is at least half of its expectation. It is also not hard to see that the same bound follows even if the variables $\mathbb{1}(e \in M)$ for edges more than distance 2 away are merely $\log n$-wise independent instead of being truly independent.

6.7 Lower Bound of $\widetilde{\Omega}(d^2)$ for Simulation of Randomized Greedy

In this section we analyze the result of Yoshida at al. Yoshida et al. (2009) for constructing a constant fraction approximate maximum matching in the LOCAL model. It was proven in Yoshida et al. (2009) that one can return whether some edge is in a maximal matching or not in time only $\Theta(d)$ when expectation is taken over both the randomness of the algorithm *and the choice of edge*. If it could be proven that this (or even a slightly weaker) bound holds for a worst case edge, the algorithm could be simply transformed into an LCA algorithm more efficient than the one we present in Section 6.6. However, we proceed to prove that this is not the case.

The algorithm we consider is a simulation of the greedy algorithm for maximal matching in LCA. Given a graph $G = (V, E)$ and a permutation π of E, a natural way to define a maximal matching of G with respect to π is as follows: process the edges of E in the ordering as given by π; when edge e is processed, add e to the matching is none of its incident edges has been already added. Motivated by this greedy approach, Yoshida et al. proposed and analyzed algorithm YYI-MAXIMAL-MATCHING(see Algorithm 23) that tests whether a given edge e is in the greedy maximal matching defined with respect to π.

Algorithm 23 Implementation of the greedy algorithm for maximal matching in LCA.

```
1: procedure YYI-MAXIMAL-MATCHING($e, \pi$)
2:     for $f \in \delta(e)$ such that $f$ precedes $e$, in order of $\pi$ do        ▷ $\delta(e)$ is the edge-neighbourhood of $e$.
3:         if YYI-MAXIMAL-MATCHING($f, \pi$) returns TRUE then
4:             return FALSE
5:     return TRUE
```

Pictorially, YYI-MAXIMAL-MATCHINGcan be viewed as a process of walking along neighboring edges (as defined via the recursive calls), and hence exploring the graph adaptively based on π. Yoshida et al. showed that the size of this exploration graph of YYI-MAXIMAL-MATCHING(e,π) is in expectation at most d, where the expectation is taken over all the starting edges e and all possible permutations π. It remained an open question whether it was necessary to take the expectation over the starting edge. If the size of the exploration tree could be shown to be $O(d)$ (or $O(d\log n)$) for even the worst case edge, this would yield an extremely efficient LCA algorithm for approximate maximum matching.

However, the main result of the section is that this is not the case:

Theorem 6.1.3. *There exists an absolute constant $b>0$ such that for every n, $d\in[5,\exp(b\sqrt{\log n})]$ and $\epsilon\in[1/d,1/2]$ there exists a graph G with n vertices and maximum degree $d+1$, and an edge e such that running* YYI-MAXIMAL-MATCHING(e,π) *from Algorithm 23 results in an exploration tree of size at least*

$$\frac{1}{8}\cdot\epsilon\cdot\left(\frac{d}{2}\right)^{2-\epsilon},$$

in expectation.

Notice that potentially $d=\exp(c\sqrt{\log n})\gg\log n$. Therefore the $O(d\log n)$ bound that we achieve in Theorem 6.1.2 is a factor $O(\frac{\exp(c\sqrt{\log n})}{\log n})$ better.

Overview of Our Approach We first construct a simple infinite tree (Definition 6.7.1): a tree in which each vertex has exactly d children with one extra special edge connected to the root. Then we prove that the number of queries made by YYI-MAXIMAL-MATCHING is bigger than d for the special edge. Afterwards, we extend the graph by merging the end points of the special edges of ϵd independent copies of these trees which creates another infinite tree. In this tree, beside the root that has ϵd children, the rest of vertices has d children (Definition 6.7.5). We also add an edge to the root and show that YYI-MAXIMAL-MATCHING uses almost d^2 queries for this edge. Infinite trees ease the computations due to the fact that each subtree is isomorphic to the main tree. In Section 6.7.2, we carefully analyze the probability and variance of YYI-MAXIMAL-MATCHING reaching high depths and show that it is unlikely that it passes depth $O(\log n)$. Later, we use that to truncate the tree after depth $O(\log n)$. This enables us to get the graph with desired bounds in Section 6.7.3. Notice that, throughout this section for the sake of simplicity, we assume that ϵd is an integer.

6.7.1 Lower Bound for Infinite Graphs

We begin by analyzing the behavior of YYI-MAXIMAL-MATCHING on infinite d-regular trees. We implement the random permutation π be assigning to each edge e a rank $r(e)$ chosen independently and uniformly at random from the interval $[0,1]$. Edges are then implicitly ordered by increasing rank. Then, Line 2 of Algorithm 23 can be thought of as being "**for** $f\in\delta(e)$ such that $r(f)<r(e)$, in increasing order of rank **do**". The behavior of the algorithm on any edge can be described by the two functions $p_e(\lambda)$ and $t_e(\lambda)$, where $\lambda\in[0,1]$. The function $p_e(\lambda)$ denotes the probability that e is in the matching, given only that $r(e)=\lambda$. Therefore, $p_e(\lambda)\in[0,1]$ and $p_e(0)=1$. The function $t_e(\lambda)$ denotes the expected size of

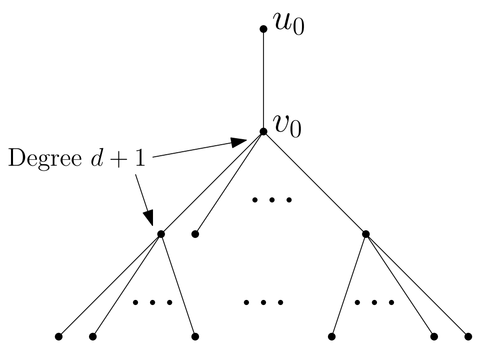


Figure 6.1 – Construction of H^d.

the exploration tree when exploring from e, given only that $r(e) = \lambda$. Therefore, we have $t_e(\lambda) \geq 1$, with equality only if $\lambda = 0$.

Definition 6.7.1 (Graph H^d, see Fig. 6.1 for illustration)**.** *For an integer $d \geq 1$, the graph H^d is defined as an infinite d-regular tree rooted in an edge $e_0 = (u_0, v_0)$. Let u_0 have no neighbor other than v_0, and let v_0 have d neighbors other than u_0. In general, let all vertices other than u_0 have $d+1$ neighbors. For an edge e,* level *of e is defined as its distance from e_0 and denoted by $\ell(e)$. In particular, $\ell(e_0) = 0$ and the level of any other edge adjacent to v_0 is 1. Every edge $e = \{u, v\} \neq e_0$ has exactly $2d$ edge-neighbors (d incident to v and d incident to u); d of these neighbors have a higher level than e, we call them e's children; $d-1$ have equal level to e, we call them e's siblings; exactly 1 has lower level than e, we call it e's parent.*

Definition 6.7.2 (Graph H_e)**.** *For every edge $e \in H^d$ let H_e be the set of edges whose unique path to e_0 goes through e (including e itself).*

In the rest of this section, we analyze the behavior of YYI-MAXIMAL-MATCHING$(e_0, \cdot)$. Specifically, we calculate the expectation and the variance of its size, and upper-bound its depth over the randomness of the ranks. It will be convenient to consider a slightly more efficient version of Algorithm 23, one in which the algorithm memorizes the results of queries across the recursive tree. That is, if in the tree of recursive calls from YYI-MAXIMAL-MATCHING(e_0, π) some edge e appears multiple times, it is counted only once in the size of the exploration tree. This memorization can lead to a great saving in the query complexity. For instance, suppose that e is the parent of f and g, which are therefore siblings. Let their ranks be $r(e) = \lambda$, $r(f) = \mu$ and $r(g) = \nu$ with $\lambda > \mu > \nu$. If the algorithm queries e, it first explores the subtree H_g. Then, if YYI-MAXIMAL-MATCHING(g, π) returns FALSE, the algorithm proceeds to querying f, only to immediately return to g and explore the same subtree of H_g, as $g \in \delta(f)$. Since the output of YYI-MAXIMAL-MATCHING(g, π) is memorized, the algorithm does not have to explore H_g again.

Thanks to memorization, in Line 2 we can ignore the parent of e, as well as all its siblings. We can ignore the parent p, since e must have been recursively queried from p, therefore $r(p) > r(e)$. As for any sibling f of e, either $r(f) > r(e)$, in which case f can be safely ignored, or $r(f) < r(e)$, in which case f must have already been queried from p and can be ignored due to memorization. This alteration to Algorithm 23 can only reduce the size of the exploration tree $T_{e_0}(\lambda)$. Since in this section we are concerned with a lower-bound on the size of a specific exploration tree, we will analyze this altered version of Algorithm 23.

Lemma 6.7.3. *Let e_0 be the root edge of the graph H^d, as defined in Definition 6.7.1. Then*

$$p_{e_0}(\lambda) = x(\lambda)^{\frac{d}{1-d}}$$
$$t_{e_0}(\lambda) = x(\lambda)^{\frac{d}{d-1}},$$

where $x(\lambda) = 1 + (d-1)\lambda$.

Proof. Let $p(\lambda) = p_{e_0}(\lambda)$ and $t(\lambda) = t_{e_0}(\lambda)$. For ease of notation we denote YYI-MAXIMAL-MATCHING(e,π) by MM(e,π).

A closed form expression for $p(\lambda)$. We first derive a recursive formula for $p(\lambda)$:

$$\begin{aligned} p(\lambda) &= \mathbb{P}\left[\text{MM}(e_0,\pi) \text{ returns } \textsc{true} \mid r(e_0) = \lambda)\right] \\ &= \mathbb{P}\left[\forall e \in \delta(e_0) : r(e) > \lambda \text{ or MM}(e,\pi) \text{ returns } \textsc{false} \text{on } H_e\right] \\ &= \prod_{e\in\delta(e_0)} \left(1 - \int_0^{\lambda} \mathbb{P}\left[\text{MM}(e,\pi) \text{ returns } \textsc{true} \text{on } H_e \mid r(e) = \mu\right] d\mu\right) \\ &= \left(1 - \int_0^{\lambda} p(\mu) d\mu\right)^d, \end{aligned} \tag{6.27}$$

since H_e is isomorphic to H^d (as per Definitions 6.7.1 and 6.7.2).

We can now solve this recursion and get a closed form formula for $p(\lambda)$. By raising both sides of Eq. (6.27) to power $1/d$, we derive that $p^{1/d}(\lambda) = 1 - \int_0^{\lambda} p(\mu) d\mu$. Differentiating both sides of this equation, we get $\frac{1}{d} \cdot p^{(1/d)-1}(\lambda) \cdot p'(\lambda) = p(\lambda)$, which implies that $p^{(1/d)-2}(\lambda) \cdot p'(\lambda) = d$ and hence

$$p(\lambda) = (C + (d-1)\lambda)^{\frac{d}{1-d}} = x^{\frac{d}{1-d}}(\lambda),$$

as claimed, where $C = 1$ due to the initial condition of $p(0) = 1$.

A closed form expression for $t(\lambda)$. We now derive a recursive formula for $t(\lambda)$. Let T_e be a random variable denoting the size of the exploration tree when running Algorithm 23 from e in H_e. Note that T_e is distributed identically for all e, and $\mathbb{E}(T_e | r(e) = \lambda) = t(\lambda)$, since H_e is always isomorphic to H^d. Let $T = T_{e_0}$. Furthermore, let I_e denote the indicator variable of e being explored through the recursive calls, when the algorithm is originally initiated from e_0. Then T satisfies

$$T = 1 + \sum_{e\in\delta(e_0)} I_e \cdot T_e,$$

and hence,

$$t(\lambda) = \mathbb{E}\left[T | r(e_0) = \lambda\right] = 1 + \sum_{e\in\delta(e_0)} \mathbb{E}\left[I_e \cdot T_e | r(e_0) = \lambda\right]$$

The expression $\mathbb{E}[I_e \cdot T_e | r(e_0) = \lambda]$ can be nicely taken apart if we condition on the rank of e, as long as it is less than λ. (If it is more than λ, $I_e = 0$.) Indeed, note that I_e depends only on the ranks and outcomes of the siblings of e, as well as the rank of e itself. Meanwhile, T_e depends only on the ranks in H_e. The only intersection between these is the rank of e, meaning that I_e and T_e are independent conditioned on $r(e)$. These observations lead to

$$t(\lambda) = 1 + \sum_{e \in \delta(e_0)} \int_0^{\lambda} \mathbb{P}\left[I_e = 1 | r(e_0) = \lambda, r(e) = \mu\right] \cdot \mathbb{E}\left[T_e | r(e) = \mu\right] d\mu.$$

The expected size of the exploration tree from e is simply $t(\mu)$, again since H_e is isomorphic to H^d. Consider the probability that e is explored at all. This happens exactly when for any sibling of e, f, either $r(f) > r(e)$ or $\mathrm{MM}(f,\pi)$ returns FALSE. This condition is very similar to the condition for $\mathrm{MM}(e_0,\pi)$ returning TRUE(recall Eq. (6.27)). The only difference is that the condition must hold only for $\delta(e_0)\backslash e$ as opposed to $\delta(e_0)$. Hence we have

$$\mathbb{P}\left[I_0 = 1 | r(e_0) = \lambda, r(e) = \mu\right] = \left(1 - \int_0^{\mu} p(v)\,dv\right)^{d-1} = p^{\frac{d}{d-1}}(\mu) = x^{-1}(\mu).$$

In particular, the rhs does not depend on λ. Therefore,

$$t(\lambda) = 1 + d\int_0^{\lambda} x^{-1}(\mu)\, t(\mu)\, d\mu.$$

We can now solve this recursion and get a closed form formula for $t(\lambda)$.

$$\begin{aligned}
&t(\lambda) = 1 + d\int_0^{\lambda} x^{-1}(\mu)\, t(\mu)\, d\mu \\
&t'(\lambda) = d x^{-1}(\lambda)\, t(\lambda) \\
&\frac{t'(\lambda)}{t(\lambda)} = \frac{d}{x(\lambda)} \\
&\log(t(\lambda)) = \frac{d}{d-1} \cdot \log(x(\lambda)) + C_1 \\
&t(\lambda) = C_2 \cdot x^{\frac{d}{d-1}}(\lambda) = x^{\frac{d}{d-1}}(\lambda),
\end{aligned}$$

as claimed, due to the initial condition of $t(0) = 1$. □

Corollary 6.7.4. *Let e_0 be the root edge of the graph H^d, as defined in Definition 6.7.1. Then,*

$$\mathbb{E}\left[T_{e_0}\right] \le \mathbb{E}_{\lambda}\left[t_{e_0}(\lambda)\right] \le t_{e_0}(1) = x^{\frac{d}{d-1}}(1) = d^{\frac{d}{d-1}} \le 2d,$$

for $d \ge 5$.

We next construct an infinite graph with an edge whose expected exploration tree size has nearly quadratic dependence on d.

Definition 6.7.5 (Graph $H^{d,\epsilon}$, see Fig. 6.2 for illustration)**.** *Fix some small positive number ϵ. Take ϵd disjoint copies of H^d, call them $H^{(1)}, H^{(2)}, \ldots, H^{(\epsilon d)}$. Let the root edge of $H^{(i)}$ be $e^{(i)} = (u^{(i)}, v^{(i)})$. We merge*

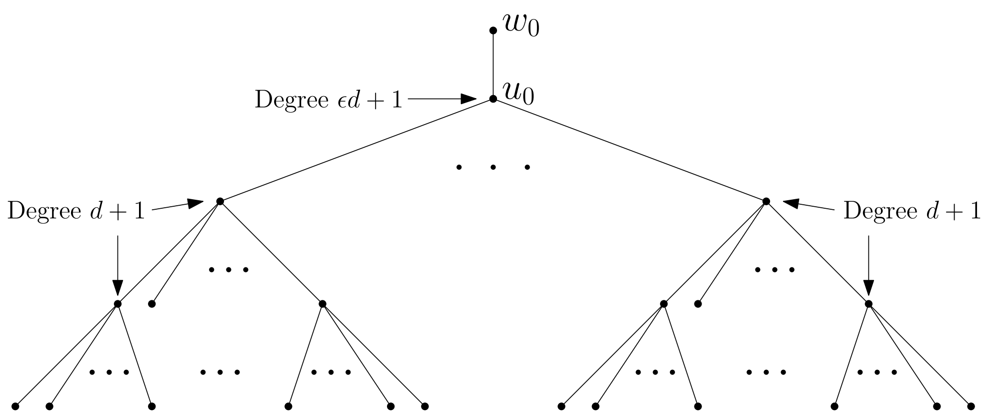


Figure 6.2 – Construction of $H^{d,\epsilon}$.

$u^{(1)}, u^{(2)}, \ldots, u^{(\epsilon d)}$ into a supernode u_0, and add a new node w_0 along with an edge $e_0 = (w_0, u_0)$. This creates the infinite graph $H^{d,\epsilon}$; we call the edge e_0 the root edge.

We will now show that querying e_0 in $H^{d,\epsilon}$ with Algorithm 23 produces an exploration tree of nearly quadratic size in expectation.

Lemma 6.7.6. *For every $\epsilon \in (0,1)$, every integer $d \geq 5$, in the graph $H^{d,\epsilon}$ (see Definition 6.7.5), where e_0 is the root edge, it holds*

$$t_{e_0}(\lambda) \geq \epsilon \cdot x^{2-\epsilon}(\lambda)/2.$$

Proof. Recall the definitions of I_e and T_e from the proof of Lemma 6.7.3: Let $I_{e^{(i)}}$ be the indicator variable of $e^{(i)}$ being explored when Algorithm 23 is called from e_0; let $T_{e^{(i)}}$ be the size of the exploration tree from $e^{(i)}$ in $H^{(i)}$. For simplicity let $T_i = T_{e^{(i)}}$ and $I_i = I_{e^{(i)}}$. We can derive a formula for $t_{e_0}(\lambda)$ similarly to Lemma 6.7.3:

$$t_{e_0}(\lambda) = 1 + \sum_{i=1}^{\epsilon d} \int_0^{\lambda} \mathbb{P}\left[I_i | r(e^{(i)}) = \mu\right] \cdot \mathbb{E}\left[T_i | r(e^{(i)}) = \mu\right] d\mu.$$

$\mathbb{E}\left[T_i | r(e^{(i)}) = \mu\right]$ is simply $t(\mu) = x^{\frac{d}{d-1}}(\mu)$ by Lemma 6.7.3, since the subtree of $e^{(i)}$ in $H^{d,\epsilon}$ is isomorphic to H^d. Similarly to Lemma 6.7.3, the probability that $e^{(i)}$ is explored is the probability that for any sibling $e^{(j)}$ of $e^{(i)}$ we have that either $r(e^{(j)}) > r(e^{(i)})$ or $\mathrm{MM}(e^{(j)}, \pi)$ returns FALSE. $e^{(i)}$ has $\epsilon d - 1$ neighbors, each of whose subtree is isomorphic to H^d, hence we have

$$\begin{aligned}
\mathbb{P}\left[I_i | r(e^{(i)}) = \mu\right] &= \prod_{i=1}^{\epsilon d} \left(1 - \int_0^{\mu} p_{e^{(i)}}(v)\, dv\right) \\
&= \left(1 - \int_0^{\mu} p(v)\, dv\right)^{\epsilon d - 1} \\
&= p^{\frac{\epsilon d - 1}{d}}(\mu) \\
&= x^{\frac{\epsilon d - 1}{1-d}}(\mu).
\end{aligned}$$

Therefore,

$$\begin{aligned} t_{e_0}(\lambda) &= 1 + \epsilon d \int_0^\lambda x^{\frac{\epsilon d - 1}{1-d}} \cdot x^{\frac{d}{d-1}}(\mu)\, d\mu \\ &\geq 1 + \epsilon d \int_0^\lambda x^{1-\epsilon}(\mu)\, d\mu \\ &= 1 + \frac{\epsilon d}{(2-\epsilon)(d-1)} \cdot \left(x^{2-\epsilon}(\lambda) - 1\right) \\ &\geq \epsilon \cdot x^{2-\epsilon}(\lambda)/2, \end{aligned}$$

as claimed. □

Corollary 6.7.7. *For every $\epsilon \in (0,1)$, every integer $d \geq 5$, for the root e_0 of the graph $H^{d,\epsilon}$ (see Definition 6.7.5) we have:*

$$\mathbb{E}\left[T_{e_0}\right] = \mathbb{E}_\lambda\left[t_{e_0}(\lambda)\right] \geq \frac{1}{2} \cdot \epsilon \cdot t_{e_0}(1/2) \geq \epsilon \cdot \frac{1}{4} \cdot \left(\frac{d}{2}\right)^{2-\epsilon}.$$

Therefore this is a construction in which an edge has expected exploration tree size which is nearly quadratic in d. However, the graph $H^{d,\epsilon}$ is infinite, so we proceed to finding a finite graph that has an edge whose exploration tree is also near quadratic. To obtain such a finite graph, we perform the following natural modification of $H^{d,\epsilon}$: we cut off the $H^{d,\epsilon}$ graphs at some depth ℓ, that is we discard all edges e such that $\ell(e) > \ell$, along with vertices that become isolated as a result. (Recall that $\ell(e)$ is the level of the edge e.) We will prove that the exploration tree usually does not explore edges beyond a depth of $O(\log d)$, and so intuitively we should be able to cut the graph at that depth. We make this precise in the next section.

6.7.2 Depth and Variance Analysis for Infinite Graphs

We will now study the depth of the exploration tree, that is the highest level of any edge in it. Let the depth of the exploration tree from e_0 be D.

Lemma 6.7.8. *For every $\ell \geq 1$, if D is the depth of the exploration tree in the graph H^d (see Definition 6.7.1), one has*

$$\mathbb{P}[D \geq \ell] \leq 2^{1-\ell} d^2.$$

Proof. Consider the weight of the exploration tree, defined as follows: For each edge e in the exploration tree we count it with weight $2^{\ell(e)}$. Let the expected weighted size of an exploration tree from e_0, conditioned on $e_0 = \lambda$ be $T_2(\lambda)$. We can derive a recursive formula for $t_2(\lambda)$ with the same technique as was used to derive a recursive formula for $t(\lambda)$ in Lemma 6.7.3, the only difference is the additional factor of 2 on the right hand side (we do not repeat the almost identical proof here). We get

$$t_2(\lambda) = 1 + d \int_0^\mu x^{-1}(\mu)(2t_2(\mu))\, d\mu.$$

We can then solve this recursion in a similar manner to the one in Lemma 6.7.3 (again, the only difference is the extra factor of 2 in the exponent):

$$\begin{aligned}
t_2(\lambda) &= 1 + 2d\int_0^\lambda x^{-1}(\mu)\, t_2(\mu)\, d\mu \\
t_2'(\lambda) &= 2d x^{-1}(\lambda)\, t_2(\lambda) \\
\frac{t_2'(\lambda)}{t_2(\lambda)} &= \frac{2d}{x(\lambda)} \\
\log(t_2(\lambda)) &= \frac{2d}{d-1}\cdot \log(x(\lambda)) + C_1 \\
t_2(\lambda) &= C_2 \cdot x^{\frac{2d}{d-1}}(\lambda) = x^{\frac{2d}{d-1}}(\lambda),
\end{aligned}$$

due to the initial condition of $t_2(0) = 1$. Specifically $t_2(\lambda) < t_2(1) = x^{\frac{2d}{d-1}} \le 2d^2$ for $d \ge 5$.

We can now complete the proof of the lemma. Note that if the depth D of the exploration tree in H^d satisfies $D \ge \ell$ then the tree contains at least an edge of level ℓ, the weight of the tree must be at least 2^ℓ. Therefore:

$$\begin{aligned}
&2d^2 \ge t_2(1) \ge \mathbb{E}\left[2^D\right] \ge 2^\ell \cdot \mathbb{P}[D \ge \ell] \\
&\mathbb{P}[D \ge \ell] \le 2^{1-\ell} d^2,
\end{aligned}$$

as claimed.

□

Corollary 6.7.9. *In the graph $H^{d,\epsilon}$ (see Definition 6.7.5),*

$$\mathbb{P}[D \ge \ell + 1] \le 2^{1-\ell}\epsilon d^3.$$

Proof. Indeed, in order for T_{e_0} in $H^{d,\epsilon}$ to have depth $\ell + 1$, $T_{e^{(i)}}$ in $H^{(i)}$ must have depth at least ℓ for at least on of the i's. We know from Lemma 6.7.8 that the probability of this is at most $2^{1-\ell}d^2$ as $H^{(i)}$ is isomorphic to H^d. By simple union bound over all values of i we get that $\mathbb{P}[D \ge \ell + 1] \le 2^{1-\ell}\epsilon d^3$. □

We have shown in Corollary 6.7.9 that the depth of the exploration tree from the root of H^d or $H^{d,\epsilon}$ doesn't exceed $O(\log d)$ with high probability. It would be intuitive to truncate these graphs at depth $\Theta(\log d)$ to get a finite example for a graph with an edge e_0 from which exploration takes quadratic time. However, Corollary 6.7.9 does not by itself rule out the possibility that T_{e_0} concentrated extremely badly around its expectation, i.e. the exploration tree is extremely large with very small probability, and most of the work is done beyond the $O(\log d)$ first levels of the tree. We rule this out by exhibiting a $d^{O(1)}$ upper bound on the second moment $\mathbb{E}\left[T_{e_0}^2\right]$ in H^d and $H^{d,\epsilon}$. Afterwards, we show that combining these second moment bounds with Corollary 6.7.9 and Lemma 6.7.6 shows that truncating $H^{d,\epsilon}$ indeed yields a hard instance. Our variance bound is given by:

Lemma 6.7.10. *In the graph H^d (see Definition 6.7.1) for the root edge e_0 one has $\mathbb{E}\left[T_{e_0}^2\right] \le 10d^5$.*

Corollary 6.7.11. *In the graph $H^{d,\epsilon}$ (see Definition 6.7.5) for the root edge e_0 one has $\mathbb{E}\left[T_{e_0}^2\right] \le 11\epsilon d^6$.*

The proofs follow along the lines of previous analysis, but are more technical and hence both are deferred to Appendix D.3.

6.7.3 The Lower Bound Instance (Truncated $H^{d,\epsilon}$)

We are now ready to truncate our graph $H^{d,\epsilon}$:

Definition 6.7.12 (Truncated graph $H_\ell^{d,\epsilon}$)**.** *Define $H_\ell^{d,\epsilon}$ as the graph $H^{d,\epsilon}$ reduced to only edges of level at most ℓ.*

Note that $H_\ell^{d,\epsilon}$ is a finite graph, specifically with approximately $n = d^\ell$ vertices. We now show that for some $\ell = O(\log d)$ the size of the exploration tree from the root edge e_0 in the graph $H_\ell^{d,\epsilon}$ is essentially the same as in the graph $H^{d,\epsilon}$ (see Corollary 6.7.7 below). This yields our final lower bound instance.

Lemma 6.7.13. *For every integer $d \geq 5$, every $\epsilon \in [1/d, 1/2]$ and $\ell \geq 7\log_2(3d)+1$, in graph $H_\ell^{d,\epsilon}$ one has $\mathbb{E}[T_\ell] \geq \frac{1}{8}\cdot\epsilon\cdot\left(\frac{d}{2}\right)^{2-\epsilon}$, where T_ℓ is the size of the exploration tree started at the root edge e_0 in $H_\ell^{d,\epsilon}$.*

Proof. Consider a graph $H^{d,\epsilon}$ and its truncated version $H_\ell^{d,\epsilon}$ for $\ell \geq 7\log_2(3d)$. Let $T = T_{e_0}$ in $H^{d,\epsilon}$ and $T_\ell = T_{e_0}$ in $H_\ell^{d,\epsilon}$. We consider the ranks of edges in $H^{d,\epsilon}$ to be identical to the ranks of the corresponding edges in $H^{d,\epsilon}$ (this is in a way a coupling of the ranks of the two graphs). Consider the events $\mathscr{S}$ and $\mathscr{D}$ referring to a shallow or a deep exploration tree respectively. Specifically, $\mathscr{S}$ refers to the event that the exploration tree in $H^{d,\epsilon}$ does not exceed a depth of ℓ; $\mathscr{D}$ is the compliment of $\mathscr{S}$. Note that given $\mathscr{S}$, $T = T_\ell$ thanks to the coupling of the ranks.

We know from Corollary 6.7.7 that

$$\epsilon\cdot\frac{1}{4}\cdot\left(\frac{d}{2}\right)^{2-\epsilon} \leq \mathbb{E}[T] = \mathbb{E}[T\cdot\mathbb{1}(\mathscr{S})] + \mathbb{E}[T\cdot\mathbb{1}(\mathscr{D})] = \mathbb{E}[T_\ell\cdot\mathbb{1}(\mathscr{S})] + \mathbb{E}[T\cdot\mathbb{1}(\mathscr{D})].$$

Thus, in order to prove that $\mathbb{E}[T_\ell\cdot\mathbb{1}(\mathscr{S})] \geq \frac{1}{8}\cdot\epsilon\cdot\left(\frac{d}{2}\right)^{2-\epsilon}$ it suffices to show that

$$\mathbb{E}[T\cdot\mathbb{1}(\mathscr{D})] \leq \frac{1}{8}\cdot\epsilon\cdot\left(\frac{d}{2}\right)^{2-\epsilon}. \tag{6.28}$$

Let $p = \mathbb{P}(\mathscr{D})$; by Corollary 6.7.9 and our choice of $\ell \geq 7\log_2(3d)$ we know that this is at most $2\epsilon\cdot 3^{-7}\cdot d^{-4} \leq (11\cdot 64)^{-1}\cdot d^{-4}$, for $\epsilon \leq 1$. Let us upper bound $\mathbb{E}[T\cdot\mathbb{1}(\mathscr{D})] = p\cdot\mathbb{E}[T|\mathscr{D}]$: We know from Corollary 6.7.11 that

$$11\epsilon d^6 \geq \mathbb{E}\left[T^2\right] \geq \mathbb{E}\left[\mathbb{1}(\mathscr{D})\cdot T^2\right] = p\cdot\mathbb{E}\left[T^2|\mathscr{D}\right] \geq p\cdot\mathbb{E}[T|\mathscr{D}]^2 = \frac{(p\cdot\mathbb{E}[T|\mathscr{D}])^2}{p},$$

and thus, rearranging, we get

$$p\cdot\mathbb{E}[T|\mathscr{D}] \leq \sqrt{11\epsilon d^6 p} \leq \frac{1}{8}\cdot\epsilon\cdot\left(\frac{d}{2}\right)^{2-\epsilon},$$

since $d \geq 5$ and $p \leq (11\cdot 64)^{-1}\cdot d^{-4}$ by assumption of the lemma and setting of parameters.

□

We can now prove the main theorem of this section.

Proof of Theorem 6.1.3: Indeed, let $G = H_\ell^{d,\epsilon}$ with e being the root edge and $\ell = \Theta(\log_d n)$. Then the theorem holds by Lemma 6.7.13 as long as $d \le \exp(b\sqrt{\log n})$ for a sufficiently small absolute constant b. □

6.8 Lower-bound on the Number of Sampled Edges

6.8.1 Overview

In this section we prove that our algorithm is nearly optimal with respect to sample complexity, even disregarding the constraint on space. That is, we show that it is impossible to obtain a constant-factor approximation of the maximum matching size with polynomially fewer than n^2 samples.

Theorem 6.8.1. *There exists a graph G consisting of $\Theta(n^2)$ edges such that no algorithm can compute a constant-factor approximation of $MM(G)$ with probability more than $6/10$ while using iid edge stream of length $n^{2-\epsilon}$. More generally, for every constant C, every m between $n^{1+o(1)}$ and $\Omega(n^2)$ it is information theoretically impossible to compute a C-approximation to maximum matching size in a graph with high constant probability using fewer than $m^{1-o(1)}$ iid samples from the edge set of G, even if the algorithm is not space bounded.*

Theorem 6.8.1 follows directly from the following result.

Theorem 6.8.2. *For any $\epsilon > 0$, any $C > 0$, any m between $n^{1+o(1)}$ and $\Omega(n^2)$, and large enough n there exists a pair of distributions of graphs on n vertices, $\mathcal{D}^{YES}$ and $\mathcal{D}^{NO}$, such that the sizes of the maximum matchings of all graphs in $\mathcal{D}^{NO}$ are M and the sizes of the maximum matchings of all graphs in $\mathcal{D}^{YES}$ are at least CM. However, the total variation distance between an iid edge stream of length $m^{1-\epsilon}$ of a random graph in $\mathcal{D}^{YES}$ and one in $\mathcal{D}^{NO}$ is at most $1/10$.*

Overview of the approach. Our lower bound is based on a construction of two graphs G and H on n vertices such that for a parameter k **(a)** matching size in G is smaller than matching size in H by a factor of $n^{\Omega(1)/k}$ but **(b)** there exists a bijection from vertices of G to vertices of H that preserves k-depth neighborhoods up to isomorphism. To the best of our knowledge, this construction is novel. Related constructions have been shown in the literature (e.g., cluster trees of Kuhn et al. (2016)), but these constructions would not suffice for our lower bound, since they do not provide a property as strong as **(b)** above. For example, the construction of Kuhn et al. (2016) only produces one graph G with a large matching together with two subsets of vertices S, S' of G whose neighborhoods are isomorphic. This suffices for proving strong lower bounds on finding near-optimal matchings in a distributed setting Kuhn et al. (2016), but not for our purpose. Indeed, it is crucial for us to have a gap (i.e., two graphs G and H) and have the strong indistinguishability property provided by **(b)**.

Our construction proceeds in two steps. We first construct two graphs G' and H' that have identical

k-level degrees (see Section 6.8.3). This produces two graphs G' and H' that are indistinguishable based on k-level degrees (but whose neighborhoods are not isomorphic due to cycles) but whose matching size differs by an $n^{\Omega(1/k)}$ factor. These graphs have $n^{2-O(1/k)}$ edges and provide nearly tight instances for peeling algorithms that we hope may be useful in other contexts. We note that a similar step is used in the construction of cluster trees of Kuhn et al. (2016), but, as mentioned above, these graphs provide neither the indistinguishability property for all vertices nor a gap in matching size. Furthermore, the number of edges in the corresponding instances of Kuhn et al. (2016) is $\widetilde{O}(n^{3/2})$, i.e., the graphs do not get denser with large k, whereas our construction appears to have the optimal behaviour. The second step of our construction is a lifting map (see Theorem 6.8.22) that relies on high girth Cayley graphs and allows us to convert graphs with identical k-level vertex degrees to graphs with isomorphic depth-k neighborhoods without changing matching size by much. The details are provided in Section 6.8.4.

Finally, the proof of the sampling lower bound proceeds as follows. To rule out factor C approximation in $m^{1-o(1)}$ space, take a pair of constant (rather, $m^{o(1)}$) size graphs G and H such that **(a)** matching size in G is smaller than matching size in H by a factor of C and **(b)** for some large k one has that k-depth neighborhoods in G are isomorphic to k-depth neighborhoods in H. Then the actual hard input distribution consists of a large number of disjoint copies of G in the **NO** case and a large number of copies of H in the **YES** case, possibly with a small disjoint clique added in both cases to increase the number of edges appropriately. Since the vertices are assigned uniformly random labels in both cases, the only way to distinguish between the **YES** and the **NO** case is to ensure that at least k edge-samples land in one of the small copies of H or G. Since k is small, the result follows.

We now give the details. Formally, our main tool will be a pair of constant sized graphs that are indistinguishable if only some given constant number of edges are sampled from either. This is guaranteed by the following theorem, proved in Section 6.8.5.

Theorem 6.8.3. *For every $\lambda > 1$ and every k, there exist graphs G and H such that $MM(G) \geq \lambda \cdot MM(H)$, but for every graph K with at most k edges, the number of subgraphs of G and H isomorphic to K are equal.*

Defining distributions $\mathscr{D}^{\mathbf{YES}}$ and $\mathscr{D}^{\mathbf{NO}}$.

All the graphs from our distributions will have the same vertex set $V \equiv [n]$. Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be the two graphs provided by Theorem 6.8.3 invoked with parameters $\lambda = 2C$ and $k = 2/\epsilon$. Let $q \equiv \max(|V_G|, |V_H|)$ (our construction in fact guarantees that $|V_G| = |V_H|$). Let $s \equiv \mathrm{MM}(H)$, and hence $\mathrm{MM}(G) \geq \lambda \cdot s = 2C \cdot s$.

Partition V into the following:

1. $r = \frac{n}{2q}$ sets of size q, denoted by $V_1, \ldots, V_r$;
2. a set V_K consisting of w vertices, for $w \in [0, ns/q]$; and
3. set I containing the remaining vertices.

The sets $V_1, \ldots, V_r$ will serve as the vertex sets of copies of G or H, where V_i equals V_G or equals V_H

depending on whether we are constructing $\mathscr{D}^{\text{YES}}$ or $\mathscr{D}^{\text{NO}}$. The set V_K will be a clique, while I will be a set of isolated vertices in the construction. The distributions $\mathscr{D}^{\text{YES}}$ and $\mathscr{D}^{\text{NO}}$ are now defined as follows:

$\mathscr{D}^{\text{YES}}$: Take r independently uniformly random permutations $\pi_1,\dots,\pi_r$ on $V_1,\dots,V_r$ respectively, and construct a copy G_i of G embedded into V_i via π_i. Then construct a clique K_w on V_K.

$\mathscr{D}^{\text{NO}}$: Take r independently uniformly random permutations $\pi_1,\dots,\pi_r$ on $V_1,\dots,V_r$ respectively and construct a copy H_i of H embedded into V_i via π_i. Then construct a clique K_w on V_K.

We will refer to copies of G and H in the two distributions above as *gadgets*. We now give an outline of the proof of Theorem 6.8.2 assuming Theorem 6.8.3. The full proof follows the same steps, but is more involved, and is deferred to Appendix D.4.

Proof outline (of Theorem 6.8.2). Naturally, our distribution-pair will be $\mathscr{D}^{\text{YES}}$ and $\mathscr{D}^{\text{NO}}$ as defined above. Note that the maximum matching size of any element of the support of $\mathscr{D}^{\text{YES}}$ is at least $r\cdot 2Cs$ as it contains r copies of G. On the other hand, any element of the support of $\mathscr{D}^{\text{NO}}$ contains r copies of H as well as a clique of size $w \le ns/q$ which means its maximum matching size is at most $r\cdot s + ns/2q \le 2r\cdot s$. Hence, the sizes of maximum matchings in $\mathscr{D}^{\text{YES}}$ and maximum matchings in $\mathscr{D}^{\text{NO}}$ differ by at least factor C, as desired.

Note that the number of edges in the construction is at least $\binom{w}{2}$ and at most $\binom{w}{2}+r\cdot\binom{q}{2} \le w^2+qn$. Thus the number of edges can be set to be (within a constant factor of) anything from $n^{1+o(1)}$ to $\Omega(n^2)$.

Next, we compute the total variation distance between iid edge streams of length $m^{1-\epsilon}$ of a graph sampled from $\mathscr{D}^{\text{YES}}$ and $\mathscr{D}^{\text{NO}}$ respectively. Denote these random iid edge streams by C_1 and C_2, respectively. Consider the following event, that we call *bad*,

$$\mathscr{E} \equiv \{\exists i \in [r] : \text{edges between vertices of } V_i \text{ appear more than } k \text{ times in the stream}\}.$$

We show below that distributions of C_1 conditioned on $\bar{\mathscr{E}}$ is identical to the distribution of C_2 conditioned on $\bar{\mathscr{E}}$. Then the total variation distance is bounded by the probability of $\mathscr{E}$. We first bound this probability, and then prove the claim above.

Upper bounding the probability of $\mathscr{E}$. Consider a realization of $\mathscr{D}^{\text{YES}}$ or a realization of $\mathscr{D}^{\text{NO}}$. Since we have $m^{1-\epsilon}$ edge samples each chosen uniformly at random from a possible m edges, each specific edge, e, appears exactly $m^{-\epsilon}$ times throughout the stream in expectation. Therefore, by Markov's inequality the probability that e ever appears in the stream is at most $\frac{3n^{-\epsilon}}{\mu^2}$. Also, the event that an edge e_1 appears in the stream and the event that an edge $e_2 \neq e_1$ appears in the stream are negatively associated.

Fix a single gadget of the realized graph; this gadget has at most q^2 edges. Therefore, by union bound and from the negative association outlined above, the probability that at least $k+1$ edges of the gadget will be sampled is at most

$$\binom{q^2}{k+1}\cdot\left(m^{-\epsilon}\right)^{k+1} \le \binom{q^2}{k+1}\cdot m^{-2},$$

where we used the assuption that $k = 2/\epsilon$. Thus, again by union bound, the probability that any of the

$r = \frac{n}{2q}$ gadgets has at least $k+1$ of its edges occurring in the stream is at most

$$\frac{n}{2q} \cdot \binom{q^2}{k+1} \cdot m^{-2} \leq 1/10,$$

for large enough n and $m > n$. So, $\mathbb{P}[\mathscr{E}] \leq 1/10$ under both distributions.

Analyzing conditional distributions. It remains to show that C_1 and C_2 are identically distributed when conditioned on $\overline{\mathscr{E}}$. We now give an overview of this proof, and defer details to Appendix D.4. Since the cliques are identical across the two distributions, edges sampled from them are no help in distinguishing between $\mathscr{D}^{\text{YES}}$ and $\mathscr{D}^{\text{NO}}$. Consider a single gadget. (As a reminder, a gadget is a copy of G or H.) By conditioning on $\overline{\mathscr{E}}$ we have that only at most k distinct edges of this gadget are observed. Since the gadgets are randomly permuted in both $\mathscr{D}^{YES}$ and $\mathscr{D}^{NO}$, only the isomorphism-class of the sampled subgraph of the gadget provides any information. However, each isomorphism-class's probability is proportional to the number of times such a subgraph appears in the gadget. This is equal across the YES and NO cases thanks to the guarantee of Theorem 6.8.3 on G and H. □

From here it remains to prove Theorem 6.8.3. We organize the rest of this section as follows. In Section 6.8.3 we define and analyze our main construction, which is a pair of graphs isomorphic with respect to k-level degrees while having greatly different matching numbers. That is to say, we produce two graphs and a bijection between them such tha the bijection preserves degree structure up to a depth of k, disregarding cycles. (For the definition of k-level degree see Section 6.8.2.) This is the main technical result of our lower bound. Then in Section 6.8.4 we use a graph lifting construction to increase the girths of our graphs, resulting in a pair of graphs that have truly isomorphic k-depth neighborhoods. Finally in Section 6.8.5 we prove Theorem 6.8.3, concluding the lower bound.

6.8.2 Preliminaries

We begin by stating a few definition which will be used in the rest of Section 6.8. First, we recall a few basic graph theoretical definitions:

Definition 6.8.4 (c-star)**.** *We call a graph with a single degree c vertex connected to c degree 1 vertices a c-star. We call the degree c vertex the center and degree 1 vertices petals.*

Definition 6.8.5 (girth of a graph)**.** *The girth of a graph is the length of its shortest cycle.*

Definition 6.8.6 (permutation group of V)**.** *Let $G = (V, E)$ be a graph. We call the subgroup of S_V (the permutation group of V) containing all permutations that preserve edges the automorphism group of G. That is, a permutation $\pi \in S_V$ is in the automorphism group if the following holds:*

$$\forall v, w \in V : (v, w) \in E \iff (\pi(v), \pi(w)) \in E.$$

We denote it $Aut(G)$.

We now define two local property of a vertex, i.e., k-hop neighborhood and k-level degree.

Definition 6.8.7 (k-hop neighbourhood of a vertex)**.** *Let the k-hop neighborhood of a vertex v in graph G be defined as the subgraph induced by vertices of G with distance at most k from v.*

Definition 6.8.8 (k-level degree of a vertex)**.** *Let the k-level degree of a vertex v in a graph G denoted by $d_k^G(v)$ be a multiset defined recursively as follows:*

- $d_1^G(v) \equiv d(v)$, *the degree of v in G.*
- *For $k > 1$, $d_k^G(v) \equiv \left\{d_{k-1}^G(w) | w \in N(v)\right\}$, where this is a multiset.*

For ease of presentation, in the future we will use the following less intuitive but more explicit formulation:

$$d_k^G(v) = \biguplus_{w \in N(v)} \{d_{k-1}^G(w)\},$$

where $\uplus$ denotes multiset union. Moreover, if G is clear from context, we will omit the superscript and write $d_k(v)$ instead of $d_k^G(v)$.

Note that the above two definitions are similar but distinct, with the k-hop neighborhood containing the more information of the two. Imagine a vertex v of a C_3 cycle and a vertex w of a C_4 cycle. Their arbitrarily high level degrees are identical, but their 2-hop neighborhoods contain their respective graphs fully, and so they are clearly different.

Observation 6.8.9. *The following conditions are sufficient for the k-hop neighborhoods of vertices v and w (from graphs G and H respectively) to be isomorphic:*

1. $d_k^G(v) = d_k^H(w)$
2. *G and H both have girth at least $2k+2$, that is neither graph contains a cycle shorter than $2k+2$.*

This observation will be crucial to our construction as our main goal will be to construct two graphs G and H such that a bijection between their vertex-sets preserves k-depth neighborhoods. We achieve this by first guaranteeing condition 1. above, then improving our construction to guarantee condition 2. as well.

6.8.3 Constructing Graphs with Identical k-Level Degrees

As stated before, our first goal is to design a pair of graphs with greatly differing maximum matching sizes such that, nonetheless, there exists a bijection between them preserving k-level degrees for some large constant k. We will later improve this construction so that the bijection also preserves k-hop neighborhoods.

Definition 6.8.10 (Degree padding $\bar{G}$ of a graph G and special vertices)**.** *For every graph $G = (V, E)$, let d_l and d_h be minimum and maximum vertex degrees in G respectively. Define the degree padding $\bar{G} = (\bar{V}, \bar{E})$ of G as the graph obtained from G by adding a bipartite clique between vertices of degree d_l and a new set of $d_h - d_l$ vertices. More formally, let S be a set of $d_h - d_l$ nodes disjoint from V, let $\bar{V} := V \cup S$, and $\bar{E} := E \cup (S \times \{v \in V | d(v) = d_l\})$ (see Fig. 6.4). We refer to the set S in $\bar{G}$ as the set of* special vertices.

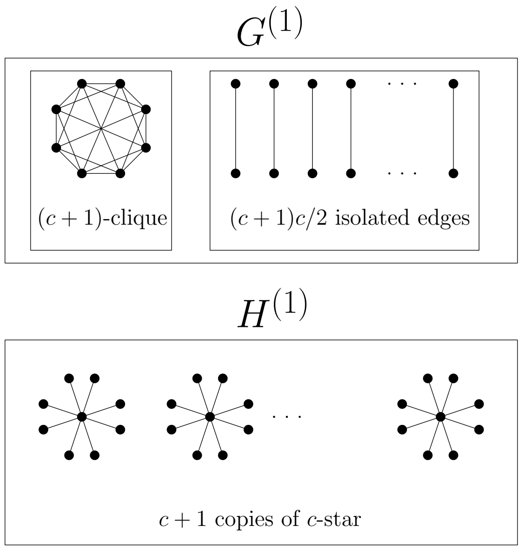


Figure 6.3 – Construction of $G^{(1)}$ and $H^{(1)}$.

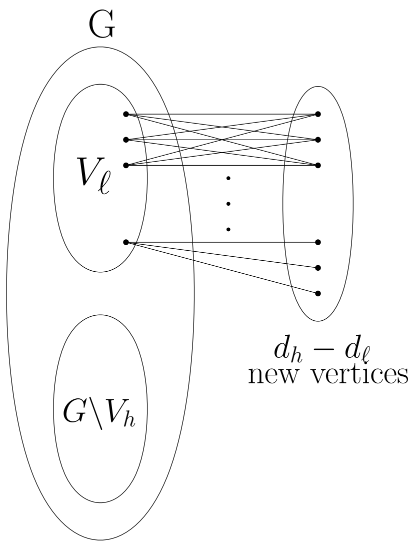


Figure 6.4 – Padding of a graph G.

Definition 6.8.11 (Recursive construction of k-depth similar graphs $G^{(k)}$ and $H^{(k)}$)**.** *For every integer $k \geq 1$ and integer $c \geq 1$, define graphs $G^{(k)}$ and $H^{(k)}$ recursively as follows. For $k = 1$, let $G^{(1)}$ be a graph that is the disjoint union of a $c+1$-clique and $(c+1)c/2$ isolated edges. Let $H^{(1)}$ be $c+1$ disjoint copies of c-stars (see Fig. 6.3). For $k > 1$, let $\bar{G}^{(k-1)}$ denote a degree padding of $G^{(k-1)}$ as per Definition 6.9.6, and let $G^{(k)} = \bar{G}_1^{(k-1)} \cup \ldots \cup \bar{G}_c^{(k-1)}$ denote the disjoint union of c copies of $\bar{G}^{(k-1)}$. Similarly let $\bar{H}^{(k)}$ denote the degree padding of $H^{(k-1)}$, and let $H^{(k)} = \bar{H}_1^{(k-1)} \cup \ldots \cup \bar{H}_c^{(k-1)}$ denote the disjoint union of c copies of $\bar{H}^{(k-1)}$.*

We first prove some simple structural claims about $G^{(j)}$ and $H^{(j)}$.

Lemma 6.8.12. *For graphs $G^{(j)}$ and $H^{(j)}$ as defined in Definition 6.8.11 there exist numbers $N_h^{(j)}$, $N_l^{(j)}$ and $d_h^{(j)} > d_l^{(j)}$ such that both $G^{(j)}$ and $H^{(j)}$ contain exactly $N_h^{(j)}$ vertices of degree $d_h^{(j)}$, $N_l^{(j)}$ vertices of $d_l^{(j)}$ and no other vertices. Let $V^{(j)}$ and $W^{(j)}$ denote the vertex-sets of $G^{(j)}$ and $H^{(j)}$, respectively. Furthermore, let $V_h^{(j)} \subseteq V^{(j)}$ and $W_h^{(j)} \subseteq W^{(j)}$ be the vertices of degree $d_h^{(j)}$; let $V_l^{(j)} \subseteq V^{(j)}$ and $W_l^{(j)} \subseteq W^{(j)}$ be vertices of degree $d_l^{(j)}$.*

Proof. We the prove the lemma by induction on j. Clearly, for $j = 1$, the claim of the lemma is satisfied with $N_h^{(1)} = c+1$, $N_l^{(1)} = (c+1)c$, $d_h^{(1)} = c$ and $d_l^{(1)} = 1$.

Inductive step: $j-1 \rightarrow j$**.** Consider $\bar{G}^{(j-1)}$ and $\bar{H}^{(j-1)}$. It is clear that the vertices of both of these graphs fall into two categories: the old vertices of $G^{(j-1)}$ and $H^{(j-1)}$ respectively as well as the newly introduced special vertices. The old vertices used to be of degree either $d_h^{(j-1)}$ or $d_l^{(j-1)}$ according to the inductive hypothesis. After the padding, the degree of those vertices is uniform and equal to $d_h^{(j-1)}$. Since the special vertices are connected to all the old vertices that used to have low degree in $G^{(j-1)}$ and $H^{(j-1)}$ respectively, the degrees of each special vertex is $N_l^{(j-1)}$. By definition, the number of special vertices in each of $\bar{G}^{(j-1)}$ and $\bar{H}^{(j-1)}$ is $d_h^{(j-1)} - d_l^{(j-1)}$.

All that remains to be proven is that the special vertices have strictly higher degree than the old vertices in, say $\bar{G}^{(j-1)}$, that is $N_l^{(j-1)} > d_h^{j-1}$. For $j = 2$ one can simply verify that this is true from the base construction of Definition 6.8.11. For $j > 2$ consider the structure of $G^{(j-1)}$: $G^{(j-1)} = \bar{G}_1^{(j-2)} \cup \ldots \cup \bar{G}_c^{(j-2)}$ where $\bar{G}_i^{(j-2)}$ are disjoint copies of the degree padded version of $G^{(j-2)}$. Hence, any high degree vertex of $G^{(j-1)}$ corresponds to a special vertex used in the padding of $G^{(j-2)}$ and so, no two high degree vertices of $G^{(j-1)}$ are connected to each other. So a high degre vertex of $G^{(j-1)}$ is only connected to its low degree vertices, and not even all of them, since $G^{(j-1)}$ falls into c disjoint copies. Therefore its degree, $d_h^{(j-1)}$ must be smaller than the number of low degree vertices in the same graph, $N_l^{(j-1)}$.

In conclusion, we have proved the equivalent of the lemma's statement for $\bar{G}^{(j-1)}$ and $\bar{H}^{(j-1)}$. It is easy to see that duplicating this c times will not disrupt this.

□

Lemma 6.8.13. *For graphs $G^{(j)}$ and $H^{(j)}$ as defined in Definition 6.8.11 and $c \geq 2k$, the following inequalities hold for the quantities from Lemma 6.8.12:*

- $N_h^{(j)} \leq c^j + 2jc^{j-1}$
- $N_l^{(j)} \leq c^{j+1} + 2jc^j$
- $d_h^{(j)} \leq c^j + 2jc^{j-1}$

Proof. We prove the claims by induction. As mentioned before $N_h^{(1)} = c+1$, $N_l^{(1)} = (c+1)c$ and $d_h^{(1)} = c$ which satisfies the inequalities.

Inductive step: $j-1 \to j$. Since these quantities can be derived equivalently from either $G^{(j)}$ or $H^{(j)}$ by Lemma 6.8.12, we will be looking at $G^{(j)}$ for simplicity. The set of high degree vertices in $G^{(j)}$ are the copies of the special vertices from degree padding $G^{(j-1)}$, which number $d_h^{(j-1)} - d_l^{(j-1)}$. So $N_h^{(j)} = c\left(d_h^{(j-1)} - d_l^{(j-1)}\right) \leq cd_h^{(j-1)} \leq c^j + 2(j-1)c^{j-1} \leq c^j + 2jc^{j-1}$ as claimed.

The set of low degree vertices in $G^{(j)}$ are copies of the old vertices from the unpadded $G^{(j-1)}$, which number $|V^{(j-1)}| = N_h^{(j-1)} + N_l^{(j-1)}$. So $N_l^{(j)} = c\left(N_h^{(j-1)} + N_l^{(j-1)}\right) \leq c^j + 2(j-1)c^{j-1} + c^{j+1} + 2(j-1)c^j \leq c^{j+1} + 2jc^j$ as claimed, since $2(j-1) \leq 2k \leq c$.

The high degree vertices in $G^{(j)}$ are copies of the special vertices added during the degree padding of $G^{(j-1)}$; their degree is $N_l^{(j-1)}$ as prescribed in Definition 6.9.6. So $d_h^{(j)} = N_l^{(j-1)} = c^j + 2(j-1)c^{(j-1)} \leq c^j + 2jc^{j-1}$ as claimed.

□

We will now show that the discrepancy in the matching numbers of $G^{(j)}$ and $H^{(j)}$ persists throughout the recursive construction.

Lemma 6.8.14. *For $c \geq 2k$ the graph $G^{(j)}$ as constructed in Definition 6.8.11 has maximum matching size at least $(c+1)c^j/2$ while $H^{(j)}$ from the same construction has maximum matching size at most $2jc^j$.*

Proof. We prove the following claim by induction: $G^{(j)}$ has a feasible matching of size $(c+1)c^j/2$ while $H^{(j)}$ has a feasible vertex cover of size $2jc^j$. For the case of $j=1$ this is clear: the set of isolated edges in $G^{(1)}$ constitutes a matching and the centers of the stars in $H^{(1)}$ constitute a vertex cover.

Inductive step: $j-1 \to j$. Degree padding does not affect the feasibility of a matching. Hence, when constructing $G^{(j)}$ by duplicating $\bar{G}^{(j-1)}$ c times, to construct a matching in $G^{(j)}$ we can simply duplicate the matchings from $\bar{G}^{(j-1)}$ as well. This results in a matching in $G^{(j)}$ that is c times larger than the one in $\bar{G}^{(j-1)}$. Therefore, by the inductive hypothesis $G^{(j)}$ has a sufficiently large feasible matching.

Now we discuss the size of minimum vertex cover in $H^{(j)}$. Upon degree padding $H^{(j-1)}$ we add the special vertices to the vertex cover, increasing its size by $d_h^{(j-1)} - d_l^{(j-1)} \le c^{j-1} + 2(j-1)c^{j-2}$. When duplicating $\bar{H}^{(j-1)}$ c times we duplicate the vertex cover as well. This makes the size of our feasible vertex cover of $H^{(j)}$ at most $2(j-1)c^j + c^j + 2(j-1)c^{j-1} \le 2jc^j$ as claimed, since $2(j-1) \le 2k \le c$.

□

Next we define a sequence of bijections between vertex-sets of $G^{(j)}$ and $H^{(j)}$. We begin by giving the following definitions.

Each of the bijections between the vertex-sets of $G^{(j)}$ and $H^{(j)}$ that we define preserves j-level degree.

Lemma 6.8.15. *For $c \ge 2k$, let $\widetilde{G} \equiv G^{(j)}$ and $\widetilde{H} \equiv H^{(j)}$ be defined as in Definition 6.8.11. Then, there exists a bijection $\Phi^{(j)} : V^{(j)} \to W^{(j)}$ such that for every $v \in W^{(j)}$ it holds $d_j^{\widetilde{H}}(v) = d_j^{\widetilde{G}}(\Phi^{(j)}(v))$.*

Proof. We prove this lemma by induction on j.

Base case: $j = 1$. Consider the following bijection $\Phi^{(1)}$ that maps the vertices from $G^{(1)}$ to $H^{(1)}$: $\Phi^{(1)}$ maps vertices of degree c (vertices in the clique in $G^{(1)}$) to the centers of stars in $H^{(1)}$, and endpoints of isolated edges in $G^{(1)}$ to petals in $H^{(1)}$. $\Phi^{(1)}$ can be arbitrary as long as it satisfies this constraint (and remains a bijection). Observe that $\Phi^{(1)}$ is a bijection such that the vertices of $G^{(1)}$ are mapped to vertices of the same degree in $H^{(1)}$.

Inductive step: $j-1 \to j$. We first define a bijection $\Phi^{(j)}$ and then argue that it maps the vertices of $G^{(j)}$ to the vertices of $H^{(j)}$ with the same j-level degree.

We define $\Phi^{(j)}$ inductively on j as follows. Recall that $G^{(j)}$ is a disjoint union of c copies of the degree padding $\bar{G}^{(j-1)}$ of $G^{(j-1)}$, and $H^{(j)}$ is a disjoint union of c copies of the degree padding $\bar{H}^{(j-1)}$ of $H^{(j-1)}$. To define $\Phi^{(j)}$, we "reuse" $\Phi^{(j-1)}$ (which exists by the inductive hypothesis) and add the mapping for the vertices not included by $\Phi^{(j-1)}$; these vertices are called *special* (see Definition 6.9.6). By Lemma 6.8.12 the number of special vertices in $\bar{G}_i^{(j-1)}$ equals the number of special vertices in $\bar{H}_i^{(j-1)}$, for every $i = 1, \ldots, c$. So, we define $\Phi^{(j)}$ to map the special set A_i of vertices in $\bar{G}_i^{(j-1)}$ bijectively (and arbitrarily) to the special set B_i of vertices in $\bar{H}_i^{(j-1)}$.

For the sake of brevity, let $d_\iota(v) \equiv d_\iota^{G^{(j-1)}}$ for vertices v of $G^{(j-1)}$ and $d_\iota(v) \equiv d_\iota^{H^{(j-1)}}$ for vertices of $H^{(j-1)}$. Similarly, let $\bar{d}_\iota(v) \equiv d_\iota^{\bar{G}^{(j-1)}}$ for vertices v of $\bar{G}^{(j-1)}$ and $\bar{d}_\iota(v) \equiv d_\iota^{\bar{H}^{(j-1)}}(v)$ for vertices of $\bar{H}^{(j-1)}$. Furthermore, let $N(v)$ and $\bar{N}(v)$ denote the neighborhood of vertex v in $G^{(j-1)} \cup H^{(j-1)}$ and in $\bar{G}^{(j-1)} \cup \bar{H}^{(j-1)}$, respectively.

Since $\widetilde{G}$ and $\widetilde{H}$ are c disjoint copies of $\bar{G}^{(j-1)}$ and $\bar{H}^{(j-1)}$, respectively, it suffices to show that $\Phi^{(j)}$ maps a vertex of $\bar{G}^{(j-1)}$ to a vertx of $\bar{H}^{(j-1)}$ with the same j-level degree. Recall that $V(\bar{G}^{(j-1)}) = V^{(j-1)} \cup A$, where $V^{(j-1)} = V(G^{(j-1)})$ and A refers to the special vertices added to $G^{(j-1)}$. Similarly, recall that $V(\bar{H}^{(j-1)}) = W^{(j-1)} \cup B$. For the rest of our proof, we show the following claim.

Claim 6.8.16. *We have that:*

(A) For any $u \in V^{(j-1)}$ and $u' \in W^{(j-1)}$ (not necessarily mapped to each other by the bijection) if $d_{\iota-1}(u) = d_{\iota-1}(u')$, then $\bar{d}_\iota(u) = \bar{d}_\iota(u')$. (For the purposes of this statement, let $d_0^G(v) = \emptyset$ for every v.)

(B) For any $u \in A$ and $u' \in B$ it holds $\bar{d}_\iota(u) = \bar{d}_\iota(u')$.

Proof. We prove this claim by induction.

Base case: $\iota = 1$.

Proof of Item (A) By construction of $\bar{G}^{(j-1)}$ and $\bar{H}^{(j-1)}$, all vertices in either $V^{(j-1)}$ or $W^{(j-1)}$ have degree exactly $d_h^{(j-1)}$, so their 1-level degrees are equal.

Proof of Item (B) By construction of $\bar{G}^{(j-1)}$ and $\bar{H}^{(j-1)}$, all vertices in either A or B have degree exactly $d_l^{(j-1)}$, so their 1-level degrees are equal.

Inductive step: $\iota - 1 \to \iota$.

Proof of Item (A) Let u and u' be two vertices satisfying condition Item (A) for $\iota - 1$, i.e., $d_{\iota-1}(u) = d_{\iota-1}(u')$. Then, by definition

$$\biguplus_{\omega \in N(u)} \{d_{\iota-2}(\omega)\} = \biguplus_{\omega' \in N(u')} \{d_{\iota-2}(\omega')\}.$$

This means that there exists a bijection between the $G_i^{(j-1)}$-neighborhoods of u and u', denoted by $\Phi_u^* : N(u) \to N(u')$ that preserves $\iota-2$-level degrees. For any $\omega \in N(u)$ and $\omega' \in N(u')$ such that $\Phi_u^*(\omega) = \omega'$, by Item (A) of the inductive hypothesis it is also true that $\bar{d}_{\iota-1}(\omega) = \bar{d}_{\iota-1}(\omega')$. Therefore

$$\biguplus_{\omega \in N(u)} \{\bar{d}_{\iota-1}(\omega)\} = \biguplus_{\omega' \in N(u')} \{\bar{d}_{\iota-1}(\omega')\}. \tag{6.29}$$

To prove the inductive step for Item (A), we will show that

$$\biguplus_{\omega \in \bar{N}(u)} \{\bar{d}_{\iota-1}(\omega)\} = \biguplus_{\omega' \in \bar{N}(u')} \{\bar{d}_{\iota-1}(\omega')\} \tag{6.30}$$

as follows. Note that in Eq. (6.30) the neighbors w iterate over $\bar{N}$, while in Eq. (6.29) they iterate over N. Now we consider two cases.

Case $u \in V_h^{(j-1)}$**:** It follows that $u' \in W_h^{(j-1)}$. Then we are done, since u is not connected to A and its neighborhoods in $G_1^{(j-1)}$ and $\bar{G}_1^{(j)}$ are identical. Similarly for u'.

Case $u \in V_l^{(j-1)}$**:** It follows that $u' \in W_l^{(j-1)}$. Then $\bar{N}(u) \setminus N(u) = A$. Similarly $\bar{N}(u') \setminus N(u') = B$. It holds that $|A| = |B|$ and, by Item (B) of the inductive hypothesis, any vertex of A and any vertex of B have identical $\bar{d}_{\iota-1}$-degrees. Therefore, extending the multiset-union from $N(u)$ and $N(u')$ to $\bar{N}(u)$ and $\bar{N}(u')$, respectively, preserves the equality of $\bar{d}_\iota$-degrees.

Hence, in both cases it holds that $\bar{d}_\iota(u) = \bar{d}_\iota(u')$, as claimed.

Proof of Item (B) Consider vertices $u \in A$ and $u' \in B$. In this case $\bar{N}(u) = V_l^{(j-1)}$ and $\bar{N}(u') = W_l^{(j-1)}$, so our goal is to prove that

$$\biguplus_{\omega \in V_l^{(j-1)}} \{\bar{d}_{\iota-1}(\omega)\} = \biguplus_{\omega' \in W_l^{(j-1)}} \{\bar{d}_{\iota-1}(\omega')\}$$

or, equivalently, we aim to show that there exists a bijection between $V_l^{(j-1)}$ and $W_l^{(j-1)}$ that preserves $\bar{d}_{\iota-1}$-degree.

By the claim of the *outer* inductive hypothesis, i.e., by Lemma 6.8.15, there exists a bijection $\Phi^{(j-1)}$ that preserves the d_{j-1}-degree between $V^{(j-1)}$ and $W^{(j-1)}$. Since $\Phi^{(j-1)}|_{V_l^{(j-1)}}$ preserves d_{j-1} degree it also preserves $d_{\iota-2}$-degree and by Item (A) it also preserves $\bar{d}_{\iota-1}$-degree. Thus, Item (B) holds.

□

To conclude the proof of Lemma 6.8.15 consider again a vertex $v \in V^{(j-1)} \cup A$ to which $\Phi^{(j)}$ can be applied. If $v \in V^{(j-1)}$, then the claim holds by Claim 6.8.16 Item (A); if it is in A, then the claim holds by Claim 6.8.16 Item (B). □

Corollary 6.8.17. *For large enough $c \geq 2k$ there exists a bijection $\Phi^{(k)} : V^{(k)} \to W^{(k)}$ such that for any $v \in V^{(k)}$ $d_k(v) = d_k(\Phi^{(k)}(v))$. However, $MM(G_1)$ and $MM(G_2)$ differ by at least a factor $\frac{c+1}{4k}$.*

Proof. By Lemma 6.8.14 and Lemma 6.8.15 the graphs $G^{(k)}$ and $H^{(k)}$ constructed in Definition 6.8.11 satisfy these requirements when $c \geq 2k$ □

6.8.4 Increasing Girth via Graph Lifting

Let us start this section by introducing Cayley graphs, we will later use them in order to increase the girth. Girth refers to the minimum length of a cycle within a graph.

Definition 6.8.18. *Let $\mathcal{G}$ be a group and S a generating set of elements. The Cayley graph associated with $\mathcal{G}$ and S is defined as follows: Let the vertex set of the graph be $\mathcal{G}$. Let any two elements of the vertex set, g_1 and g_2 be connected by an edge if and only if $g_1 \cdot s = g_2$ or $g_1 \cdot s^{-1} = g_2$ for some element $s \in S$. We can think of the edges of a Cayley graph as being directed and labeled by generator elements from S.*

Remark 6.8.19. *Consider traversing a (undirected) walk in a Cayley graph. Let the sequence of edge-labels of the walk be $s_1, s_2, \ldots, s_l$. Further, let $\epsilon_1, \epsilon_2, \ldots, \epsilon_l$ be ± 1 variables, where ϵ_i corresponds to whether we have crossed the i^{th} edge in the direction associated with right-multiplication by s_i ($\epsilon_i = 1$) or in the* *direction associated with right-multiplication by s_i^{-1} ($\epsilon_i = -1$). Then traversing the walk corresponds to multiplication by $s_1^{\epsilon_1} \cdot s_2^{\epsilon_2} \cdot \ldots \cdot s_l^{\epsilon_l}$, that is the final vertex of the walk corresponds to the starting vertex multiplied by this sequence.*

Definition 6.8.20. *Let $\mathscr{G}$ be a group. A sequence of elements $s_1^{\epsilon_1} \cdot s_2^{\epsilon_2} \cdot \ldots \cdot s_l^{\epsilon_l}$, from some generator set S, is considered irreducible if no two consecutive elements cancel out. That is*

$$\nexists i : s_i^{\epsilon_i} \cdot s_{i+1}^{\epsilon_{i+1}} = \mathbb{1}.$$

Remark 6.8.21. *By the previous remark, circuits of a Cayley graph associated with $\mathscr{G}$ and S correspond to irreducible sequences from S multiplying to $\mathbb{1}$. A circuit is a closed walk with no repeating edges. (Note that the other direction is not true: Not all irreducible sequences from S multiplying to $\mathbb{1}$ correspond to circuits in the Cayley graph, as they could have repeating edges.)*

Taking $(G_1^{(k)}, G_2^{(k)})$ from the previous section we now have a pair of graphs that differ greatly in their maximum matching size but are identical with respect to their k-level degree composition. In order to turn this result into a hard instance for approximating the maximum matching size, we need the additional property that both graphs are high girth (particularly $\geq 2k+2$).

Theorem 6.8.22. *For every graph $G = (V, E), |V| = n$, every integer $g \geq 1$, there exists an integer $R = R(n, g) = n^{O(g)}$ and a graph $L = (V_L, E_L), V_L = V \times [R]$, such that the following conditions hold.*

(1) Size of the maximum matching of L is multiplicatively close to that of G: $R \cdot MM(G) \leq MM(L) \leq 2R \cdot MM(G)$;

(2) L contains no cycle shorter than g (i.e., L has high girth);

(3) For every $k \in \mathbb{N}$ and every $v \in V$ one has, for all $r \in R$ that $d_k^H((v, r)) = d_k^G(v)$.

We refer to L as the lift of G.

Before proving this theorem, let us state a key lemma that we use in proving it. The proof of this lemma is deferred to Appendix D.4.2.

Lemma 6.8.23. *For any parameters g and l, there exists a group $\mathscr{G}$ of size $l^{O(g)}$ along with a set of generator elements S of size at least l, such that the associated Cayley graph (Definition 6.8.18) has girth at least g.*

Equivalently, this means that no irreducible sequence of elements from S and their inverses, shorter than g, equates to the identity. We are now ready to prove Theorem 6.8.22.

Proof of Theorem 6.8.22. Let $\mathscr{G}$ be a group according to Lemma 6.8.23 with parameter $l = |E|$, and let $S \subseteq \mathscr{G}$ denote the set of elements of $\mathscr{G}$ whose Cayley graph has girth at least g, as guaranteed by Lemma 6.8.23. We think of the elements of S as indexed by the edges of the graph G, and write $S = (s_e)_{e \in E}$. We direct the edges of G arbitrarily, and define the edge-set E_L of L as

$$E_L \equiv \left\{((v_1, g_1), (v_2, g_2)) \in V_L \times V_L | e = (v_1, v_2) \in E \text{ and } g_1 \cdot s_e = g_2\right\}.$$

That is we connect vertices $(v_1, g_1), (v_2, g_2) \in V_L$ by an edge if and only if $e = (v_1, v_2)$ is an (directed) edge in E and $g_1 \cdot s_e = g_2$ in $\mathscr{G}$. In this construction $R = |\mathscr{G}| = m^{O(g)} = n^{O(g)}$ as stated in the theorem.

Note that every vertex v in the original graph G corresponds to an independent set of R vertices $v \times \mathscr{G}$ in L. For any pair of vertices v_1 and v_2 in the original graph if $(v_1, v_2) \in E$, then the subgraph induced by $(v_1 \times \mathscr{G}) \cup (v_2 \times \mathscr{G})$ is a perfect matching; if not, the union $(v_1 \times \mathscr{G}) \cup (v_2 \times \mathscr{G})$ forms an independent set. Overall, every edge $e = (v_1, v_2) \in E$ can be naturally mapped to R edges among the edges of L, namely

$$\left\{((v_1, g_1), (v_2, g_1 \cdot s_e)) \in V_L \times V_L | g_1 \in \mathscr{G}\right\}.$$

Property Item (1) . Any matching M of G can be converted into a matching M^L of size $R|M|$ in L, for instance $M^L = \{((v_1, g_1), (v_2, g_2)) \in E_L | (v_1, v_2) \in M \text{ and } g_1 \in \mathscr{G}\}$. As mentioned above, g_2 is uniquely defined here. The same statement is true for vertex covers: If V is a vertex cover in G, then $V \times \mathscr{G}$ is a vertex cover in L. Therefore,

$$R \cdot \mathrm{MM}(G) \leq \mathrm{MM}(L) \leq \mathrm{VC}(L) \leq R \cdot \mathrm{VC}(G) \leq 2R \cdot \mathrm{MM}(G),$$

where the last inequality is due to the fact that the minimum vertex cover of a graph is at most twice the size of its maximum matching.

Property Item (2) . Toward contradiction, suppose C is a short cycle in L, that is it has length $f < g$. Let C be

$$((v_1, g_1)(v_2, g_2), \ldots, (v_h, g_h)(v_{f+1}, g_{f+1}) = (v_1, g_1)).$$

Let $e_i := ((v_i, g_i), (v_{i+1}, g_{i+1})) \in E_L$. Let ϵ_i be a ± 1 variable indicating whether the direction of the edge e_i is towards v_{i+1} ($\epsilon_i = 1$) or towards v_i ($\epsilon_i = -1$). (Recall that the edges of G were arbitrarily directed during the construction of L.)

If C as described above is indeed a cycle, this means that $g_1 \cdot s_{e_1}^{\epsilon_1} \cdot s_{e_2}^{\epsilon_2} \cdot \ldots \cdot s_{e_f}^{\epsilon_f} = g_1$. Therefore $s_{e_1}^{\epsilon_1} \cdot s_{e_2}^{\epsilon_2} \cdot \ldots \cdot s_{e_f}^{\epsilon_f}$ is an irreducible sequence of elements from S and their inverses shorter than g that equates to unity. Indeed, if it was not irrdeducible, that is an element and its inverse appeared consecutively, then that would mean C crossed an edge twice consecutively ($e_i = e_{i+1}$ for some i) and therefore it would not be a true cycle. This is a contradiction of theorem Lemma 6.8.23, so L must have girth at least g.

Property Item (3) . The third statement is proven by induction on k. The **base case** is provided by $k = 1$. Fix v and $h \in \mathscr{G}$. Every neighbor of v in G corresponds to exactly one neighbor of (v, h) in L. Indeed, $w \in N(v)$ (such that $e = (v, w) \in E$) corresponds to $(w, h \cdot s_e^{\epsilon})$ where ϵ indicates the direction of e. Therefore $d^G(v) = d^L((v, h))$.

We now show the **inductive step** $(k-1 \to k)$**.** Again, fix v and h. Similarly to the base case, every neighbor $w \in N(v)$ corresponds to a single neighbor of (v, h) in L: (w, h_w) for some $h_w \in \mathscr{G}$. By the inductive hypothesis $d_{k-1}^G(w) = d_{k-1}^L((w, h_w))$. So

$$d_k^G(v) = \biguplus_{w \in N(v)} \{d_{k-1}^G(w)\} = \biguplus_{w \in N(v)} \{d_{k-1}^L((w, h_w))\} = d_k^L((v, h))$$

□

Thus, there exists a pair of graphs G_1 and G_2 such that there is a bijection between their vertex-sets that preserves high level degrees up to level k and such that neither G_1 nor G_2 contains a cycle shorter than $2k+2$. Furthermore, $\mathrm{MM}(G_1)$ and $\mathrm{MM}(G_2)$ differ by a factor of at least $\frac{c+1}{8k}$, which can be set to be arbitrarily high by the choice of c.

Corollary 6.8.24. *For $c \geq 2k$, there exists a pair of graphs G and H with vertex sets V and W, respectively, such that there is a bijection $\Phi : V \to W$ with the property that the k-depth neighborhoods of v and $\Phi(v)$ are isomorphic. Also, $MM(G) \geq \frac{c+1}{8k} MM(H)$.*

Proof. This follows directly from Corollary 6.8.17, Theorem 6.8.22 and Observation 6.8.9. Indeed, consider graphs G' and H' guaranteed by Corollary 6.8.17 with the same parameters. Apply to each Theorem 6.8.22 to get the lifted graphs G and H respectively. The bijection Φ guaranteed in Corollary 6.8.17 extends naturally to G and H. By the guarantee of Theorem 6.8.22 this bijection still preserves k-level degrees, and furthermore, both G and H have girth at least $2k+2$. By Observation 6.8.9 this is sufficient to show Corollary 6.8.24. □

6.8.5 k-Edge Subgraph Statistics in G and H

The main result of this section is the equality of numbers of subgraphs in G and H. This will result in proving Theorem 6.8.3.

Definition 6.8.25 (Subgraph counts)**.** *For a graph $G = (V, E)$ and any graph K we let*

$$\#(K : G) = |\{U \subseteq E | U \cong K\}|,$$

where we write $U \cong K$ to denote the condition that U is isomorphic to K.

Lemma 6.8.26. *Let $k \geq 1$ be an integer and let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two graphs such that a bijection $\Phi : V_G \to V_H$ between their vertex-sets preserves the k-depth neighborhoods. That is, for every $v \in V_G$, the k-depth neighborhood of v is isomorphic to that of $\Phi(v)$. Then for any graph $K = (V_K, E_K)$ of at most k edges $\#(K : G) = \#(K : H)$.*

Proof. We prove below that if

$$\alpha_K := |\{\Psi : V_K \hookrightarrow V_G | \forall (u, w) \in E_K : (\Psi(u), \Psi(w)) \in E_G\}|$$

and

$$\beta_K := |\{\Psi : V_K \hookrightarrow V_H | \forall (u, w) \in E_K : (\Psi(u), \Psi(w)) \in E_H\}|,$$

then $\alpha_K = \beta_K$. Since $\alpha_K = \#(K : G) \cdot |\mathrm{Aut}(K)|$ and $\beta_K = \#(K : H) \cdot |\mathrm{Aut}(K)|$, where $|\mathrm{Aut}(K)|$ is the number of automorphisms of K, then result then follows.

We proceed by induction on the number of connected components q in K. We start with the **base case** ($q = 1$), which is when K is connected, i.e. K has one connected component. Select arbitrarily a root

$r \in V_K$ of K. Define further, for all $v \in V_G$ and $v \in V_H$ respectively

$$\#(K : G|v) = |\{\Psi : V_K \hookrightarrow V_G | \forall (u, w) \in E_K : (\Psi(u), \Psi(w)) \in E_G \ \wedge \ \Psi(r) = v\}|$$

and

$$\#(K : H|v) = |\{\Psi : V_K \hookrightarrow V_H | \forall (u, w) \in E_K : (\Psi(u), \Psi(w)) \in E_H \ \wedge \ \Psi(r) = v\}|$$

so that $\#(K : G) = \sum_{v \in V_G} \#(K : G|v)$ and $\#(K : H) = \sum_{v \in V_H} \#(K : H|v)$. Recall that there exists a bijection $\Phi : V_G \to V_H$ such that for every $v \in V_G$ the k-depth neighborhood of v in G is identical to the k-depth neighborhood of $\Phi(v)$ in H. Since the number of edges in K is at most k, and K is connected, we get that $\#(K : G|v) = \#(K : H|\Phi(v))$, and hence

$$\#(K : G) = \sum_{v \in V_G} \#(K : G|v) = \sum_{v \in V_G} \#(K : H|\Phi(v)) = \sum_{v \in V_H} \#(K : H|v) = \#(K : H),$$

as required.

We now provide the **inductive step ($q - 1 \to q$).** Since $q \geq 2$, we let $K_1 = (V_{K_1}, E_{K_1})$ and $K_2 = (V_{K_2}, E_{K_2})$ be a bipartition of K into two disjoint non-empty subgraphs, i.e., V_K is the disjoint union of V_{K_1} and V_{K_2} and E_K is the disjoint union of E_{K_1} and E_{K_2}. Since the number of components of K_1 and K_2 are both smaller than q, we have by the inductive hypothesis that $\alpha_{K_1} = \beta_{K_1}$ and $\alpha_{K_2} = \beta_{K_2}$.

We will write the number of embeddings of K into G (resp. H) in terms of the number of embeddings of K_1, K_2 into G (resp. H), as well as embeddings of natural derived other graphs. Indeed, every pair of embeddings (Ψ_1, Ψ_2), where $\Psi_i : V_{K_i} \hookrightarrow V_G, i \in \{1, 2\}$, naturally defines a mapping $\Psi : V_F \to V_G$. However, this mapping is not necessarily injective. Indeed $\Psi_1(v_1)$ might clash with $\Psi_2(v_2)$ for some pairs $(v_1, v_2) \in V_{K_1} \times V_{K_2}$. In this case Φ defines different graph K' which we get by merging all clashing pairs from $V_{K_1} \times V_{K_2}$, aling with an embedding of K' into G. Note that K' has strictly fewer than q components. We call such a pair (Ψ_1, Ψ_2) an *K'-clashing pair*. We now get

$$\begin{aligned}
\alpha_K &= |\{\text{embedding } \Psi \text{ of } K \text{ into } G\}| \\
&= |\{(\Psi_1, \Psi_2) | \text{embeddings of } K_i \text{ into } G, i \in \{1, 2\}\}| \\
&\qquad - \sum_{\substack{K' \text{ graph with} \\ < q \text{ components}}} |\{(\Psi_1, \Psi_2) | K' - \text{clashing embeddings of } K_i, i \in \{1, 2\} \text{ into } G\}| \\
&= |\{(\Psi_1, \Psi_2) | \text{embeddings of } K_i \text{ into } H, i \in \{1, 2\}\}| \\
&\qquad - \sum_{\substack{K' \text{ graph with} \\ < q \text{ components}}} |\{(\Psi_1, \Psi_2) | K' - \text{clashing embeddings of } K_i, i \in \{1, 2\} \text{ into } H\}| \\
&= \beta_K,
\end{aligned}$$

where in the third transition above we used the fact that for any K' with $< q$ connected components one has

$$\begin{aligned} &|\{(\Psi_1,\Psi_2)|K'-\text{clashing embeddings of } K_i, i\in\{1,2\} \text{ into } G\}| \\ =&|\{\Psi'|\text{embeddings of } K' \text{ into } G\}| \\ =&|\{\Psi'|\text{embeddings of } K' \text{ into } H\}| \quad \text{(by inductive hypothesis)} \\ &|\{(\Psi_1,\Psi_2)|K'-\text{clashing embeddings of } K_i, i\in\{1,2\} \text{ into } H\}|. \end{aligned}$$

This completes the proof. □

Corollary 6.8.27 (Theorem 6.8.3)**.** *For every $\lambda > 1$ and every k, there exist graphs G and H such that $MM(G) \geq \lambda \cdot MM(H)$, but for every graph K with at most k edges, the number of subgraphs of G and H isomorphic to K are equal.*

Proof. By Lemma 6.8.26 the pair of graphs satisfying the guarantee of Corollary 6.8.24 will satisfy the guarantees of Theorem 6.8.3 as well. We just need to set c such that $c \geq 2k$ and $\frac{c+1}{8(k+1)} \geq \lambda$.

□

6.9 Analysis of the algorithm on a random permutation stream

6.9.1 Introduction and Technical Overview

In this section we focus on the setting in which the set of elements, e.g., edges, is presented as a random permutation. This has been a popular model of computation for graph algorithms in recent years with many results, including for matching size approximation Konrad et al. (2012); Kapralov et al. (2014); Monemizadeh et al. (2017); Peng and Sohler (2018); Assadi et al. (2019a). As mentioned before, our goal is to show that Algorithm 15 is robust to the correlations introduced by replacing independent samples with a random permutation. This results in algorithm for approximating matching size to within a factor of $O(\log^2 n)$, in polylogarythmic space.

Theorem 6.1.4. *There exists an algorithm that given access to a random permutation edge stream of a graph $G = (V, E)$, with n vertices and $m \geq 3n$ edges, produces an $O(\log^2 n)$ factor approximation to maximum matching size in G using $O(\log^2 n)$ bits of memory in a single pass over the stream with probability at least* $3/4$.

Recall that this improves the previous best-known approximation ratio (Kapralov et al. (2014)) by at least a factor $O(\log^6 n)$.

Our overall strategy consists of showing that the algorithm behaves identically when applied to iid samples or a permutation. That is, we show that distribution of the state of the algorithm at each moment is similar under these two settings. To this end, we use total variation distance and KL-divergence as a measure of similarity of distributions. Furthermore, we break down the algorithm to the level of LEVEL-j-TEST tests to show that these behave similarly.

More specifically, consider an invocation of LEVEL-j-TEST(v) in either the iid or the permutation stream.

In the permutation stream, we have already seen edges pass and therefore know that they will not reappear during the test; this biases the output of the test compared to the iid version which is oblivious to the prefix of the stream. However, we are able to prove in Section 6.9.4 that KL-divergence between the output-distribution of these two versions is proportional to the number of samples used, with a factor of $O(\log^2 n/m)$, see Lemma 6.9.11. Since this is true for all tests, intuitively, the algorithm should still work in the permutation setting as long as it uses $O(m/\log^2 n)$ samples. Conversely, our original algorithm uses $\Theta(m)$ samples; however, we can reduce the number of samples used by slightly altering it, at the expense of a $O(\log^2 n)$ factor in the approximation ratio.

In fact the algorithm we use in permutation stream setting is nearly identical to the one defined in Section 6.3; the one difference is that in the subroutine EDGE-LEVEL-TEST we truncate the number of tests to $J - \Omega(\log^2 n)$ to reduce the number of edges used, (see Algorithm 29 in Algorithm 29).

One technical issue that arises in carrying out this approach is the fact that KL-divergence does not satisfy the triangle inequality, not even an approximate one, when the distributions of the random variables in question can be very concentrated. (Or, equivalently, we can assume very small values, since we are thinking of Bernoulli variables). Essentially triangle inequality is not satisfied even approximately when some of the random variables in question are nearly deterministic (see Remark 6.9.5 in Section 6.9.2).

We circumvent this problem using a mixture of total variation and KL-divergence bounds. Furthermore, we develop a weaker version of triangle inequality for KL-divergence, see Lemma 6.9.7. This both loses a constant factor in the inequality and is only true under the condition that the variables involved are bounded away from deterministic. However, it suffices for our proofs in Sections 6.9.3 and 6.9.4.

Our proof strategy consists of two steps: we first modify the tests somewhat to ensure that all relevant variables are not too close to deterministic (see the padded tests presented in Section 6.9.3), at the same time ensuring that the total variation distance to the original tests is very small (see Lemma 6.9.9 in Section 6.9.3). We then bound the KL-divergence between the padded tests on the iid and permutation stream in Section 6.9.4. An application of Pinsker's inequality then completes the proof.

6.9.2 Preliminaries

Throughout this section, for sake of simplicity, we use n as an upperbound on the maximum degree d of G. This does not affect the guarantees that we provide.

Next we provide some notations that we will use in the sequel. Let Π be the random variable describing the permutation stream. Let the residual graph at time t be G^t, that is the original graph, but with the edges that have already appeared in the stream up until time t deleted, for $0 \le t \le m$. In this case G^0 is simply the original input graph and G^m is the empty graph of n isolated vertices. Let the residual degree of v at time T be

$$d^t \equiv \frac{d^{G^t}}{1 - t/m}.$$

Let the outcome of a jth level test on vertex v (recall LEVEL-j-TEST from Algorithm 16) be $T_j^{\mathrm{IID}}(v)$. Let of the same test on the permutation stream, performed at time t will be denoted $T_j^{\pi,t}(v)$.

Definition 6.9.1 (KL-divergence)**.** *For two distributions P and Q over $\mathcal{X}$, the KL-divergence of P and Q is*

$$D_{KL}(P\|Q) = \sum_{x\in\mathcal{X}} -P(x)\log\left(\frac{Q(x)}{P(x)}\right).$$

Remark 6.9.2. *Throughout this document,* log *is used to denote the natural logarithm in the definition of KL-divergence, even though it is conventionally the base* 2 *logarithm. This only scales down $D_{KL}(P\|Q)$ by a factor of* $\log 2$ *and does not affect any of the proofs.*

Lemma 6.9.3 (Chain rule)**.** *For random vectors $P = (P_i)_i$ and $Q = (Q_i)_i$,*

$$D_{KL}(P\|Q) = \sum_i \mathbb{E}_{x_1,\ldots,x_{i_1}}[D_{KL}(P_i\|Q_i|x_1,\ldots,x_{i-1})]$$

Lemma 6.9.4. *For every $p,\epsilon \in [0,1]$ such that $p+\epsilon \in [0,1]$ one has*

$$D_{KL}\left(Ber(p+\epsilon)\|Ber(p)\right) = \frac{16\epsilon^2}{p(1-p)}$$

A proof of Lemma 6.9.4 is provided in Appendix D.5.

Remark 6.9.5. *The triangle inequality does not hold for KL-divergence, not even with a constant factor loss, as the following example shows. For sufficiently small ϵ, let $A = Ber(\epsilon)$, $B = Ber(\epsilon^2)$ and $C = Ber(\epsilon^{1/\epsilon})$. One can verify that $D_{KL}(A\|B) \le \epsilon$, $D_{KL}(B\|C) \le \epsilon$, but $D_{KL}(A\|C) \ge \omega(\epsilon)$. Nevertheless, we provide a restricted version of triangle inequality in Lemma 6.9.7 that forms the basis of our analysis.*

Definition 6.9.6 (θ-padding of a Bernoulli random variable)**.** *We define the padding operation as follows. Given a Bernoulli random variable X and a threshold $\theta \in (0,1)$ we let*

$$\textsc{Padding}(X,\theta) \equiv \begin{cases} X & \text{if } \mathbb{E}[X] \in [\theta, 1-\theta] \\ Ber(\theta) & \text{if } \mathbb{E}[X] < \theta \\ Ber(1-\theta) & \text{if } \mathbb{E}[X] > 1-\theta \end{cases}$$

Lemma 6.9.7 (Triangle inequality for padded KL-divergence)**.** *Suppose $p,q,r \in [0,1]$ and consider the KL-divergence between Bernoulli variables $Ber(p)$, $Ber(q)$ and $Ber(r)$. Then for some absolute constant C, any $\epsilon \in [0,1/32]$, if $D_{KL}\left(Ber(p)\|Ber(q)\right) \le \epsilon$ and $D_{KL}\left(Ber(q)\|Ber(r)\right) \le \epsilon$, then*

$$D_{KL}\left(Ber(p)\|Ber(\textsc{Padding}(r,\epsilon))\right) \le C\epsilon.$$

The proof is provided in Appendix D.5.

Remark 6.9.8. *We note that the padding is crucial, due to the example in Remark 6.9.5.*

6.9.3 Padding and total variation distance

In this and the next section we compare the behavior of the $\textsc{Level-}j\textsc{-Test}$'s on the iid and permutation streams. Recall the definition of $\textsc{Level-}j+1\textsc{-Test}$ from Algorithm 17 in Section 6.3. We call this the T^{IID}_{j+1} test and restate it here for completeness:

Algorithm 24 Vertex test in the i.i.d. stream

```
1: procedure $T_{j+1}^{\mathrm{IID}}(v)$
2:     $S \leftarrow 0$
3:     for $k = 1$ to $c^j \cdot \frac{m}{n}$ do
4:         $e \leftarrow$ next edge in the iid stream          ▷ Equivalent to sampling and iid edge
5:         if $e$ is adjacent to $v$ then
6:             $w \leftarrow$ the other endpoint of $e$
7:             $i \leftarrow 0$
8:             while $i \le j$ and $T_i^{\mathrm{IID}}(w)$ do
9:                 $S \leftarrow S + c^{i-j}$
10:                if $S \ge \delta$ then
11:                    return FALSE
12:                $i \leftarrow i + 1$
```

Similarly, define the version of the T_{j+1} tests on the permutation stream, starting at position t, which we call $T_{j+1}^{\pi,t}$ as follows:

Algorithm 25 Vertex test in the permutation stream

```
1: procedure $T_{j+1}^{\pi,t}(v)$
2:     $S \leftarrow 0$
3:     for $k = 1$ to $c^j \cdot \frac{m}{n}$ do
4:         $e \leftarrow$ next edge in the permutation stream     ▷ We start using the stream from the $t + 1^{\text{th}}$ edge
5:         if $e$ is adjacent to $v$ then
6:             $w \leftarrow$ the other endpoint of $e$
7:             $i \leftarrow 0$
8:             while $i \le j$ and $T_i^{\pi}(w)$ do
9:                 $S \leftarrow S + c^{i-j}$
10:                if $S \ge \delta$ then
11:                    return FALSE
12:                $i \leftarrow i + 1$
13:    return True
```

We also define recursively padded versions of T_{j+1}^{IID} define recursively

Algorithm 26 Padded T_1 test

```
1: procedure $\widetilde{T}_1^{\mathrm{IID}}(v)$
2:     return $T_1^{\mathrm{IID}}(v)$.
```

Algorithm 27 Recursively padded T_{j+1} tests

```
1: procedure $\overline{T}^{\mathrm{IID}}_{j+1}(v)$
2:     $S \leftarrow 0$
3:     for $c^j m/n$ edges $e$ of the iid stream do
4:         if $e$ is adjacent to $v$ then
5:             $w \leftarrow$ the other endpoint of $e$.
6:             $i \leftarrow 0$
7:             while $i \le j$ and $\widetilde{T}^{\mathrm{IID}}_i(w)$ do
8:                 $S \leftarrow S + c^{i-j}$
9:                 if $S \ge \delta$ then
10:                    return FALSE
11:                $i \leftarrow i+1$
12:    return True
```

and, using Definition 6.9.6,

Algorithm 28 Recursively padded T_{j+1} tests

```
1: procedure $\widetilde{T}^{\mathrm{IID}}_{j+1}(v)$
2:     return PADDING($\overline{T}^{\mathrm{IID}}_{j+1}(v)$, $\frac{200c^j \log^2 n}{n}$)
```

Note that these alternate tests are not implementable and merely serve as a tool to proving that the T^{π} tests work similarly to the T^{IID} tests.

We begin by proving that padding the T^{IID} tests, even recursively, only changes the output with probability proportional to the fraction of the stream (of length m) consumed by the test.

Lemma 6.9.9. *For sufficiently small $\delta > 0$ and large enough c the following holds. For all $v \in V$ and $j = [0, J]$ one has*

$$\left\| T^{IID}_{j+1}(v) - \widetilde{T}^{IID}_{j+1}(v) \right\|_{TV} \le \frac{400 \cdot c^j \log^2 n}{n}.$$

Proof. The proof follows by triangle inequality of total variation distance: We will prove the following bounds:

$$\left\| T^{\mathrm{IID}}_{j+1}(v) - \overline{T}^{\mathrm{IID}}_{j+1}(v) \right\|_{\mathrm{TV}} \le \frac{200 \cdot c^j \log^2 n}{n} \tag{6.31}$$

$$\left\| \overline{T}^{\mathrm{IID}}_{j+1}(v) - \widetilde{T}^{\mathrm{IID}}_{j+1}(v) \right\|_{\mathrm{TV}} \le \frac{200 \cdot c^j \log^2 n}{n}. \tag{6.32}$$

Eq. (6.32) follows easily from definitions, since $\widetilde{T}^{\mathrm{IID}}_{j+1}(v)$ is $200c^j \log^2(n)/n$-padded version of $\overline{T}^{\mathrm{IID}}_{j+1}(v)$. We proceed to proving Eq. (6.31).

Consider the following 0,1 vector $W_{j+1}(v)$ describing the process of a $T_{j+1}(v)$ test: The first $c^j m/n$

coordinates denote the search phase; specifically we write a 1 if a neighbor was found and a 0 if not. The following coordinates denote the outcomes of the recursive tests, 1 for pass 0 for fail, in the order they were performed. There could be at most c^j recursive tests performed (if they were all T_1's) so $W_{j+1}(v)$ has length $c^j m/n + c^j$ in total. However, often much fewer recursive tests are performed due to the early stopping rule, or simply because too few neighbors of v were found. In this case, tests not performed are represented by a 0 in $W_{j+1}(v)$.

Like with $T_{j+1}(v)$, we will define different versions of $W_{j+1}(v)$, namely $W_{j+1}^{\mathrm{IID}}(v)$ and $\overline{W}_{j+1}^{\mathrm{IID}}(v)$. Since $W_{j+1}(v)$ determines $T_{j+1}(v)$ we can simply bound $\left\|W_{j+1}^{\mathrm{IID}}(v)-\overline{W}_{j+1}^{\mathrm{IID}}(v)\right\|_{\mathrm{TV}}$.

The first $c^j m/n$ coordinates of W^{IID} and $\overline{W}^{\mathrm{IID}}$ are distributed identically and contribute nothing to the divergence. Consider the test corresponding to the i^{th} coordinate of the recursive phase of $W_{j+1}(v)$. Let the level of the test be $\ell_i \in [0,j]$ (with 0 representing no test) and let the vertex of the test be u_i. These are random variables determined by Π. Furthermore, let t_i be the position in the stream where the recursive T_{ℓ_i} test is called on u_i. Then we have

$$\begin{aligned}\left\|T_{j+1}^{\mathrm{IID}}(v)-\overline{T}_{j+1}^{\mathrm{IID}}(v)\right\|_{\mathrm{TV}} &\le \left\|W_{j+1}^{\mathrm{IID}}(v)-\overline{W}_{j+1}^{\mathrm{IID}}(v)\right\|_{\mathrm{TV}}\\ &=\sum_i \mathbb{E}_{\ell_i,u_i}\left\|\left(W_{j+1,i}^{\mathrm{IID}}(v)\middle|\ell_i,u_i\right)-\left(\overline{W}_{j+1,i}^{\mathrm{IID}}(v)\middle|\ell_i,u_i\right)\right\|_{\mathrm{TV}}\\ &=\sum_i \mathbb{E}_{\ell_i,u_i}\left\|T_{\ell_i}^{\mathrm{IID}}(u_i)-\widetilde{T}_{\ell_i}^{\mathrm{IID}}(u_i)\right\|_{\mathrm{TV}}\\ &\le \sum_i \mathbb{E}_{\ell_i}\left(\frac{400c^{\ell_i-1}\log^2 n}{n}\right),\end{aligned}$$

by the inductive hypothesis. Here $\left(W_{j+1,i}(v)\middle|\ell_i,u_i\right)$ denotes conditional distribution of $W_{j+1,i}(v)$ on ℓ_i and u_i.

Note that the term in the sum is proportional to the number of edges used by the corresponding recursive test. Indeed, recall by Lemma 6.4.1 that a T_{ℓ_i} test runs for at most $c^{\ell_i-1}\cdot\frac{m}{n}\cdot(1+2\delta)$ samples with probability one. It is crucial that this result holds with probability one, and therefore extends to not just the iid stream it was originally proven on, but any arbitrary stream of edges.

Therefore, the term in the sum above is at most $200C\cdot c\log^2(n)/m$ times the number of edges used by the corresponding recursive test. So the entire sum is at most $400\log^2(n)/m$ times the length of the recursive phase of the original T_{j+1} test. It was also derived in Lemma 6.4.1 that the recursive phase itself takes at most $c^j\cdot\frac{m}{n}\cdot 2\delta$ edges. (Again, the result holds with probability one.) Therefore,

$$\left\|T_{j+1}^{\mathrm{IID}}(v)-\widetilde{T}_{j+1}^{\mathrm{IID}}(v)\right\|_{\mathrm{TV}}\le\frac{400\log^2 n}{m}\cdot\frac{2\delta c^j m}{n}\le\frac{400\cdot c^j\log^2 n}{n},$$

for small enough absolute constant δ. This shows Eq. (6.31) and concludes the proof.

□

6.9.4 Bounding KL-divergence

In this section we will bound the KL-divergence between tests on the permutation stream and padded tests on the iid stream. We will use the padded triangle inequality of Lemma 6.9.7 as well as the data processing inequality for kl-divergence:

Lemma 6.9.10. (Data Processing Inequality) *For any random variables X, Y and any function f one has* $D_{KL}(f(X)||f(Y)) \le D(X||Y)$.

Lemma 6.9.11. *For sufficiently small $\delta > 0$ and large enough c the following holds. With high probability over Π, for all $v \in V$, $j \le J$, $t \le m/2 - 2c^j \cdot \frac{m}{n}$,*

$$D_{KL}\left(T^{\pi,t}_{j+1}(v) \middle\| \widetilde{T}^{IID}_{j+1}(v) \middle| G^t\right) \le \frac{200C \cdot c^j \log^2 n}{n}, \tag{6.33}$$

where C is the constant from Lemma 6.9.7.

Note that the choice of t is such that we guarantee the T_{j+1} tests finishing before half of the stream is up, by Lemma 6.4.1.

Naturally, we will prove the above lemma by induction on j. Specifically, our inductive hypothesis will be that Eq. (6.33) holds up to some threshold j. Furthermore, recall that Algorithm 28 satisfies

$$\widetilde{T}^{\text{IID}}_{j+1}(v) = \text{PADDING}\left(\overline{T}^{\text{IID}}_{j+1}(v), \frac{200 \cdot c^j \log^2 n}{n}\right).$$

Thus, to bound $D_{KL}\left(T^{\pi,t}_{j+1}(v) \middle\| \widetilde{T}^{\text{IID}}_{j+1}(v) \middle| G^t\right)$ we can apply Lemma 6.9.7 directly. By setting $\text{Ber}(p)$ to $\left(T^{\pi,t}_{j+1}(v) \middle| G^t\right)$ and $\text{Ber}(r)$ to $\overline{T}^{\text{IID}}_{j+1}(v)$ we get that $\widetilde{T}^{\text{IID}}_{j+1}(v) \sim \text{Ber}(\widetilde{r})$. So in order to bound $D_{KL}(\text{Ber}(p) \| \text{Ber}(\widetilde{r}))$ as required by the lemma, we need only to bound $D_{KL}(\text{Ber}(p) \| \text{Ber}(q))$ and $D_{KL}(\text{Ber}(q) \| \text{Ber}(r))$.

Our q, the midpoint of our triangle inequality, will be the outcome of a newly defined hybrid test using both the permutation and iid streams.

```
1: procedure $T^{\chi,t}_{j+1}(v)$
2:     $S \leftarrow 0$
3:     for $k = 1$ to $c^j \cdot \frac{m}{n}$ do
4:         $e \leftarrow$ next edge in the permutation stream      ▷ We start using the stream from the $t+1$th edge
5:         if $e$ is adjacent to $v$ then
6:             $w \leftarrow$ the other endpoint of $e$
7:             $i \leftarrow 0$                                    ▷ Represents the last level that $w$ passes
8:             while $i \le j$ and $\widetilde{T}^{\text{IID}}_i(w)$ do
9:                 $S \leftarrow S + c^{i-j}$
10:                if $S \ge \delta$ then
11:                    return FALSE
12:                $i \leftarrow i + 1$
13:    return True
```

We will begin by bounding $D_{KL}\left(T^{\chi,t}_{j+1}(v) \middle\| \overline{T}^{\text{IID}}_{j+1}(v) \middle| G^t\right)$

Lemma 6.9.12. *For sufficiently small $\delta > 0$ and large enough c the following holds. With probability $1 - n^{-5}$, for all, $v \in V$, $j \le J$, $t \le m/2 - 2c^j \cdot \frac{m}{n}$*

$$D_{KL}\left(T^{\chi,t}_{j+1}(v) \middle\| \overline{T}^{IID}_{j+1}(v) \middle| G^t\right) \le \frac{200 \cdot c^j \cdot \log^2 n}{n}. \tag{6.34}$$

Proof. We will deconstruct $T_{j+1}(v)$ a little differently than before, in the proof of Lemma 6.9.9. Consider the following integer vector, $Y_{j+1}(v)$, describing the process of $T_{j+1}(v)$: There are $c^j m/n$ coordinates in total, with the ith coordinate corresponding to the ith edge sampled from the stream. If this edge is not adjacent to v the coordinate is 0. If it is adjacent, with its other endpoint being u_i, the algorithm performs higher and higher level tests on u_i until it fails or the level exceeds j. So let the ith coordinate of $Y_{j+1}(v)$ simply denote the level ℓ_i at which $T_{\ell_i}(u_i)$ failed, or $j+1$ if the vertex never failed.

Like with $T_{j+1}(v)$, we will define different versions of $Y_{j+1}(v)$, namely $Y^{\chi,t}_{j+1}(v)$ and $\overline{Y}^{\mathrm{IID}}_{j+1}(v)$. Note that $Y_{j+1}(v)$ determines $T_{j+1}(v)$ and it suffices to bound $D_{KL}\left(Y^{\chi}_{j+1}(v) \middle\| \overline{Y}^{\mathrm{IID}}_{j+1}(v) \middle| G^t\right)$ by the data processing inequality (Lemma 6.9.10).

$$\begin{aligned} D_{KL}\left(T^{\chi,t}_{j+1}(v) \middle\| \overline{T}^{\mathrm{IID}}_{j+1}(v) \middle| G^t\right) &\le D_{KL}\left(Y^{\chi,t}_{j+1}(v) \middle\| \overline{Y}^{\mathrm{IID}}_{j+1}(v) \middle| G^t\right) \\ &= \sum_i \mathbb{E}_{G^{t_i}} D_{KL}\left(Y^{\chi,t}_{j+1,i}(v) \middle\| \overline{Y}^{\mathrm{IID}}_{j+1,i}(v) \middle| G^{t_i}\right), \end{aligned}$$

where G^{t_i} represents the residual graph right after we sample the ith edge for the T_{j+1} test (that is ith excluding edges sampled during recursion). Consider the distribution of $Y^{\chi,t}_{j+1,i}(v)$ and $\overline{Y}^{\mathrm{IID}}_{j+1,i}(v)$. In both cases the recursive tests run would use the iid stream. Consider for each neighbor of v running all iid tests $\widetilde{T}^{\mathrm{IID}}_{\ell}$ for ℓ from 1 to j.

Let $\psi : E \to [0, j+1]$ be the following random mapping: If e is not adjacent on v then $\psi(e) = 0$. If $e = (u, v)$, $\psi(e)$ is distributed as the smallest level ℓ at which u would fail when running $\overline{T}^{\mathrm{IID}}_1(u), \ldots, \overline{T}^{\mathrm{IID}}_j(u)$, and $j+1$ if it never fails. As described above, $\overline{Y}^{\mathrm{IID}}_{j+1,i}(v)$ is distributed as $\psi(e) : e \sim \mathrm{U}(E)$ and $Y^{\chi}_{j+1,i}(v)$ is distributed as $\psi(e) : e \sim \mathrm{U}(E^{t_i})$. Here U represents the uniform distribution and E^{t_i} represents the residual edge-set.

$$\mathbb{E}_{G^{t_i}} D_{KL}\left(Y^{\chi}_{j+1,i}(v) \middle\| \overline{Y}^{\mathrm{IID}}_{j+1,i}(v) \middle| G^{t_i}\right) = \mathbb{E}_{G^{t_i},\psi} D_{KL}\left(\psi(e) : e \sim \mathrm{U}(E^{t_i}) \middle\| \psi(e) : e \sim \mathrm{U}(E) \middle| G^{t_i}\right)$$

We will now bound the right hand side with high probability over G^{t_i}. Fix v and j, thereby fixing the distribution of ψ. Define for every $k \in [0, j+1]$

$$\begin{aligned} p^{\mathrm{IID}}_k(v) &:= \mathbb{P}_{e \sim \mathrm{U}(E)}[\psi(e) = k] \\ p^{\chi,t_i}_k(v) &:= \mathbb{P}_{e \sim \mathrm{U}(E^{t_i})}[\psi(e) = k]. \end{aligned} \tag{6.35}$$

We know by Chernoff bound that for every fixed choice of $v \in V$, $k \in [0, j+1]$ and $t_i \le m/2 - 2c^{k-1} \cdot \frac{m}{n}$

$$|p_k^{\mathrm{IID}}(v) - p_k^{\chi,t_i}(v)| \le \sqrt{\frac{100 p_k \log n}{m}} \tag{6.36}$$

with probability at least $1 - n^{-10}$ over the randomness of Π. Indeed

$$p_k^{\chi,t_i}(v) = \sum_{e \in E} \frac{\mathbb{1}(e \in G^{t_i}) \cdot \mathbb{P}\left(\psi(e) = k\right)}{m - t_i},$$

so $\mathbb{E} p_k^{\chi,t_i}(v) = p_k^{\mathrm{IID}}(v)$ and p_k^{χ,t_i} is a sum of independent variables bounded by $2/m$. Therefore, by Chernoff bounds, specifically Items (a) and (c) of Theorem 6.5.6,

$$\mathbb{P}\left[|p_k^{\chi,t_i}(v) - p_k^{\mathrm{IID}}(v)| \ge \sqrt{\frac{100 p_k^{\mathrm{IID}} \log n}{m}}\right] \le 2\exp\left(\frac{100 p_k^{\mathrm{IID}} \log n}{3 m p_k^{\mathrm{IID}} \cdot (2/m)}\right) \le n^{-10}.$$

Therefore, with probability at least $1 - n^{-5}$ this bound holds for all choices of v, k and t_i simultaneously. Let us restrict our analysis to this event from now on.

In the following calculation we denote $p_k^{\chi,t_i}(v)$ by $\widetilde{p}_k$ and $p_k^{\mathrm{IID}}(v)$ by p_k for simplicity of notation. We have

$$D_{KL}\left(\psi(e) : e \sim \mathrm{U}(E^{t_i}) \middle\| \psi(e) : e \sim \mathrm{U}(E) \middle| \psi, G^{t_i}\right) = \sum_{k=0}^{j+1} -\widetilde{p}_k \log\left(\frac{p_k}{\widetilde{p}_k}\right) = \sum_{k=0}^{j+1} -\widetilde{p}_k \log\left(1 + \frac{p_k - \widetilde{p}_k}{\widetilde{p}_k}\right).$$

Note that because $t_i \le m/2$, $\widetilde{p} \in [0, 2p_k]$, so $(p_k - \widetilde{p}_k)/\widetilde{p}_k$ is in the range $[-1/2, \infty]$. In the range $x \in [-1/2, \infty]$, $\log(1+x)$ can be lower bounded by $x - x^2$. Therefore, the above sum can be upper bounded as follows:

$$\begin{aligned}
D_{KL}\left(\psi(e) : e \sim \mathrm{U}(E^{t_i}) \middle\| \psi(e) : e \sim \mathrm{U}(E) \middle| \psi, G^{t_i}\right) &\le \sum_{k=0}^{j+1} -\widetilde{p}_k \cdot \left(\frac{p_k - \widetilde{p}_k}{\widetilde{p}_k} - \frac{(p_k - \widetilde{p}_k)^2}{\widetilde{p}_k^2}\right) \\
&= \sum_{k=0}^{j+1} \left(\widetilde{p}_k - p_k + \frac{(p_k - \widetilde{p}_k)^2}{\widetilde{p}_k}\right) \\
&\le \sum_{k=0}^{j+1} \frac{\left(\sqrt{100 \widetilde{p}_k \log n / m}\right)^2}{\widetilde{p}_k} && \text{due to Eq. (6.36),} \\
&\le \sum_{k=0}^{j+1} \frac{100 \log n}{m} \\
&\le \frac{200 \log^2 n}{m}.
\end{aligned}$$

With this we can bound the whole sum, and thus $D_{KL}\left(T_{j+1}^{\chi}(v) \middle\| \overline{T}_{j+1}^{\mathrm{IID}}(v)\right)$.

$$D_{KL}\left(T^{\chi}_{j+1}(v)\middle\|\overline{T}^{\mathrm{IID}}_{j+1}(v)\right) = \frac{c^j m}{n}\cdot\frac{200\log^2 n}{m}$$
$$= \frac{200\cdot c^j\log^2 n}{n},$$

as claimed.

□

We will now proceed to bound the divergence between $T_1^{\pi,t}(v)$ and $T_1^{\mathrm{IID}}(v)$. This will serve as the base case of the induction in the proof of Lemma 6.9.11.

Lemma 6.9.13. *For sufficiently small $\delta > 0$ and large enough c the following holds. With probability $1 - n^{-5}$, for all $v \in V$, $t \le m/2 - 2m/n$*

$$D_{KL}\left(T_1^{\pi,t}(v)\middle\| T_1^{IID}(v)\middle| G^t\right) \le \frac{200C\cdot\log^2 n}{n}, \tag{6.37}$$

where C is the constant from Lemma 6.9.7.

Proof. We will deconstruct the tests similarly to the previous proof. For both types of $T_1(v)$ test consider the random boolean vector $Y \in \{0,1\}^{m/n}$ denoting whether the next m/n edges in the appropriate stream are adjacent to v or not. That is $Y_i = 1$ if and only if the ith edge in the appropriate stream is adjacent to v. Let Y^π and Y^{IID} denote such random variables for $T_1^{\pi,t}(v)$ and $T_1^{\mathrm{IID}}(v)$ respectively.

Because Y determines the outcome of $T_1(v)$, by data processing inequality (Lemma 6.9.10) one has

$$D_{KL}\left(T_1^{\pi,t}(v)\middle\| T_1^{\mathrm{IID}}(v)\middle| G^t\right) \le D_{KL}\left(Y^\pi\middle\| Y^{\mathrm{IID}}\middle| G^t\right). \tag{6.38}$$

$$\begin{aligned} D_{KL}\left(Y^\pi\middle\| Y^{\mathrm{IID}}\middle| G^t\right) &= \sum_{i=1}^{m/n} D_{KL}\left(Y_i^\pi\middle\| Y_i^{\mathrm{IID}}\middle| G^t, Y_1^\pi,\dots,Y_{i-1}^\pi\right)\\ &\le \sum_{i=1}^{m/n} D_{KL}\left(Y_i^\pi\middle\| Y_i^{\mathrm{IID}}\middle| G^{t+i-1}\right)\\ &= \sum_{i=1}^{m/n} D_{KL}\left(\mathrm{Ber}\left(d^{t+i-1}(v)/m\right)\middle\|\mathrm{Ber}\left(d(v)/m\right)\right). \end{aligned} \tag{6.39}$$

Recall that $d^t(v)$ is the residual degree of v at time t. By Lemma 6.9.4 one has

$$D_{KL}\left(\mathrm{Ber}\left(d^{t+i-1}(v)/m\right)\middle\|\mathrm{Ber}\left(d(v)/m\right)\middle| G^{t+i-1}\right) \le \frac{16\left(\frac{d^{t+i-1}(v)}{m} - \frac{d(v)}{m}\right)^2}{\frac{d(v)}{m}\left(1-\frac{d(v)}{m}\right)}, \tag{6.40}$$

and hence it suffices to upper bound the squared deviation of the residual degree $d^{t+i-1}(v)$ from $d(v)$. Note that we have $\mathbb{E}[d^t(v)] = d(v)$ for every v and t, and by Chernoff bounds the residual degree concentrates around its expectation. More specifically:

$$\mathbb{P}\left(|d^t(v) - d(v)| \geq \sqrt{100d(v)\log(n)}\right) \leq n^{-10}. \tag{6.41}$$

Let us constrain ourselves to the event of probability at least $1 - n^{-5}$ where this bound is satisfied for every t and v. That is one has $|d^t(v) - d(v)| \leq \sqrt{100d(v)\log(n)}$.

We thus have, using Eqs. (6.40) and (6.41) together with the fact that $1 \leq d(v) \leq m/3$

$$\begin{aligned} D_{KL}\left(\mathrm{Ber}\left(d^{t+i-1}(v)/m\right) \middle\| \mathrm{Ber}\left(d(v)/m\right) \middle| G^{t+i-1}\right) &\leq \frac{16\left(\sqrt{100d(v)\log n}\right)^2}{m\cdot d(v)\left(1-\frac{d(v)}{m}\right)} \\ &\leq \frac{1600\cdot d(v)\log n}{m\cdot d(v)\cdot 2/3} \\ &= \frac{2400\log n}{m}. \end{aligned} \tag{6.42}$$

(Note that we assumed $d(v) \leq n \leq m/3$.)

Therefore,

$$D_{KL}\left(Y^{\pi} \| Y^{\mathrm{IID}} | G^t\right) \leq \frac{2400\log n}{m}\cdot\frac{m}{n} = \frac{2400\log n}{n},$$

which is stronger than the desired bound of Eq. (6.37).

□

We are finally ready to prove Lemma 6.9.11.

Proof of Lemma 6.9.11. First, let us constrain ourselves to the high probability event where Eqs. (6.34) and (6.37) hold from Lemmas 6.9.12 and 6.9.13. As mentioned before, we will proceed by induction on j. Our base case is simply Lemma 6.9.13 with a slight modification. Indeed, Lemma 6.9.13 bounds the divergence of $T_1^{\pi,t}(v)$ and $T_1^{\mathrm{IID}}(v)$, whereas we need to bound the divergence of $T_1^{\pi,t}(v)$ and $\widetilde{T}_1^{\mathrm{IID}}(v)$. Note however, that $\widetilde{T}_1^{\mathrm{IID}}(v)$ is simply the padding of $\overline{T}_1^{\mathrm{IID}}(v)$ by $200\log^2(n)/n$, which is itself identical to $T_1^{\mathrm{IID}}(v)$ by definition.

If $\overline{T}_1^{\mathrm{IID}}(v)$ is not close to deterministic, the padding does nothing and the base case holds. If $T_1^{\mathrm{IID}}(v)$ is close enough to deterministic that it gets padded, the divergence from $T_1^{\pi,t}(v)$ either decreases or increases to at most $D_{KL}(\mathrm{Ber}(0) \| \mathrm{Ber}(\epsilon))$, where $\epsilon = 200\log^2(n)/n$ is the padding parameter. In this case

$$\begin{aligned} D_{KL}\left(T_1^{\pi,t}(v) \| \widetilde{T}_1^{\mathrm{IID}}(v) | G^t\right) &\leq D_{KL}\left(\mathrm{Ber}(0) \middle\| \mathrm{Ber}\left(\frac{200\log^2 n}{n}\right)\right) \\ &= -\log\left(1 - \frac{200\log^2 n}{n}\right) \end{aligned}$$

$$\le \frac{400 \log^2 n}{n},$$

which suffices. (See Fact D.5.3 from the proof of Lemma 6.9.7 in Appendix D.5.)

For, the inductive step, we will use the triangle inequality of Lemma 6.9.7, pivoting on $T^{\chi,t}_{j+1}(v)$. Specifically, we invoke Lemma 6.9.7 with

$$p = \mathbb{P}\left(T^{\pi,t}_{j+1}(v) \text{ succeeds}\right)$$
$$q = \mathbb{P}\left(T^{\chi,t}_{j+1}(v) \text{ succeeds}\right)$$
$$r = \mathbb{P}\left(\overline{T}^{\text{IID}}_{j+1}(v) \text{ succeeds}\right).$$

and $\epsilon = 200c^j \log^2 n/n$. Since the ϵ-padding of r, denoted by $\widetilde{r} = \text{PADDING}(r,\epsilon)$ is exactly $\mathbb{P}\left(\widetilde{T}^{\pi,t}_{j+1}(v) \text{ succeeds}\right)$, by Lemma 6.9.7 it suffices to prove

$$D_{KL}\left(T^{\pi,t}_{j+1}(v) \middle\| T^{\chi,t}_{j+1}(v) \middle| G^t\right) \le \frac{200c^j \log^2 n}{n} \tag{6.43}$$

and

$$D_{KL}\left(T^{\chi,t}_{j+1}(v) \middle\| \overline{T}^{\text{IID}}(v) \middle| G^t\right) \le \frac{200c^j \log^2 n}{n}. \tag{6.44}$$

Eq. (6.44) holds by Lemma 6.9.12.

Comparing $T^{\pi,t}_{j+1}(v)$ and $T^{\chi,t}_{j+1}(v)$ (establishing Eq. (6.43)): We will deconstruct T_{j+1} identically to what was done in the proof of Lemma 6.9.9, and we will use techniques for bounding the divergence similar to the ones used in the proof of Lemma 6.9.9 for bounding the total variation distance. Recall the 0, 1 vector $W_{j+1}(v)$ describing the process of a $T_{j+1}(v)$ test: The first $c^j m/n$ coordinates denote the search phase; specifically we write a 1 if a neighbor was found and a 0 if not. The following coordinates denote the outcomes of the recursive tests, 1 for pass 0 for fail, in the order they were performed. There could be at most c^j recursive tests performed (if they were all T_1's) so $W_{j+1}(v)$ has length $c^j m/n + c^j$ in total. However, often much fewer recursive tests are performed due to the early stopping rule, or simply because too few neighbors of v were found. In this case, tests not performed are represented by a 0 in $W_{j+1}(v)$.

Since $W_{j+1}(v)$ determines $T_{j+1}(v)$ we can simply bound $D_{KL}\left(W^{\pi,t}_{j+1}(v) \middle\| W^{\chi,t}_{j+1}(v) \middle| G^t\right)$ by the data processing inequality of Lemma 6.9.10.

The first $c^j m/n$ coordinates of W^π and W^χ are distributed identically and contribute nothing to the divergence. Consider the test corresponding to the i^{th} coordinate of the recursive phase of $W_{j+1}(v)$. Let the level of the test be $\ell_i \in [0, j]$ (with 0 representing no test) and let the vertex of the test be u_i. These are random variable determined by Π. Furthermore, let t_i be the position in the stream where the recursive T_{ℓ_i} test is called on u_i. Then we have

$$\begin{aligned} D_{KL}\left(T^{\pi,t}_{j+1}(v) \middle\| T^{\chi,t}_{j+1}(v) \middle| G^t\right) &\le D_{KL}\left(W^{\pi,t}_{j+1}(v) \middle\| W^{\chi,t}_{j+1}(v) \middle| G^t\right) \\ &= \sum_i D_{KL}\left(W^{\pi,t}_{j+1,i}(v) \middle\| W^{\chi,t}_{j+1,i}(v) \middle| G^t, W^{\pi,t}_{j+1,1}(v), \ldots, W^{\pi,t}_{j+1,i-1}(v)\right) \end{aligned}$$

$$\begin{aligned}
&\le \sum_i \mathbb{E}_{\ell_i,u_i} D_{KL}\left(W^{\pi,t}_{j+1,i}(v) \middle\| W^{\chi,t}_{j+1,i}(v) \middle| G^{t_i}, \ell_i, u_i\right) \\
&= \sum_i \mathbb{E}_{\ell_i,u_i} D_{KL}\left(T^{\pi,t_i}_{\ell_i}(u_i) \middle\| \widetilde{T}^{\mathrm{IID}}_{\ell_i}(u_i) \middle| G^{t_i}\right) \\
&\le \sum_i \mathbb{E}_{\ell_i}\left(\frac{200C \cdot c^{\ell_i - 1}\log^2 n}{n}\right),
\end{aligned}$$

by the inductive hypothesis. Here in the third line we condition on ℓ_i and u_i, thereby only increasing the divergence.

Note that the term in the sum is proportional to the number of edges used by the corresponding recursive test. Indeed, recall by Lemma 6.4.1 that a T_{ℓ_i} test runs for at most $c^{\ell_i - 1} \cdot \frac{m}{n} \cdot (1+2\delta)$ samples with probability one. It is crucial that this result holds with probability one, and therefore extends to not just the iid stream it was originally proven on, but any arbitrary stream of edges.

Therefore, the term in the sum above is at most $200C\log^2(n)/m$ times the number of edges used by the corresponding recursive test. So the entire sum is at most $200C\log^2(n)/m$ times the length of the recursive phase of the original T_{j+1} test. It was also derived in Lemma 6.4.1 that the recursive phase itself takes at most $c^j \cdot \frac{m}{n} \cdot 2\delta$ edges. (Again, the result holds with probability one.) Therefore,

$$D_{KL}\left(T^{\pi,t}_{j+1}(v) \middle\| T^{\chi,t}_{j+1}(v) \middle| G^t\right) \le \frac{200C\log^2(n)}{m} \cdot \frac{\delta c^j m}{n} \le \frac{200 \cdot c^j \log^2 n}{n},$$

for small enough absolute constant δ. This shows Eq. (6.43) and concludes the proof.

□

6.9.5 The full algorithm

We proceed to derive an algorithm for approximating the matching size of a graph in a random permutation stream. The following well-known theorem will be useful for proving correctness:

Theorem 6.9.14 (Pinsker's Inequality). *For two distributions P and Q,*

$$\|P - Q\|_{TV} \le \sqrt{2\log_2 e \cdot D_{KL}(P\|Q)}.$$

Recall the EDGE-LEVEL-TEST from Algorithm 16 in Section 6.3. We will run a similar edge test on the permutation stream, with one crucial difference: Our original algorithm from Section 6.3 uses $\Theta(m)$ samples from the stream. Since our bound on the divergence between the iid and permutation variants of LEVEL-j-TEST scales with $\log^2 n$ times the sample complexity, it would grow past constant if we were to adapt our iid algorithm as is. Therefore, we will constrain the algorithm to only use a $\log^2 n$ fraction of the available stream. Specifically EDGE-LEVEL-TEST will only calculate the level of an edge up to $J - 2\log_c(\log n)$, as opposed to J. We will call this edge test $E^{\mathrm{IID}}(e)$.

Algorithm 29 Given an edge e, this algorithm returns a fractional matching-weight of e.

```
1: procedure $E^{\mathrm{IID}}(e = (u, v))$
2:     $w \leftarrow 1/n$
3:     for $i = 1$ to $J - 2\log_c(\log n)$ do          ▷ Recall that $J = \lfloor \log_c n \rfloor - 1$
4:         if LEVEL-$i$-TEST($u$) and LEVEL-$i$-TEST($v$) then
5:             $w \leftarrow w + c^i/n$
6:         else
7:             return $w$
8:     return $w$
```

As in the previous section we define the edge level test $E(e)$ for the permutation stream as well, namely $E^{\pi,t}(e)$. We also define $\widetilde{E}^{\mathrm{IID}}$ using the padded $\widetilde{T}_i^{\mathrm{IID}}$ tests in place of LEVEL-i-TEST.

Although the truncated edge test E^{IID} is not as powerful as the untruncated version, it is a $\Theta(\log^2 n)$-factor estimator for the matching number of the input graph, MM(G).

Corollary 6.9.15. *For all c large enough, there exists $\delta > 0$ such that the following holds. For all $G = (V, E)$ and an edge $e \in E$, let $E^{IID}(e)$ denote the value returned by Algorithm 29. Then,*

$$\sum_{e \in E} E^{IID}(e) = O(MM(G))$$
$$\sum_{e \in E} E^{IID}(e) = \Omega\left(\frac{MM(G)}{\log^2 n}\right).$$

Proof. Recall that Theorem 6.4.3 guarantees that

$$\sum_{e \in E} M_e = \Theta(\mathrm{MM}(G)),$$

where M_e is the is the output of the untruncated edge level test EDGE-LEVEL-TEST from Algorithm 16. Note that, by stopping $2\log_c(\log n)$ levels early, E^{IID} may misclassify edges by assigning them to levels up to $2\log_c(\log n)$ levels lower, but never misclassifies them by putting them higher. Therefore, $E^{\mathrm{IID}}(e)$ is no greater than M_e and at most $\Theta(\log^2 n)$ factor lower. □

Finally, recall Algorithm 15, our main algorithm for estimating the matching size.

Algorithm 30 Algorithm 15 estimating the value of MM(G).

```
procedure IID-PEELING($G = (V, E)$)
    $M' \leftarrow 0$
    $s \leftarrow 1$
    while samples lasts do
        $M' \leftarrow$ SAMPLE($G, s$)
        $s \leftarrow 2s$
    return $M'$

procedure SAMPLE($G = (V, E), s$)
    $M' \leftarrow 0$
    for $k = 1$ to $s$ do
        $e \leftarrow$ iid edge from the stream
        $M' \leftarrow M'$ + EDGE-LEVEL-TEST($e$)
    return $m \cdot \frac{M'}{s}$
```

Note that the variable denoting the number of edges sampled in SAMPLE, originally t, has been changed to s here so as to not be confused with the variable used to denote our position in the stream.

We will now describe the permutation variant, which can be used to approximate the maximum matching size to an $O(\log^2 n)$ factor, in a random permutation stream with $O(\log^2 n)$ bits of space.

Definition 6.9.16. *Let* PERMUTATION-PEELING$(G = (V, E))$ *be a variant of* IID-PEELING *that uses the permutation stream in Line 12 and the subroutine* $E^{\pi,t}(e)$ *in Line 13. Furthermore, it should only continue until a* $\Theta(1/\log^2 n)$ *fraction of the stream is exhausted, as opposed to Algorithm 15 which exhausts the entire* $\Theta(m)$ *sized stream in Line 4.*

We define further variants of Algorithm 30 that will be useful in the proof of correctness of PERMUTATION-PEELING.

Definition 6.9.17. *We define the following three variants of Algorithm 30:*

- TRUNCATED-IID-PEELING$(G = (V, E))$ *uses the iid stream in Line 12 and* E^{IID} *in Line 13.*
- HYBRID-PEELING$(G = (V, E))$ *uses the permutation stream in Line 12 and* E^{IID} *in Line 13.*
- PADDED-PEELING$(G = (V, E))$ *uses the permutation stream in Line 12 and* $\widetilde{E}^{IID}$ *in Line 13.*

All three algorithms terminate after exhausting a $\Theta(1/\log^2 n)$ *fraction of the stream, like* PERMUTATION-PEELING.

Theorem 6.9.18. *For sufficiently small* $\delta > 0$ *and large enough* c *the following holds. For any graph* $G = (V, E)$, PERMUTATION-PEELING *(Algorithm 30) with* $3/4$ *probability outputs a* $O(\log^2 n)$ *factor approximation of* $MM(G)$ *by using a single pass over a randomly permuted stream of edges.*

Proof. **Correctness of TRUNCATED-IID-PEELING.** We will first prove the same claim for TRUNCATED-IID-PEELING. Recall first the proof of Theorem 6.4.4 from Section 6.4. Let M_e denote the outcome of EDGE-LEVEL-TEST(e) and $\mu = \mathbb{E}_{e\sim U(E)}[M_e]$. Recall that the proof hinges on showing that μ is within a constant factor of MM(G). Also $M_e : e \sim U(E)$ has variance at most $2\mu/c$, so $\Theta(1/\mu c)$ independent edge tests suffice to be able to bound the deviation from the mean powerfully enough using Chebyshev's inequality. Furthermore, we know that in expectation, EDGE-LEVEL-TEST$(e) : e \sim U(E)$ takes samples proportional to the number $\mu \cdot m$ (see Lemma 6.4.2). Putting all this together we get that with constant probability the output of SAMPLE(G, s) is within a constant factor of MM(G) for large enough s, and such a call of SAMPLE fits into $O(m)$ edges from the stream.

We will have a very similar proof for the correctness of TRUNCATED-IID-PEELING. Indeed, consider the random variable $S = E^{\mathrm{IID}}(e) : e \sim U(E)$. Let $\mathbb{E}S = \overline{\mu}$. In Corollary 6.9.15 we have shown that $\overline{\mu}$ is a $\Theta(\log^2 n)$-factor approximation of MM(G). Furthermore, S is bounded by $\frac{2}{c\log^2 n}$, so its variance is at most $\frac{2\overline{\mu}}{c\log^2 n}$. By Chebyshev's inequality $s = \Theta(\frac{1}{c\overline{\mu}\log^2 n})$ edge tests suffice to get an accurate enough empirical mean. This many edge tests take $s\cdot\overline{\mu}\cdot m = \Theta\left(\frac{m}{c\log^2 n}\right)$ samples, as desired. If the algorithm doesn't terminate within $\frac{mR}{c\log^2 n}$ we consider it to have failed. This happens with probability less than $1/10$ for some large absolute constant R.

Correctness of HYBRID-PEELING. The only difference between TRUNCATED-IID-PEELING and HYBRID-PEELING is that the latter we use the permutation stream for sampling edges to run E^{IID} on. This guarantees no repetitions which only improves our variance bound and does not hurt the previous proof.

Comparing HYBRID-PEELING and PADDED-PEELING. Note that these two algorithms differ only in the type of vertex level tests they use: Specifically, HYBRID-PEELING uses T^{IID} while PADDED-PEELING uses $\widetilde{T}^{\mathrm{IID}}$. By Lemma 6.9.9 for all $v \in V$ and $j \in [0, J]$

$$\left\| T^{\mathrm{IID}}_{j+1}(v) - \widetilde{T}^{\mathrm{IID}}_{j+1}(v) \right\|_{\mathrm{TV}} \le \frac{400 \cdot c^j \log^2 n}{n}.$$

Note that this is proportional (with multiplicative factor $\frac{200\log^2 n}{m}$) to the sample complexity of the corresponding test. The output of the main peeling algorithm depends only on the outputs of vertex level tests called directly from edge level tests, which are disjoint (in the samples they use). Furthermore, the entire algorithm takes at most $\frac{mR}{c\log^2 n}$ samples, so the total variation distance between the outputs of HYBRID-PEELING and PADDED-PEELING is at most

$$\frac{200\log^2 n}{m} \cdot \frac{mR}{c\log^2 n} = \frac{200R}{c}.$$

For more details on this proof technique see the proofs of Lemmas 6.9.9 and 6.9.11.

Comparing PADDED-PEELING and PERMUTATION-PEELING. Note that these two algorithms again differ only in the type of vertex level test they use: Specifically, PADDED-PEELING uses $\widetilde{T}^{\mathrm{IID}}$ while PERMUTATION-PEELING uses $T^{\pi,t}$. By Lemma 6.9.11, with high probability, for all $v \in V$, $j \le J$, $t \le m/2 - 2c^j \cdot \frac{m}{n}$,

$$D_{KL}\left(T^{\pi,t}_{j+1}(v) \middle\| \widetilde{T}^{\mathrm{IID}}_{j+1}(v) \middle| G^t\right) \le \frac{200C \cdot c^j \log^2 n}{n}, \tag{6.45}$$

where C is the constant from Lemma 6.9.7. Note that this is proportional (with multiplicative factor $\frac{100C\log^2 n}{m}$) to the sample complexity of the corresponding test. Again we note that the output of the main peeling algorithm is a function of the outputs of the vertex level tests directly called from edge level tests, which are disjoint (in the samples they use). Furthermore, since the entire algorithm takes at most $\frac{mR}{c\log^2 n}$ samples, the divergence between the outputs of PADDED-PEELING and PERMUTATION-PEELING is at most

$$\frac{100\log^2 n}{m} \cdot \frac{mR}{c\log^2 n} = \frac{100R}{c}.$$

This translates to a total variation distance of at most $\sqrt{\frac{200R\log_2 e}{c}}$ by Pinsker's inequality. Again, for more details on this proof technique see the proofs of Lemmas 6.9.9 and 6.9.11.

In conclusion, PERMUTATION-PEELING returns a $\log^2 n$-factor approximation of $\mathrm{MM}(G)$ with probability at least $4/5 - \frac{200R}{c} - \sqrt{\frac{200R\log_2 e}{c}} \geq 3/4$ for large enough c. □

From here the proof of the main theorem follows.

Proof of Theorem 6.1.4. The proof follows from Theorem 6.9.18 and the fact that Algorithm 30 has recursion depth of $O(\log n)$, where each procedure in the recursion maintains $O(1)$ variables, hence requiring $O(\log n)$ bits of space. Therefore, the total memory is $O(\log^2 n)$. □

7 Conclusion

In this thesis, we have explored a range of algorithmic techniques for handling massive data (such as parallel processing, streaming processing, and data compression) in the context of graph algorithms. The focus was more on developing broadly applicable techniques, as opposed to specialized solutions to individual problems. Here, we summarize our contributions, and point out new directions and open problems, where, hopefully, our techniques and ideas will lead to further progress.

Spectral Graph Theory

In Chapter 2, we studied the construction of spectral sparsifiers from sketches; in particular, this allowed us to construct them in dynamic streams, where both edge insertions and deletions may appear. Our sketch and sparsifier are both nearly-optimally space-efficient, at $\widetilde{O}(n)$ memory, and our decoding algorithm takes a similarly nearly-optimal $\widetilde{O}(n)$ time. Then, in Chapter 3, we studied the extension of spectral sparsification to hypergraphs. We proved, for the first time, the existence of spectral hypergraph sparsifiers with a nearly linear number of hyperedges. With a corresponding lower, we showed that this is nearly-optimal, not only among sparsifiers, but among all possible compression of the hypergraph spectral structure. Finally, in Chapter 4, we proposed a fully-scalable MPC algorithm for simulating a large number of random walks of length ℓ from an arbitrary starting distribution, using $O(\log \ell)$ rounds of communication.

Although the result of Chapter 2 is nearly optimal by all metrics for the streaming setting, one interesting problem that remains open is to design a *fully dynamic* spectral sparsification algorithm. Much like in the setting of Chapter 2, dynamic algorithms process graphs through a stream of edge insertions and deletions. However, instead of just outputting a sparsifier at the end, a fully dynamic algorithm would be required to *maintain* a sprasifier of the current graph, throughout the process. Fully dynamic algorithms typically use only a polylogarithmic amount of time to process each update, making our decoding algorithm far too slow to transfer trivially to this setting.

Our hypergraph spectral sparsifier from Chapter 3 is *nearly*-optimal in size, with a gap of $\log^{O(1)} / \epsilon^{O(1)}$ between construction and lower-bound. However, this problem is far from closed; in ordinary graph spectral sparsification, a tremendous amount of research has gone into reducing this gap to only 2. In particular, Batson et al. (2012) shows the existence of ϵ-spectral sparsifiers of size $4n/\epsilon^2$. There is no reason to believe a similarly small sparsifier could not be achieved for hypergraphs. Interestingly, very different techniques than the ones in Chapter 3 would have to be used to achieve this, as all known importance sampling based methods lose at least a factor $\log n$ compared to the optimum, due to their reliance on randomness. Nevertheless, any attempt to extend such results to hypergraphs would face similar challenges to those we overcame in Chapter 3, namely, the adaptation of a proof which relies heavily on linear algebraic thinking, to the non-linear realm of the hypegraph energy function.

Maximum Matching

In Chapter 5, we proposed a new randomized composable coreset for maximum matching, with size at most $n-1$ and an approximation guarantee of $1/2$. In Chapter 6, we presented an extremely space-efficient implementation of a peeling algorithm for calculating an approximate maximum matching. This implementation, with some modifications, yielded new results in both uniformly random streams, where we were able to get a constant approximation to the maximum matching size in only $O(\log^2 n)$ space, and in the LCA model, where we were able to construct implicitly a constant-approximate maximum matching in $d\log^3 n$ space.

The technique we preseted in Chapter 6 is versatile, and may be generalizable to further models of computation, where calculating approximate maximum matchings is of interest. One open problem is the construction of large matchings in the MPC model. The current best known algorithms for constructing constant-approximate maximum matchings either require $O(\sqrt{\log n})$ rounds and are fully scalable Czumaj et al. (2018); Ghaffari and Uitto (2019), or require $(\log\log n)^{O(1)}$ rounds but nearly linear space Assadi et al. (2019a); Ghaffari et al. (2018). Achieving doubly logarithmic round complexity with a fully scalable algorithm remains open. (We also study matchings in the MPC model in Chapter 5; however, randomized composable coresets do not lead to fully scalable algorithms, as they typically require at least $\Omega(n)$ space to store.)

Finally, one of the most highly studied models of sublinear-space computation is adversarial order streaming (or simply streaming). Here, edges of the graph appear sequentially in a stream *in any order*, as opposed to a uniformly random order, in the model studied in Chapter 6. Our setting in Chapter 6 is very similar to this, but our algorithm crucially relies on the randomness of the order of the stream, and does not readily generalize to adversarially ordered streams. Surprisingly little is known about this setting: Neither a polynomial space ($n^{\Omega(1)}$) lower bound, nor a truly sublinear space ($n^{1-\Omega(1)}$) algorithm is known for approximating the maximum matching size to within a constant in a single pass.

A Supplementary Material for Chapter 2

We will need Lemma A.0.1 that we use in the correctness proof of our algorithm.

Lemma A.0.1 (Chain of Coarse Sparsifiers Li et al. (2013); Kapralov et al. (2017a))**.** *Consider any PSD matrix K with maximum eigenvalue bounded from above by $\lambda_u = 2n$ and minimum nonzero eigenvalue bounded from below by $\lambda_\ell = \frac{1}{8n^2}$. Let $d = \lceil \log_2 \frac{\lambda_u}{\lambda_\ell} \rceil$. For $\ell \in \{0, 1, 2, \ldots, d\}$, define:*

$$\gamma(\ell) = \frac{\lambda_u}{2^\ell}.$$

So $\gamma(d) \leq \lambda_\ell$, and $\gamma(0) = \lambda_u$. Then the chain of PSD matrices, $[K_0, K_1, \ldots, K_d]$ with $K_\ell = K + \gamma(\ell) I$ satisfies the following relations:

1. $K \preceq_r K_d \preceq_r 2 \cdot K$,
2. $K_\ell \preceq K_{\ell-1} \preceq 2 \cdot K_\ell$ *for all* $\ell \in \{1, \ldots, d\}$,
3. $K_0 \preceq 2 \cdot \gamma(0) \cdot I \preceq 2 \cdot K_0$.

We will need Theorem A.0.2 that we use in the proof of correctness of the main algorithm. It is well known that by sampling the edges of B according to their effective resistance, it is possible to obtain a weighted edge vertex incident matrix $\widetilde{B}$ such that $(1-\epsilon)B^\top B \preceq \widetilde{B}^\top \widetilde{B} \preceq (1+\epsilon)B^\top B$ with high probability (see Theorem A.0.2).

Theorem A.0.2 (Spectral Approximation via Effective Resistance Sampling Spielman and Srivastava (2011))**.** *Let $B \in \mathscr{R}^{\binom{n}{2} \times n}$, $K = B^\top B$, and let $\widetilde{\tau}$ be a vector of leverage score overestimates for B's rows, i.e. $\widetilde{\tau}_y \geq \mathbf{b}_y^\top K^+ \mathbf{b}_y$ for all $y \in [m]$. For $\epsilon \in (0,1)$ and fixed constant c, define the sampling probability for row $\mathbf{b}_y$ to be $p_y =$ $\min\{1, c \cdot \epsilon^{-2} \log n \cdot \widetilde{\tau}_y\}$. Define a diagonal sampling matrix W with $W(y,y) = \frac{1}{p_y}$ with probability p_y and $W(y,y) = 0$ otherwise. With high probability, $\widetilde{K} = B^\top W B \approx_\epsilon K$. Furthermore W has $O(||\widetilde{\tau}||_1 \cdot \epsilon^{-2} \log n)$ non-zeros with high probability.*

Lemma A.0.3 (ℓ_2 Heavy Hitters)**.** *For any $\eta > 0$, there is a decoding algorithm denoted by* HEAVYHITTER *and a distribution on matrices S^h in $\mathscr{R}^{O(\eta^{-2} \operatorname{polylog}(N)) \times N}$ such that, for any $x \in \mathscr{R}^N$, given $S^h x$, the algorithm* HEAVYHITTER$(S^h x, \eta)$ *returns a list $F \subseteq [N]$ such that $|F| = O(\eta^{-2} \operatorname{polylog}(N))$ with probability $1 - \frac{1}{\operatorname{poly}(()N)}$ over the choice of S^h one has*

(1) *for every $i \in [N]$ such that $|x_i| \geq \eta ||x||_2$ one has $i \in F$;*

(2) *for every $i \in F$ one has $|x_i| \geq (\eta/2)||x||_2$.*

The sketch $S^h x$ can be maintained and decoded in $O(\eta^{-2} \operatorname{polylog}(N))$ time and space.

Lemma A.0.4 (Binary Johnson-Lindenstrauss Lemma Achlioptas (2003))**.** *Let P be an arbitrary set of points in $\mathbb{R}^d$, represented by a $d \times n$ matrix A, such that the j^{th} point is $A\chi_j$. Given ϵ, $\beta > 0$ and*

$$q \geq \frac{4+2\beta}{\epsilon^2/2 - \epsilon^3/3} \log n.$$

Let Q be a random $q \times d$ matrix $(q_{ij})_{ij}$ where q_{ij}'s are independent identically distributed variables taking 1 and -1 each with probability $1/2$. Then, if $M = \frac{1}{\sqrt{q}} QA$, then with probability at least $1 - n^{-\beta}$, for all $u, v \in [n]$

$$(1-\epsilon)||A\chi_u - A\chi_v||_2^2 \leq ||M\chi_u - M\chi_v||_2^2 \leq (1+\epsilon)||A\chi_u - A\chi_v||_2^2$$

B Supplementary Material for Chapter 3

The following concentration bound is standard.

Theorem B.0.1 (Chernoff bound, see e.g. Alon and Spencer (2008))**.** *Let $X_1, \ldots, X_n$ be independent random variables in the range $[0, a]$. Let $\sum_{i=1}^n X_i = S$. Then for any $\delta \in [0,1]$ and $\mu \geq \mathbb{E}S$,*

$$\mathbb{P}[|S - \mathbb{E}S| \geq \delta\mu] \leq 2\exp\left(-\frac{\delta^2\mu}{3a}\right).$$

The following slight variation, allowing for both multiplicative and additive error, will be the most convenient for our purposes throughout Chapter 3.

Theorem B.0.2 (Additive-multiplicative Chernoff bounds Badanidiyuru and Vondrák (2013))**.** *Let $X_1, \ldots X_n$ be independent random variables in the range $[0, a]$. Let $\sum_{i=1}^n X_i = S$. Then for all $\delta \in [0,1]$ and $\alpha > 0$,*

$$\mathbb{P}[|S - \mathbb{E}S| \geq \delta\mathbb{E}S + \alpha] \leq 2\exp\left(-\frac{\delta\alpha}{3a}\right).$$

C Supplementary Material for Chapter 4

C.1 Proof of Theorem 4.1.2

Theorem 4.1.2. *There exists a fully scalable MPC algorithm that, given a graph $G = (V, E)$ with n vertices and m edges and a collection of non-negative integer budgets $(b_u)_{u\in V}$ for vertices in G such that $\sum_{u\in V} b_u = B^*$, parameters ℓ and λ, can simulate, for every $u \in V$, b_u independent random walks on G of length ℓ from u with an arbitrarily low error, in $O(\log \ell \log_\lambda B^*)$ rounds and $\widetilde{O}(m\lambda\ell^4 + B^*\lambda\ell)$ total space. The generated walks are independent across starting vertices $u \in V$.*

Proof. First we consider the setting where all budgets b_u are either the same value b, or 0. We call vertices u, where $b_u = b$ roots, and the set of roots R. We can now run Algorithm 13, with two simple modification: In Line 7 we set W to be all rooted walks, that is $W \leftarrow \cup_{r\in R} W_1(R)$. Correspondingly, in Line 11, we set the budget to $B(v,k) = (B_0(v) + R\cdot\lambda^i \cdot \frac{\kappa}{|W|})\cdot\tau^{3k-3}$, since there are now R times as many rooted walks.

From here, the proof of correctness proceeds nearly identically. In the case of a single vertex, we defined $P^k(v)$ as the probability that a walk from r reaches v as its k^{th} step. Here we must define such a quantity for each $r \in R$: $P_r^k(v)$. The analogous claim to the central Claim 4.2.4 is that for all $v \in V$ and $k \in [\ell]$:

$$B(v,k) \in \left[\left(B_0(v) + \lambda^i \cdot \sum_{r\in R} P_r^k(v)\right)\cdot\tau^{3k-4}, \left(B_0(v) + \lambda^i \cdot \sum_{r\in R} P_r^k(v)\right)\cdot\tau^{3k-2}\right].$$

In order to generalize to an arbitrary vector of budgets $(b_u)_{u\in V}$, we simply write b as the summation of vectors $b^{(1)}, \ldots, b^{(\log B^*)}$, where each vector b_i has all of it's non-zero entries within a factor 2 of each other. We then simply augment the coordinates of each b_i where necessary, to get vectors $\widetilde{b}^{(i)}$ which have all non-zero entries equal to each other. At this point we have reverted to the simpler case: we can run our algorithm in parallel for all $\log B*$ budget vector, which incurs the insignificant extra factor of $\log B*$ in memory.

□

C.2 Preliminaries of Section 4.3

For an undirected graph $G = (V, E)$, for each vertex $v \in V$, we denote its degree by $d(v)$ and for any set $S \subset V$, we define $\mathrm{Vol}(S) := \sum_{v \in S} d(v)$ and $\mathrm{Vol}(G) = 2|E|$. We define the stationary distribution over the graph as

$$\forall v \in V: \ \psi(v) := \frac{d(v)}{\mathrm{Vol}(G)}$$

For any vector p over the vertices and any $S \subseteq V$ we define

$$p(S) := \sum_{v \in S} p(v).$$

Moreover for any vector p over the vertices, we define p_+ as follows:

$$\forall v \in V: \ p_+(v) = \max(p(v), 0).$$

The *edge boundary* of a set $S \subseteq V$ is defined as

$$\partial(S) = \{\{u, v\} \in E \text{ such that } u \in S, v \notin S\}.$$

The conductance of any set $S \subseteq V$ is defined as

$$\Phi(S) = \frac{|\partial(S)|}{\min\{\mathrm{Vol}(S), 2m - \mathrm{Vol}(S)\}}$$

PageRank In the literature, PageRank was introduced for the first time in Brin and Page (1998); Page et al. (1999) for search ranking with starting vector of $s = \vec{1}/n$ (the uniform vector). Later, personalized PageRank introduced where the starting vector is not the uniform vector, in order to address personalized search ranking problem and context sensitive-search Berkhin (2007); Fogaras and Rácz (2004); Haveliwala (2003); Jeh and Widom (2003). In the rest of this chapter, we mostly work with personalized PageRanks, where the starting vector is an indicator vector for a vertex in the graph, and we use the general term of PageRank (as opposed to personalized PageRank) to avoid repetition.

Definition C.2.1 (PageRank)**.** *The PageRank vector $pr_\alpha(s)$ is defined as the unique solution of the linear system* $\mathrm{pr}_\alpha(s) = \alpha s + (1-\alpha)\mathrm{pr}_\alpha(s)W$, *where $\alpha \in (0,1]$ and called the* teleport probability, *s is the* starting vector, *and W is the lazy random walk transition matrix $W := \frac{1}{2}(I + D^{-1}A)$.*

Below, we mention a few facts about PageRank vectors.

Fact C.2.2. *For any starting vector s, and any constant $\alpha \in (0,1]$, there is a unique vector* $\mathrm{pr}_\alpha(s)$ *satisfying* $\mathrm{pr}_\alpha(s) = \alpha s + (1-\alpha)\mathrm{pr}_\alpha(s)W$.

Fact C.2.3. *A PageRank vector is a weighted average of lazy random walk vectors. More specifically,* $\mathrm{pr}_\alpha(s) = \alpha s + \alpha \sum_{t=1}^{\infty} (1-\alpha)^t (sW^t)$.

Now, we define a notion of approximation that will be used throughout the chapter.

Definition C.2.4. *(η-additive approximations) We call a vector q, an η-additive approximate PageRank vector for $p := \mathrm{pr}_\alpha(s)$, if for all $v \in V$, we have $q(v) \in \left[p(v) - \eta, p(v) + \eta\right]$.*

Sweeps Suppose that we are given a vector p that imposes an ordering over the vertices of graph $G = (V, E)$, as $v_1, v_2, \ldots, v_n$, where the ordering is such that

$$\frac{p(v_1)}{d(v_1)} \geq \ldots \geq \frac{p(v_n)}{d(v_n)}.$$

For any $j \in [n]$ define, $S_j := \{v_1, \ldots, v_j\}$. We define

$$\Phi(p) := \min_{i \in [n]} \Phi(S_i).$$

Empirical vectors Suppose that a distribution over vertices of the graph is given by a vector q. Now, imagine that at each step, one samples a vertex according to q, independently, and repeats this procedure for M rounds. Let vector N be such that for any vertex $v \in V$, $N(v)$ is equal to the number of times vertex v is sampled. We call vector $\widetilde{q}$ a (M, q)-empirical vector, where

$$\forall v \in V: \ \widetilde{q}(v) := \frac{N(v)}{M}$$

Claim C.2.5 (Additive guarantees for empirical vectors)**.** *Let q be a distribution vector over vertices of graph, where for each coordinate. Then, let vector $\widetilde{q}$ be a $(\frac{100}{\beta^2}\log n, q)$-empirical vector, for some β. Then $\forall v \in V: |q(v) - \widetilde{q}(v)| \leq \beta$ with high probability.*

Proof. Using additive Chernoff Bound (Lemma C.3.1 with $N = \frac{100 \log n}{\beta^2}$ and $\Delta = \beta$), for any $v \in V$, we have

$$\Pr[|q(v) - \widetilde{q}(v)| > \beta] \leq 2\exp\left(-2\frac{100\log n}{\beta^2}\beta^2\right) \leq n^{-20}.$$

Taking union bound over the vertices of the graph concludes the proof. □

C.3 Omitted claims and proofs

Lemma C.3.1 (Additive Chernoff Bound)**.** *Let $X_1, X_2, \ldots, X_N \in [0, 1]$ be N iid random variables, let $\bar{X} := (\sum_{i=1}^N X_i)/N$, and let $\mu = \mathbb{E}[\bar{X}]$. For any $\Delta > 0$ we have*

$$\Pr[\bar{X} - \mu \geq \Delta] \leq \exp\left(-2N\Delta^2\right)$$

and

$$\Pr[\bar{X} - \mu \leq -\Delta] \leq \exp\left(-2N\Delta^2\right).$$

Proof of Theorem 4.3.1: We prove this theorem in a few steps. First, we prove that a proper truncation of the formula in Fact C.2.3 is a good approximation for PageRank vector:

Claim C.3.2. *For $T \geq \frac{10\log n}{\alpha}$, we have that $q := \alpha s + \alpha\sum_{i=1}^{T}(1-\alpha)^i(sW^i)$ is a n^{-10}-additive approximate PageRank vector for $p := \mathrm{pr}_\alpha(s)$.*

Proof. Since s is an indicator vector and W is a lazy random walk matrix, for any integer $t > 0$, sW^t is a distribution vector, and consequently every coordinate is bounded by 1. So, for any vertex $v \in V$, we can bound $q(v) - p(v)$ in the following way:

$$|q(v) - p(v)| \leq \alpha \sum_{i=T+1}^{\infty}(1-\alpha)^i \leq (1-\alpha)^{\frac{10\log n}{\alpha}} \leq \left(e^{-\alpha}\right)^{\frac{10\log n}{\alpha}} = n^{-10},$$

since $1-\alpha \leq e^{-\alpha}$ and $T \geq \frac{10\log n}{\alpha}$. □

From now on, we set $T := \frac{10\log n}{\alpha}$. Now, we show that using empirical vectors output by our parallel algorithm for generating random walks incurs small error.

Claim C.3.3. *For any $i \in [T]$, let q_i be the distribution vector for the end point of lazy random walks of length i, output by the main algorithm with TVD error of n^{-10} (see Theorem 4.1.1). Additionally, let vector $\widetilde{q}_i$ be a $(\frac{10^6\log^3 n}{\eta^2\alpha^2}, q_i)$-empirical vector. Now define*

$$\widetilde{q} := \alpha s + \alpha\sum_{i=1}^{T}(1-\alpha)^i \cdot \widetilde{q}_i$$

for a constant $\alpha \in (0,1)$ and $T = \frac{10\log n}{\alpha}$. Then, $\widetilde{q}$ is an η-additive approximation to $p := \mathrm{pr}_\alpha(s)$.

Proof. For the upper bound, for any $v \in V$ we have

$$\begin{aligned}
\widetilde{q}(v) &= \alpha s + \alpha\sum_{i=1}^{T}(1-\alpha)^i \cdot \widetilde{q}_i(v) \\
&\leq \alpha s + \alpha\sum_{i=1}^{T}(1-\alpha)^i \cdot \left(q_i(v) + \frac{\eta\alpha}{100\log n}\right) && \text{By Claim C.2.5 with } \beta = \frac{\eta\alpha}{100\log n} \\
&\leq \alpha s + \alpha\sum_{i=1}^{T}(1-\alpha)^i \cdot q_i(v) + \frac{\eta}{10} && \text{Since } T = \frac{10\log n}{\alpha} \\
&\leq \alpha s + \alpha\sum_{i=1}^{T}(1-\alpha)^i(sW^i) + n^{-10} + \frac{\eta}{10} && \text{Using the main algorithm with TVD error } n^{-10} \\
&\leq p(v) + 2n^{-10} + \frac{\eta}{10} && \text{By Claim C.3.2} \\
&\leq p(v) + \eta.
\end{aligned}$$

And similarly for the lower bound, for any $v \in V$ we have

$$\widetilde{q}(v) = \alpha s + \alpha\sum_{i=1}^{T}(1-\alpha)^i \cdot \widetilde{q}_i(v)$$

$$\begin{aligned}
&\geq \alpha s + \alpha \sum_{i=1}^{T} (1-\alpha)^i \cdot \left(q_i(v) - \frac{\eta\alpha}{100\log n}\right) && \text{By Claim C.2.5 with } \beta = \frac{\eta\alpha}{100\log n} \\
&\geq \alpha s + \alpha \sum_{i=1}^{T} (1-\alpha)^i \cdot q_i(v) - \frac{\eta}{10} && \text{Since } T = \frac{10\log n}{\alpha} \\
&\geq \alpha s + \alpha \sum_{i=1}^{T} (1-\alpha)^i (sW^i) - n^{-10} + \frac{\eta}{10} && \text{Using the main algorithm with TVD error } n^{-10} \\
&\geq p(v) - 2n^{-10} - \frac{\eta}{10} && \text{By Claim C.3.2} \\
&\geq p(v) - \eta.
\end{aligned}$$

□

This means that we need to generate $B^* := \frac{10^6 \log^3 n}{\eta^2\alpha^2}$ random walks of length $\ell := \frac{10\log n}{\alpha}$. Now, using Lemma 4.2.3

1. in $O(\log \ell \cdot \log_\lambda B^*)$ rounds of MPC communication,
2. and with the total amount of memory of $O(m\lambda\ell^4 \log n + B^*\lambda\ell)$

we can generate the required random walks. □

Theorem C.3.4. *Let q be an η-additive approximate PageRank vector for $p := \mathrm{pr}_\alpha(s)$, where $||s_+||_1 \leq 1$. If there exists a subset of vertices S and a constant δ satisfying*

$$q(S) - \psi(S) > \delta$$

and η is such that

$$\eta \leq \frac{\delta}{8\left\lceil \frac{8}{\phi^2}\log(4\sqrt{\mathrm{Vol}(S)}/\delta)\right\rceil \min(\mathrm{Vol}(S), 2m - \mathrm{Vol}(S))},$$

then

$$\Phi(q) < \sqrt{\frac{18\alpha \log(4\sqrt{\mathrm{Vol}(S)}/\delta)}{\delta}}.$$

Proof of Theorem C.3.4: Let $\phi := \Phi(q)$. By Lemma C.3.5, for any subset of vertices S and any integer t, we have

$$q(S) - \psi(S) \leq \alpha t + \sqrt{X}\left(1 - \frac{\phi^2}{8}\right)^t + 2t \cdot X\eta$$

where $X := \min(\mathrm{Vol}(S), 2m - \mathrm{Vol}(S))$. If we set

$$t = \left\lceil \frac{8}{\phi^2}\log(4\sqrt{\mathrm{Vol}(S)}/\delta)\right\rceil \leq \frac{9}{\phi^2}\log(4\sqrt{\mathrm{Vol}(S)}/\delta),$$

then we get

$$\sqrt{\min(\mathrm{Vol}(S), 2m - \mathrm{Vol}(S))}\left(1 - \frac{\phi^2}{8}\right)^t \leq \frac{\delta}{4}.$$

This results in

$$q(S) - \psi(S) \leq \alpha \frac{9}{\phi^2} \log(4\sqrt{\mathrm{Vol}(S)}/\delta) + \frac{\delta}{4} + 2tX\eta$$

Now, as we did set η such that

$$\eta \leq \frac{\delta}{8tX}$$

then since we assumed that $q(S) - \psi(S) \geq \delta$ then

$$\frac{\delta}{2} < \alpha \frac{9}{\phi^2} \log(4\sqrt{\mathrm{Vol}(S)}/\delta),$$

which is equivalent to

$$\phi < \sqrt{\frac{18\alpha \log(4\sqrt{\mathrm{Vol}(S)}/\delta)}{\delta}}.$$

□

Lemma C.3.5. *Let q be an η-additive approximate PageRank vector for $p := \mathrm{pr}_\alpha(s)$, where $||s_+||_1 \leq 1$. Let ϕ and γ be any constants in $[0,1]$. Either the following bound holds for any set of vertices S and any integer t:*

$$q(S) - \psi(S) \leq \gamma + \alpha t + \sqrt{X}\left(1 - \frac{\phi^2}{8}\right)^t + 2t \cdot X\eta$$

where $X := \min(\mathrm{Vol}(S), 2m - \mathrm{Vol}(S))$, or else there exists a sweep cut S_j^q, for some $j \in [1, |\mathrm{Supp}(q)|]$, with the following properties:

1. *$\Phi(S_j^q) < \phi$,*
2. *For some integer t,*

   $$q(S_j^q) - \psi(S_j^q) > \gamma + \alpha t + \sqrt{X'}\left(1 - \frac{\phi^2}{8}\right)^t + 2t \cdot X'\eta,$$

   where $X' := \min(\mathrm{Vol}(S_j^q), 2m - \mathrm{Vol}(S_j^q))$.

Proof of Lemma C.3.5: For simplicity of notation let $f_t(x) := \gamma + \alpha t + \sqrt{\min(x, 2m - x)}\left(1 - \frac{\phi^2}{8}\right)^t$. We are going to prove by induction that if there does not exist a sweep cut with both of the properties then equation

$$q[x] - \frac{x}{2m} \leq f_t(x) + 2t \cdot \min(x, 2m - x)\eta \tag{C.1}$$

holds for all $t \geq 0$.

Base of induction ($t = 0$)**:**We need to prove that for any $x \in [0, 2m]$, $q[x] - \frac{x}{2m} \leq \gamma + \sqrt{\min(x, 2m - x)}$. The claim is true for $x \in [1, 2m-1]$ since $q[x] \leq 1$ for any x, so, we only need to prove the claim for $x \in [0,1] \cup [2m-1, 2m]$.

Case I, $x \in [0,1]$**:**For $x \in [0,1]$, $q[0] = 0$ and $q[1] \leq 1$ and $q[x]$ is a linear function for $x \in [0,1]$. Also $\sqrt{\min(x, 2m - x)} = \sqrt{x}$. Since $\sqrt{x}$ is a concave function then the claim holds for $x \in [0,1]$.

Case II, $x \in [2m-1, 2m]$**:**In this case $\sqrt{\min(2m - x, x)} + \frac{x}{2m} = \sqrt{2m - x} + \frac{x}{2m}$, which is a concave function. So we only need to check the end points of this interval. For $x = 2m$, the claim holds since $q[2m] = 1$. Similarly, for $x = 2m - 1$, $q[x] \leq 1 \leq \sqrt{1} + \frac{2m-1}{2m}$.

So the base of induction holds.

Inductive step: Now assume that Eq. (C.1) holds for some integer t. We prove that it holds for $t+1$. We only need to prove that it holds for $x_j = \mathrm{Vol}(S_j^q)$ for each $j \in [1, \mathrm{Supp}(q)]$. Consider any $j \in [1, |\mathrm{Supp}(q)]$, and let $S := S_j^q$. If property 2 does not hold, then the claim holds. If property 1 does not hold, then we have $\Phi(S) \geq \phi$. Assume that $x_j \leq m$ (the other case is similar)

$$\begin{aligned} q[\mathrm{Vol}(S)] - \frac{x_j}{2m} &= q(S) - \frac{x_j}{2m} && \text{Since } S \text{ is a sweep cut of } q \\ &\leq p(S) + |S| \cdot \eta - \frac{x_j}{2m} && \text{By Definition C.2.4} \end{aligned}$$

Let $F := \mathrm{in}(S) \cap \mathrm{out}(S)$ and $F' := \mathrm{in}(S) \cup \mathrm{out}(S)$. By Lemma C.3.8,

$$p(S) = \alpha s(S) + (1-\alpha)\left(\frac{1}{2}p(F) + \frac{1}{2}p(F')\right). \tag{C.2}$$

Consequently, we have

$$\begin{aligned} q[x_j] &\leq p(S) + |S| \cdot \eta \\ &\leq \alpha s(S) + (1-\alpha)\left(\frac{1}{2}p(F) + \frac{1}{2}p(F')\right) + |S| \cdot \eta && \text{By Eq. (C.2)} \\ &\leq \alpha + \left(\frac{1}{2}p(F) + \frac{1}{2}p(F')\right) + |S| \cdot \eta && \text{By } \|s_+\|_1 \leq 1 \text{ and } \alpha \in [0,1] \\ &\leq \alpha + \left(\frac{1}{2}q(F) + \frac{1}{2}q(F') + x_j\eta\right) + |S| \cdot \eta && \text{By Claim C.3.9} \\ &\leq \alpha + \left(\frac{1}{2}q[x_j - |\partial(S)|] + \frac{1}{2}q[x_j + |\partial(S)|] + x_j\eta\right) + |S| \cdot \eta && \text{By definition of } q[\cdot] \\ &= \alpha + \left(\frac{1}{2}q[x_j - \Phi(S)x_j] + \frac{1}{2}q[x_j + \Phi(S)x_j] + x_j\eta\right) + |S| \cdot \eta && \text{By definition of } \Phi(S) \end{aligned}$$

$$\le \alpha + \frac{1}{2}q[x_j - \phi x_j] + \frac{1}{2}q[x_j + \phi x_j] + 2x_j\eta \qquad \text{By concavity of } q$$
$$\le \alpha + \frac{1}{2}f_t[x_j - \phi x_j] + \frac{1}{2}f_t[x_j + \phi x_j] + 2tx_j\eta + \frac{x_j}{2m} + 2x_j\eta \qquad \text{By induction assumption}$$

Therefore

$$\begin{aligned}
&q[x_j] - \frac{x_j}{2m}\\
&\le \alpha + \frac{1}{2}f_t[x_j - \phi x_j] + \frac{1}{2}f_t[x_j + \phi x_j] + 2(t+1)x_j\eta\\
&= \gamma + \alpha + \alpha t + \frac{1}{2}\left(\sqrt{x_j - \alpha x_j} + \sqrt{x_j + \alpha x_j}\right)\left(1 - \frac{\phi^2}{8}\right)^t + 2(t+1)x_j\eta\\
&\le \gamma + \alpha(t+1) + \sqrt{x_j}\left(1 - \frac{\phi^2}{8}\right)^{t+1} + 2(t+1)x_j\eta
\end{aligned}$$

□

Definition C.3.6. *For any vertex $u \in V$ and any v in neighborhood of u, we define*

$$p(u,v) = \frac{p(u)}{d(u)}.$$

Also, we replace each edge $(u,v) \in E$ with two directed edges (u,v) and (v,u). Now, for any subset of directed edges A, we define

$$P(A) = \sum_{(u,v)\in A} p(u,v).$$

Definition C.3.7. *For any subset of vertices S, we define*

$$\text{in}(S) = \{(u,v) \in E | v \in S\}$$

and

$$\text{out}(S) = \{(u,v) \in E | u \in S\}$$

Lemma C.3.8. *If $p = \text{pr}_\alpha(s)$ is a PageRank vector, then for any subset of vertices S,*

$$p(S) = \alpha(S) + (1-\alpha)\left(\frac{1}{2}p(\text{in}(S) \cap \text{out}(S)) + \frac{1}{2}p(\text{in}(S) \cup \text{out}(S))\right).$$

Claim C.3.9. *Suppose that q is an η-additive approximate PageRank vector for $p = \text{pr}_\alpha(s)$ (see Definition C.2.4). Then, for any subset of vertices S, if we let $F := \text{in}(S) \cap \text{out}(S)$ and $F' := \text{in}(S) \cup \text{out}(S)$,*

$$-2\text{Vol}(S)\eta \le \left(q(F) + q(F')\right) - \left(p(F) + p(F')\right) \le 2\text{Vol}(S)\eta$$

Proof. By Definition C.3.7, if we define

$$q(F) = \sum_{(u,v)\in F} \frac{q(u)}{d(u)} \le \sum_{(u,v)\in F} \frac{p(u)+\eta}{d(u)} \le \sum_{(u,v)\in F} \frac{p(u)}{d(u)} + \eta|F| = p(F) + \eta|F|.$$

Similarly,

$$p(F) - \eta|F| \le q(F).$$

If we repeat the same procedure for $F' := \mathrm{in}(S) \cup \mathrm{out}(S)$, we get,

$$p(F') - \eta|F'| \le q(F') \le p(F') + \eta|F'|.$$

In order to conclude the proof, we only need to note that

$$|F| + |F'| = 2\mathrm{Vol}(S).$$

□

Lemma C.3.10 (Theorem 4 of Andersen et al. (2006))**.** *For any set C and any constant $\alpha \in (0,1]$, there is a subset $C_\alpha \subseteq C$ with volume $\mathrm{Vol}(C_\alpha) \ge \mathrm{Vol}(C)/2$ such that for any vertex $v \in C_\alpha$, the PageRank vector $\mathrm{pr}_\alpha(\chi_v)$ satisfies*

$$[\mathrm{pr}_\alpha(\chi_v)](C) \ge 1 - \frac{\Phi(C)}{\alpha}$$

where $[\mathrm{pr}_\alpha(\chi_v)](C)$ is the amount of probability from PageRank vector over set C.

See Andersen et al. (2006) for the proof of Lemma C.3.10.

Lemma C.3.11. *Let $\alpha \in (0,1]$ be a constant and let C be a set satisfying*

1. *$\Phi(C) \le \alpha/10$,*
2. *$\mathrm{Vol}(C) \le \frac{2}{3}\mathrm{Vol}(G)$.*

If q is a η-additive approximation to $\mathrm{pr}_\alpha(\chi_v)$ where $v \in C_\alpha$ and $\eta \le 1/(10\mathrm{Vol}(C))$, then a sweep over q produces a cut with conductance $\Phi(q) = O(\sqrt{\alpha \log(\mathrm{Vol}(C))})$.

Proof. Since q is a η-additive approximation to $\mathrm{pr}_\alpha(\chi_v)$, then using Lemma C.3.10 we have

$$q(C) \ge 1 - \frac{\Phi(C)}{\alpha} - \eta \cdot |C| \ge 1 - \frac{\Phi(C)}{\alpha} - \eta \cdot \mathrm{Vol}(C),$$

since $|C| \le \mathrm{Vol}(C)$. Combining this with the facts that $\Phi(C)/\alpha \le \frac{1}{10}$ and $\eta \le 1/(10\mathrm{Vol}(C))$, we have $q(C) \ge 4/5$, which implies

$$q(C) - \psi(C) \ge \frac{4}{5} - \frac{2}{3} = \frac{2}{15}.$$

Now, Theorem C.3.4 implies that

$$\Phi(q) \leq \sqrt{135\alpha \log(30\sqrt{\mathrm{Vol}(C)})}.$$

□

Proof of Theorem 4.1.4: The proof is by combining Theorem 4.3.1 and Lemma C.3.11. □

C.4 Additional Experiments

We present the result of experimentation with longer walks ($\ell = 32$) in Table C.1. Similarly to the other cases, the algorithm scales extremely well with the size of the graph. Furthermore, we observe that in the case of the smaller of the graphs (COM-DBLP, COM-YOUTUBE), doubling the walk-length has a relatively small effect on the run-time. This is to be expected, as the number of Map-Reduce rounds performed scales logarithmically in ℓ (see Theorem 4.1.1). In the larger graphs, this is less evident, as the running time depends more and more on the work-load as opposed to the rounds complexity.

Table C.1 – Experiments with $\ell = 32$, $C = 3$, $B_0 = 5n/m$, $\lambda = 32$, $\tau = 1.3$.

GRAPH	TIME	B_0	ROOTED WALKS GENERATED	WALK FAILURE RATE
COM-DBLP	25 ± 2 MINUTES	1.51	$79,103 \pm 2412$	$19.4 \pm 1.1\%$
COM-YOUTUBE	45 ± 1 MINUTES	1.9	$44,839 \pm 179$	$7.8 \pm 1\%$
COM-LIVEJOURNAL	115 ± 3 MINUTES	0.576	$152,126 \pm 3028$	$7.9 \pm 0.2\%$
COM-ORKUT	95 ± 1 MINUTES	0.131	$163,056 \pm 1612$	$5 \pm 0.1\%$

In Table C.2 we see an experiment similar to that of Table 4.2, but with the parameters B_0 and τ somewhat lowered. We confirm the results on Section 4.4.1 on the scaling of running time with the size of the graph. The lower parameters allow for faster running time. However, this is at the expense of both the walk failure rate and the number of rooted walks generated. With lower B_0 and τ the vertex budgets ($B(v, K)$ from Section 4.2) are smaller, and allow for higher relative deviation from the expectation, leading to more walk failure. The running time decrease is not significant, especially in the case of our smaller graphs, and we conclude that the setting of parameters in Table 4.2 are closer to optimal for most applications.

Table C.2 – Experiments with $\ell = 16$, $C = 3$, $B_0 = 3n/m$, $\lambda = 32$, $\tau = 1.2$.

GRAPH	TIME	B_0	ROOTED WALKS GENERATED	WALK FAILURE RATE
COM-DBLP	17 ± 1 MINUTES	0.906	$23,837 \pm 2210$	$38.3 \pm 0.7\%$
COM-YOUTUBE	23 ± 2 MINUTES	1.14	$15,977 \pm 2298$	$28.1 \pm 1.7\%$
COM-LIVEJOURNAL	35 ± 0 MINUTES	0.346	$57,460 \pm 2104$	$26.2 \pm 0.5\%$
COM-ORKUT	33 ± 1 MINUTES	0.079	$66,715 \pm 1502$	$21.5 \pm 0.3\%$

Finally, in Table C.3 we present the results of a comparison experiment, extremely similar to that of Table 4.3, but with λ increased to 20. The discrepancy is even more striking. Increasing the target budget by a factor of 4 produces no measurable difference for Algorithm 13. However, UNIFORM STITCHING is no longer able to complete on the cluster for inputs EMAIL-ENRON and COM-DBLP, due to the high memory requirement (denoted as '—').

Table C.3 – Experiments with $\ell = 16$, $\lambda = 20$, $\tau = 1.3$. The row labeled 'Algorithm 13' corresponds to $B_0 = 1$, $C = 3$, while the row labeled 'Uniform Stitching' corresponds to $B_0 = 400$, $C = 1$.

ALGORITHM	CA-GRQC	EMAIL-ENRON	COM-DBLP
ALGORITHM 13	15 ± 1 MINUTES	19 ± 1 MINUTES	17 ± 1 MINUTES
UNIFORM STITCHING	8 ± 0 MINUTES	—	—

Implementation details. In Algorithm 13, $B(v,k)$ – the budget associated with the k^{th} step of the random walk – is proportional to τ^{3k} (see Line 11 and Line 13) which can lead to a factor $\tau^{\theta(\ell)}$ blow-up in space. In theory this is not a significant loss asymptotically, due to the settings of τ and θ. Nonetheless, in practice, we use a more subtle formula which leads only to a factor $\tau^{\log_2 \ell}$ blow-up, while retaining a similar guarantee on the probability of failure.

Furthermore, in Algorithm 13 (and the intuitive explanation before it) we distinguish between $W_k(v)$ for different k. That is walk segments have predetermined positions in the walk, and a request to stitch to a walk ending in v with its k^{th} step can only be served by a walk starting in v with its $k+1^{\text{st}}$ step. This is mostly for ease of understanding and analysis. In the implementation we make no such distinction. Each node simply stores a set of walks of length 2^i in the i^{th} round. The initial budget of each vertex v (at the beginning of the cycle) is set to $\sum_k B(v,k)$, where $B(v,k)$ is still calculated according to the formulas in Line 11 and Line 13 of Algorithm 13 (with the exception of the altered τ-scaling term, as mentioned in the paragraph above).

D Supplementary Material for Chapter 6

D.1 Proof of Lemma 6.5.8

In this section we prove the following result, stated in Section 6.5.2.

Lemma 6.5.8 (Concentration on $\widehat{M}(v)$)**.** *For any vertex v, any $j > 0$, and constants c and x such that $c \geq 20$ and $x \geq 100c\log c$, we have:*

$$\mathbb{P}\left[\widehat{M}(v) \geq x \wedge \widehat{L}(v) = j+1\right] \leq \frac{10c^2}{x^2} \cdot \mathbb{E}\left[\widehat{M}(v) \cdot \mathbb{1}\left[\widehat{L}(v) = j\right]\right]. \tag{6.18}$$

Where $\widehat{L}(v)$ and $\widehat{M}(v)$ are defined in Eq. (6.10) and Eq. (6.16), respectively.

Proof. We begin by rewriting the LHS and the RHS of Eq. (6.18).

Rewriting the LHS of Eq. (6.18) Observe that

$$\mathbb{P}\left[\widehat{M}(v) \geq x \,|\widehat{L}(v) = j+1\right] \overset{\text{Observation 6.5.4 Item (b)}}{=} \mathbb{P}\left[\widehat{M}_{j+1}(v) \geq x \,|\widehat{L}(v) = j+1\right].$$

First observe that $\mathbb{P}\left[\widehat{M}(v) \geq x \,|\widehat{L}(v) = j+1\right] = \mathbb{P}\left[\widehat{M}_{j+1}(v) \geq x \,|\widehat{L}(v) = j+1\right]$, since $\widehat{M}_{j+1}(v) = \widehat{M}(v)$ conditioned on $\widehat{L}(v) = j+1$ by definition. We thus have

$$\begin{aligned}
\mathbb{P}\left[\widehat{M}(v) \geq x \wedge \widehat{L}(v) = j+1\right] &= \mathbb{P}\left[\widehat{M}_{j+1}(v) \geq x \wedge \widehat{L}(v) = j+1\right] \\
&\overset{\text{Observation 6.5.4 Item (a)}}{=} \mathbb{P}\left[\widehat{M}_{j+1}(v) \geq x\right]\mathbb{P}\left[\widehat{L}(v) = j+1\right] \\
&= \mathbb{P}\left[\widehat{M}_{j+1}(v) \geq x\right]\mathbb{P}\left[v \in \widehat{V}_j\right]\mathbb{P}\left[v \in \widehat{V}_{j+1} | v \in \widehat{V}_j\right]\mathbb{P}\left[v \not\in \widehat{V}_{j+2} | v \in \widehat{V}_{j+1}\right] \\
&= \mathbb{P}\left[\widehat{M}_{j+1}(v) \geq x\right]\mathbb{P}\left[v \in \widehat{V}_j\right]\mathbb{P}\left[S_j(v) < \delta\right]\mathbb{P}\left[S_{j+1}(v) \geq \delta\right],
\end{aligned} \tag{D.1}$$

where $S_j(v)$ is as defined in Eq. (6.13). (When $j = J$, $\mathbb{P}\left[v \not\in \widehat{V}_{j+2}\right]$ is simply 1.)

Rewriting the RHS of Eq. (6.18) We have

$$\mathbb{E}\left[\widehat{M}(v) \cdot \mathbb{1}\left[\widehat{L}(v) = j\right]\right] = \mathbb{P}\left[v \in \widehat{V}_j\right]\mathbb{P}\left[\widehat{L}(v) = j | v \in \widehat{V}_j\right]\mathbb{E}\left[\widehat{M}_j(v)\right].$$

By definition $\mathbb{P}\left[\widehat{L}(v)=j|v\in\widehat{V}_j\right]=\mathbb{P}\left[S_j\geq\delta\right]$, and hence

$$\mathbb{E}\left[\widehat{M}(v)\cdot\mathbb{1}\left[\widehat{L}(v)=j\right]\right]=\mathbb{P}\left[v\in\widehat{V}_j\right]\mathbb{P}\left[S_j(v)\geq\delta\right]\mathbb{E}\left[\widehat{M}_j(v)\right]. \tag{D.2}$$

The proof strategy In the rest of the proof, to establish Eq. (6.18) we upper-bound the ratio of the LHS of Eq. (D.1) and the RHS of Eq. (D.2) by $10c^2/x^2$. At a high level, the RHS of Eq. (D.1) is small when $x\gg\delta$. This is the case since the random variables $\widehat{M}_{j+1}(v)$ and $S_j(v)$ (see Eq. (6.17) and Eq. (6.13) respectively) have similar expectations and they both concentrate well around their expectations. Hence, it is unlikely that at the same time $\widehat{M}_{j+1}(v)$ is large and $S_j(v)$ is small.

To implement this intuition, we consider two cases with respect to $\mathbb{E}\left[\widehat{M}_{j+1}(v)\right]$. First, when $\mathbb{E}\left[\widehat{M}_{j+1}(v)\right]$ is relatively large, i.e., at least $x/2$, we show that $\mathbb{P}\left[S_j(v)<\delta\right]$ is small. On the other hand, for the terms appearing on the RHS of Eq. (D.2) we have: large $\mathbb{E}\left[\widehat{M}_{j+1}(v)\right]$ implies large $\mathbb{E}\left[\widehat{M}_j(v)\right]$, and small $\mathbb{P}\left[S_j(v)<\delta\right]$ implies that $\mathbb{P}\left[S_j(v)\geq\delta\right]\geq 1/2$.

Second, when $\mathbb{E}\left[\widehat{M}_{j+1}(v)\right]<x/2$, we show that $\mathbb{P}\left[\widehat{M}_{j+1}(v)\geq x\right]$ is very small.

We complete the proof by balancing the two cases.

Case 1: $\mathbb{E}\left[\widehat{M}_{j+1}(v)\right]\geq x/2$. In this case we have

$$\mathbb{E}\left[S_j(v)\right]\overset{\text{Observation 6.5.5}}{\geq}\frac{\mathbb{E}\left[S_{j+1}(v)\right]}{c+1}\overset{\text{Observation 6.5.4 Item (c)}}{=}\frac{\mathbb{E}\left[\widehat{M}_{j+1}(v)\right]}{c+1}\geq\frac{x}{2(c+1)}. \tag{D.3}$$

This further implies

$$\mathbb{E}\left[\widehat{M}_j(v)\right]\overset{\text{Observation 6.5.4 Item (c)}}{=}\mathbb{E}\left[S_j(v)\right]\overset{\text{Eq. (D.3)}}{\geq}\frac{x}{2(c+1)}. \tag{D.4}$$

Let us define a random Z_k for iid edge e_k as follows

$$Z_k=\mathbb{1}\left[e_k\text{ equals }\{w,v\}\right]\sum_{i=0}^{\min\{L(w),j\}}c^{i-j}.$$

Notice that $S_j(v)=\sum_{k=1}^{c^j m/n}Z_k$ and $Z_k\in[0,2]$, therefore by applying Chernoff bound Theorem 6.5.6 Item (c) to $S_j(v)$:

$$\begin{aligned}\mathbb{P}\left[S_j(v)<\delta\right]&\leq\mathbb{P}\left[S_j(v)\leq(1-1/2)\mathbb{E}\left[S_j(v)\right]\right]\\&\leq\exp\left(-\frac{(1/2)^2\mathbb{E}\left[S_j(v)\right]}{2\cdot 2}\right)\\&\overset{\text{Eq. (D.3)}}{\leq}\exp\left(-\frac{(1/2)^2\frac{x}{2(c+1)}}{2\cdot 2}\right)\\&\leq\exp\left(-\frac{x}{32(c+1)}\right)\\&\leq\exp\left(-\frac{x}{40c}\right),\end{aligned} \tag{D.5}$$

where the first inequality follows as $\delta \le 1/2$ and $\mathbb{E}\left[S_j(v)\right] \ge 1$ from Eq. (D.3) and the definition of x. The last inequality of Eq. (D.5) follows since $c \ge 20$ by the assumption of the lemma. Therefore, substituting Eq. (D.5) into Eq. (D.1), we get

$$\mathbb{P}\left[\widehat{M}(v) \ge x \wedge \widehat{L}(v) = j+1\right] \le \mathbb{P}\left[v \in \widehat{V}_j\right] \exp\left(-\frac{x}{40c}\right). \tag{D.6}$$

Furthermore, from Eq. (D.5) and for $x \ge 100c\log c$ we have $\mathbb{P}\left[S_j \ge \delta\right] \ge 1/2$. Substituting this bound and Eq. (D.4) into Eq. (D.2) leads to

$$\mathbb{E}\left[\widehat{M}(v) \cdot \mathbb{1}\left[\widehat{L}(v) = j\right]\right] \ge \frac{1}{2}\mathbb{P}\left[v \in \widehat{V}_j\right] \cdot \frac{x}{2(c+1)}.$$

Combining the last inequality with Eq. (D.6) leads to

$$\frac{\mathbb{P}\left[\widehat{M}(v) \ge x \wedge \widehat{L}(v) = j+1\right]}{\mathbb{E}\left[\widehat{M}(v) \cdot \mathbb{1}\left[\widehat{L}(v) = j\right]\right]} \le \frac{4(c+1) \cdot \exp(-\frac{x}{40c})}{x}. \tag{D.7}$$

Case 2: $\mathbb{E}\left[\widehat{M}_{j+1}(v)\right] < x/2$.Let $\mathbb{E}\left[\widehat{M}_{j+1}(v)\right] = tx$ for some $t < 1/2$. To apply Chernoff bound, similar to previous case we define Z_w for any vertex $w \in N(v)$ as follows

$$Z_w \equiv \sum_{i=0}^{\min\{\widehat{L}(w), j\}} c^i / n.$$

Therefore we have $\widehat{M}_j(v) = \sum_{w \in N(v)} Z_w$. Observe that $Z_w \in [0,2]$. Since $t < 1/2$, by applying Chernoff bound Theorem 6.5.6 Item (b) we get

$$\begin{aligned}
\mathbb{P}\left[\widehat{M}_{j+1}(v) \ge x\right] &= \mathbb{P}\left[\widehat{M}_{j+1}(v) \ge (1 + (1/t - 1)) \cdot \mathbb{E}\left[\widehat{M}_{j+1}(v)\right]\right] \\
&\le \exp\left(-\frac{1/t \cdot \log 1/t \cdot \mathbb{E}\left[\widehat{M}_{j+1}(v)\right]}{3 \cdot 2}\right) \\
&\le \exp\left(-\frac{x \log 1/t}{6}\right).
\end{aligned}$$

Substituting this bound into Eq. (D.1) we obtain

$$\mathbb{P}\left[\widehat{M}(v) \ge x \wedge \widehat{L}(v) = j+1\right] \le \mathbb{P}\left[v \in \widehat{V}_j\right] \mathbb{P}\left[S_{j+1} \ge \delta\right] \exp\left(-\frac{x \log 1/t}{6}\right). \tag{D.8}$$

Eq. (D.2) and Eq. (D.8) imply

$$\begin{aligned}
\frac{\mathbb{P}\left[\widehat{M}(v) \ge x \wedge \widehat{L}(v) = j+1)\right]}{\mathbb{E}\left[\widehat{M}(v) \cdot \mathbb{1}\left[\widehat{L}(v) = j\right]\right]} &\le \frac{2(c+1) \cdot \exp\left(-\frac{x \log 1/t}{6}\right)}{tx} \cdot \frac{\mathbb{P}\left[S_{j+1}(v) \ge \delta\right]}{\mathbb{P}\left[S_j(v) \ge \delta\right]} \\
&= \frac{2(c+1)}{x} \cdot t^{x/6-1} \cdot \frac{\mathbb{P}\left[S_{j+1}(v) \ge \delta\right]}{\mathbb{P}\left[S_j(v) \ge \delta\right]}
\end{aligned}$$

$$\leq \frac{2(c+1)}{x} \cdot 2^{-x/6+1} \cdot \frac{\mathbb{P}\left[S_{j+1}(v) \geq \delta\right]}{\mathbb{P}\left[S_j(v) \geq \delta\right]}$$

Now we upper bound $\frac{\mathbb{P}[S_{j+1}(v)\geq\delta]}{\mathbb{P}[S_j(v)\geq\delta]}$. Consider the definition of $S_j(v)$ from Eq. (6.13)

$$S_j(v) \equiv \sum_{\substack{k=1 \\ e_k \sim U_E}}^{c^j m/n} \mathbb{1}\left[e_k \text{ equals } \{w,v\}\right] \sum_{i=0}^{\min(L(w),j)} c^{i-j}$$

Let us split this definition into two parts: one corresponding to the last term of the second sum and one corresponding to all other terms:

$$\begin{aligned}
S_j(v) &= A_j(v) + B_j(v) \\
A_j(v) &= \sum_{\substack{k=1 \\ e_k \sim U_E}}^{c^j m/n} \mathbb{1}\left[e_k \text{ equals } \{w,v\}\right] \sum_{i=0}^{\min(L(w),j-1)} c^{i-j} \\
B_j(v) &= \sum_{\substack{k=1 \\ e_k \sim U_E}}^{c^j m/n} \mathbb{1}\left[e_k \text{ equals } \{w,v\}\right] \mathbb{1}\left[w \in \widehat{V}_j\right]
\end{aligned}$$

Note that $\mathbb{P}\left[S_j(v) \geq \delta\right] = \mathbb{P}\left[A_j(v) \geq \delta \ \vee\ B_j \geq 1\right] \leq \mathbb{P}\left[A_j(v) \geq \delta\right] + \mathbb{P}\left[B_j(v) \geq 1\right]$. We must bound

$$\frac{\mathbb{P}\left[A_{j+1} \geq \delta\right] + \mathbb{P}\left[B_{j+1}(v) \geq 1\right]}{\mathbb{P}\left[S_j(v) \geq \delta\right]}.$$

Notice that $A_{j+1}(v)$ is the average of c independently sampled copies of $S_j(v)$, say $S_j^{(i)}(v)$. In order for $A_{j+1}(v)$ to be greater than δ at least one of the $S_j^{(i)}(v)$'s must be greater than δ, therefore by union bound $\mathbb{P}\left[A_{j+1} \geq \delta\right] \leq c\mathbb{P}\left[S_j(v) \geq \delta\right]$. Notice now that $B_{j+1}(v)$ is *at most* the sum of c independent copies of $B_j(v)$, say $B_j^{(i)}(v)$. Since $B_j(v)$ is integral, in order for $B_{j+1}(v)$ to be greater than 1 at least one of the $B_j^{(i)}(v)$'s need to be greater than 1, therefore by union bound $\mathbb{P}\left[B_{j+1}(v)\right] \leq c\mathbb{P}\left[B_j(v) \geq 1\right] \leq c\mathbb{P}\left[S_j(v) \geq \delta\right]$. So in conclusion

$$\frac{\mathbb{P}\left[S_{j+1}(v) \geq \delta\right]}{\mathbb{P}\left[S_j(v) \geq \delta\right]} \leq 2c.$$

This finalizes the bound of this case as well

$$\frac{\mathbb{P}\left[\widehat{M}(v) \geq x \ \wedge\ \widehat{L}(v) = j+1)\right]}{\mathbb{E}\left[\widehat{M}(v)\mathbb{1}(\widehat{L}(v) = j)\right]} \leq \frac{2(c+1)}{x} \cdot 2^{-x/6+1} \cdot 2c. \tag{D.9}$$

Finalizing Combining the two cases, from Eq. (D.7) and Eq. (D.9) we conclude

$$\frac{\mathbb{P}\left[\widehat{M}(v) \geq x \wedge \widehat{L}(v) = j+1\right]}{\mathbb{E}\left[\widehat{M}(v) \mathbb{1} \widehat{L}(v) = j\right]} \leq \max\left(\frac{4c(c+1)}{x} \cdot 2^{-x/6+1}, \frac{4(c+1) \cdot \exp(-\frac{x}{40c})}{x}\right)$$

The RHS of the inequality above is upper-bounded by $10c^2/x^2$ for $c \geq 20$ and $x \geq 100 c \log c$. □

D.2 Oversampling Lemma

In this section we formally proof the following lemma.

Lemma 6.3.3 (Oversampling lemma)**.** *For sufficiently small $\delta > 0$ and large enough c the following holds. Let $X = \sum_{k=1}^{K} Y_k$ be a sum of independent random variables Y_k taking values in $[0,1]$, and $\overline{X} \equiv \frac{1}{c}\sum_{i=1}^{c} X_i$ where X_i are iid copies of X. If $\mathbb{E}[X] \leq \delta/3$ and $\mathbb{P}[X \geq \delta] = p$, then $\mathbb{P}\left[\overline{X} \geq \delta\right] \leq p/2$.*

Proof. Let $Z = \sum_{i=1}^{c} X_i$. Notice that Z is a sum of independent random variables each in the range $[0,1]$. Also, $\mathbb{P}\left[\overline{X} \geq \delta\right] = \mathbb{P}[Z \geq c\delta]$. From the definition of X_i and the linearity of expectation, we have $\mathbb{E}[Z] \leq c\delta/3$. This, in compination with Chernoff bound (Theorem 6.5.6Item (b)), further implies

$$\mathbb{P}\left[\overline{X} \geq \delta\right] = \mathbb{P}[Z \geq c\delta] \leq \exp\left(-\frac{2 \cdot c\delta/3}{3}\right) \leq \exp\left(-\frac{c\delta}{9}\right). \tag{D.10}$$

We now consider two cases depending on the value of p.

Case 1: $p \geq 2\exp\left(-\frac{c\delta}{}9\right)$**.**The proof follows directly from Eq. (D.10).

Case 2: $p < 2\exp\left(-\frac{c\delta}{}9\right)$**.**In this case we consider the following three events which we call *bad*.

- Event $\mathscr{E}_1$: At least two of X_i's have value at least δ.
- Event $\mathscr{E}_2$: At least one X_i has value more than t, for a threshold $t := \delta c/30 \gg \delta$.
- Event $\mathscr{E}_3$: At least one X_i has value more than δ and less than $0.1 \cdot c$ of the X_i's have value below $2\delta/3$.

If none of the bad events happen, then $\overline{X} \leq \delta$. To see that, observe that $\bar{\mathscr{E}}_1$ and $\bar{\mathscr{E}}_2$ imply that at most one X_i has value more than δ, and that the same X_i has value at most t. Note that $\bar{\mathscr{E}}_3$ is the event that either none of the X_i's has value more than δ or more than $0.1c$ of the X_i's have value below $2\delta/3$. In the former case, $\overline{X} \leq \delta$ is clearly satisfied. Consider now the latter case intersected with $\bar{\mathscr{E}}_1$ and $\bar{\mathscr{E}}_2$; denote the X_i larger than δ by X_{large}. At least $0.1 \cdot c$ values of X_i are less than $2\delta/3$ and the rest, excluding X_{large}, are less than δ. Therefore, the average of X_{large} and the elements having value less than $2\delta/3$ is at most $\frac{0.1 \cdot c \cdot 2\delta/3 + t}{0.1 \cdot c} = 2\delta/3 + 10t/c$. This is less than δ as long as $t \leq \delta c/30$. All other elements are below δ as well.

In the rest of the proof we upper-bound the probability that each of the bad events occurs. Then, by taking union bound we will upper-bound $\mathbb{P}\left[\overline{X} \geq \delta\right]$.

Upper-bound on $\mathbb{P}[\mathscr{E}_1]$. By union bound we have

$$\mathbb{P}[\mathscr{E}_1] = \mathbb{P}\left[\exists i_1 \neq i_2 : X_{i_1} \geq \delta \ \wedge \ X_{i_2} \geq \delta\right] \leq \binom{c}{2} p^2 \leq c^2 p \exp\left(-\frac{c\delta}{9}\right) \tag{D.11}$$

Upper-bound on $\mathbb{P}[\mathscr{E}_2]$. Again by union bound we derive

$$\begin{aligned}\mathbb{P}[\mathscr{E}_2] &= \mathbb{P}[\exists i : X_i \geq t] \\ &\leq c \cdot \mathbb{P}[X \geq \delta] \cdot \mathbb{P}[X \geq t | X \geq \delta] \\ &= cp \cdot \mathbb{P}[X \geq t | X \geq \delta]. \end{aligned} \tag{D.12}$$

To upper-bound $\mathbb{P}[X \geq t | X \geq \delta]$, consider the random variable L defined as the lowest integer such that the partial sum $\sum_{k=1}^{L} Y_k$ is already at least δ. Then

$$\begin{aligned}\mathbb{P}[X \geq t | X \geq \delta] &= \sum_{l=1}^{K} \mathbb{P}[X \geq t | L = l]\, \mathbb{P}[L = l | X \geq \delta] \\ &\leq \max_l \mathbb{P}[X \geq t | L = l] \\ &= \max_l \mathbb{P}\left[\sum_{k=1}^{l-1} Y_k + Y_l + \sum_{k=l+1}^{K} Y_k \geq t | L = l\right]. \end{aligned} \tag{D.13}$$

Recall that each $Y_k \in [0,1]$. Also, for $L = l$, by the definition we have $\sum_{k=1}^{l-1} Y_k < \delta < 1$. Hence,

$$\sum_{k=1}^{l-1} Y_k + Y_l \leq 2. \tag{D.14}$$

This together with Eq. (D.13) implies

$$\begin{aligned}\mathbb{P}[X \geq t | X \geq \delta] &\overset{\text{from Eq. (D.13)}}{\leq} \max_l \mathbb{P}\left[\sum_{k=l+1}^{K} Y_k \geq t - \sum_{k=1}^{l-1} Y_k - Y_l | L = l\right] \\ &\overset{\text{from Eq. (D.14)}}{\leq} \max_l \mathbb{P}\left[\sum_{k=l+1}^{K} Y_k \geq t - 2 | L = l\right] \\ &\leq \mathbb{P}[X \geq t-2]. \end{aligned} \tag{D.15}$$

From the assumption given in the statement of the lemma, it holds that $\mathbb{E}[X] \leq \delta/3 < 1$ (we may contrain δ to be less than 3). By Chernoff bound (Theorem 6.5.6Item (b)) and taking into account that X is a sum of random variables in $[0,1]$, we obtain

$$\mathbb{P}[X \geq t-2] \overset{\text{from } \mathbb{E}[X] < 1}{\leq} \mathbb{P}[X \geq \mathbb{E}[X] + t - 3] \leq \exp\left(-\frac{t-3}{3}\right).$$

From the last chain of inequalities and Eq. (D.12) we derive

$$\mathbb{P}[\mathcal{E}_2] \le cp \cdot \exp\left(-\frac{t-3}{3}\right). \tag{D.16}$$

Upper-bound on $\mathbb{P}[\mathcal{E}_3]$. Consider $\mathcal{E}_3$ as the union of the subevents $\mathcal{E}_3(i^*)$ when X_{i*} is specifically greater than δ and less than $0.1 \cdot c$ of the rest of the X_i's are below $2\delta/3$.

$$\mathbb{P}[\mathcal{E}_3] \le \sum_{i*=1}^{c} \mathbb{P}[\mathcal{E}_3(i*)] = c\mathbb{P}[\mathcal{E}_3(1)] = cp\mathbb{P}[|\{i>1 : X_i \le 2\delta/3\}| < 0.1 \cdot c]. \tag{D.17}$$

Note that by Markov's inequality we have $\mathbb{P}[X_i \le 2\delta/3] \ge 1/2$ (since $\mathbb{P}[X_i \ge 2\delta/3] \le 1/2$). Therefore,

$$\mathbb{E}[|\{i>1 : X_i \le 2\delta/3\}|] \ge (c-1)/2.$$

Hence, by Chernoff bound (Theorem 6.5.6Item (c)) we derive

$$\mathbb{P}[|\{i>1 : X_i \le 2\delta/3\}| \le 0.1 \cdot c] \le \exp\left(\frac{(3/4)^2(c-1)/2}{2}\right) \le \exp\left(-\frac{c}{8}\right).$$

assuming that $c \ge 10$. This bound together with Eq. (D.17) implies

$$\mathbb{P}[\mathcal{E}_3] \le cp\exp\left(-\frac{c}{8}\right). \tag{D.18}$$

Combining all the bounds. From Eq. (D.11), Eq. (D.16) and Eq. (D.18) we conclude

$$\begin{aligned}\mathbb{P}\left[\overline{X} \ge \delta\right] &\le \mathbb{P}[\mathcal{E}_1] + \mathbb{P}[\mathcal{E}_2] + \mathbb{P}[\mathcal{E}_3] \\ &\le c^2 p\exp\left(-\frac{c\delta}{9}\right) + cp\exp\left(-\frac{t-3}{3}\right) + cp\exp\left(-\frac{c}{8}\right) \\ &\le p/2,\end{aligned}$$

when δ and c are set appropriately. Indeed recalling that $t = \frac{c\delta}{30}$ and set $c \ge 2000\log(1/\delta)/\delta$ to achieve this goal. Notice that these bounds are not tight.

□

D.3 Proofs omitted from Section 6.7

Proof of Lemma 6.7.10: Recall the definitions of I_e and T_e from the proof of Lemma 6.7.3: Let I_e be the indicator variable of e being explored when Algorithm 23 is called from e_0; let T_e be the size of the exploration tree from e in H_i. Let $T = T_{e_0}$, $t(\lambda) = t_{e_0}(\lambda)$. Let $v(\lambda) = \mathbb{E}(T^2 | r(e_0) = \lambda)$; we will derive a recursive formula for $v(\lambda)$ and prove that $\sup_\lambda v(\lambda) \le 10d^5$, thus proving the lemma. Recall further from the proof of Lemma 6.7.3 our formula for T

$$T = 1 + \sum_{e\in\delta(e_0)} I_e \cdot T_e.$$

Therefore,

$$\begin{aligned}
v(\lambda) &= \mathbb{E}\left(1 + \sum_{e\in\delta(e_0)} I_e \cdot T_e \middle| r(e_0) = \lambda\right)^2 \\
&= 1 + 2\mathbb{E}\left(\sum_{e\in\delta(e_0)} I_e \cdot T_e \middle| r(e_0) = \lambda\right) + \mathbb{E}\left(\sum_{e\in\delta(e_0)} \sum_{f\in\delta(e_0)} I_e \cdot I_f \cdot T_e \cdot T_f \middle| r(e_0) = \lambda\right) \\
&\le 2\mathbb{E}\left(1 + \sum_{e\in\delta(e_0)} I_e \cdot T_e \middle| r(e_0) = \lambda\right) + \mathbb{E}\left(\sum_{e\ne f} I_e \cdot I_f \cdot T_e \cdot T_f \middle| r(e_0) = \lambda\right) + \mathbb{E}\left(\sum_{e\in\delta(e_0)} I_e \cdot T_e^2 \middle| r(e_0) = \lambda\right).
\end{aligned}$$

The first term is simply $2\mathbb{E}(T|r(e_0) = \lambda) = 2t(\lambda)$ and is therefore bounded by $4d$, due to Corollary 6.7.4.

To bound the second term, we drop the I_e and I_f. Then we note that T_e and T_f are independent, as they depend only on H_e and H_f respectively.

$$\begin{aligned}
\mathbb{E}\left(\sum_{e\ne f} I_e \cdot I_f \cdot T_e \cdot T_f \middle| r(e_0) = \lambda\right) &\le \sum_{e\ne f} \mathbb{E}(T_e \cdot T_f | r(e_0) = \lambda) \\
&= \sum_{e\ne f} \mathbb{E}T_e \cdot \mathbb{E}T_f \\
&\le d(d-1)\cdot(\mathbb{E}T)^2 \\
&\le 4d^4,
\end{aligned}$$

again by Corollary 6.7.4.

The third term does not admit to an outright bound. However we can express it recursively in terms of $v(\mu)$. Note, as in the proof of Lemma 6.7.3, that I_e and T_e are independent when conditioned on the rank of e.

$$\begin{aligned}
\mathbb{E}\left(\sum_{e\in\delta(e_0)} I_e \cdot T_e^2 \middle| r(e_0) = \lambda\right) &= \sum_{e\in\delta(e_0)} \int_0^\lambda \mathbb{E}(I_e \cdot T_e^2 | r(e) = \mu)\, d\mu \\
&= \sum_{e\in\delta(e_0)} \int_0^\lambda \mathbb{E}(I_e | r(e) = \mu) \cdot \mathbb{E}(T_e^2 | r(e) = \mu)\, d\mu \\
&= d \int_0^\lambda x^{-1}(\mu) v(\mu)\, d\mu.
\end{aligned}$$

Therefore, the full recursive inequality for $v(\lambda)$ is

$$v(\lambda) \le 4d + 4d^4 + d\int_0^\lambda x^{-1}(\mu)v(\mu)\,d\mu \le 5d^4 + d\int_0^\lambda x^{-1}(\mu)v(\mu)\,d\mu,$$

for $d \ge 5$. This is very similar for to the recursive formula for $t(\lambda)$ seen in the proof of Lemma 6.7.3. Let

$\tilde{v}(\lambda) = v(\lambda)/(5d^4)$. Then

$$v(\lambda) \le 1 + d\int_0^\lambda x^{-1}(\mu)v(\mu)d\mu.$$

This is now identical to the formula for $t(\lambda)$ (with an inequality instead of the equality), so $\tilde{v}(\lambda) \le t(\lambda)$ by Grönwall's inequality, since $x^{-1}(\mu) \ge 0$. So $v(\lambda) \le 5d^4 t(\lambda) \le 10d^5$ as claimed. □

Proof of Corollary 6.7.11: Let $T = T_{e_0}$ and $T_i = T_{e^{(i)}}$.

$$\begin{aligned}
\mathbb{E}\left[T^2\right] &\le \mathbb{E}\left[(1+\sum_{i=1}^{\epsilon d} T_i)^2\right]\\
&= 1 + 2\mathbb{E}\left[\sum_{i=1}^{\epsilon d} T_i\right] + \mathbb{E}\left[\sum_{i\neq j}^{\epsilon d} T_i \cdot T_j\right] + \mathbb{E}\left[\sum_{i=1}^{\epsilon d} T_i^2\right]\\
&\le 1 + 2\epsilon d \cdot \mathbb{E}[T_1] + (\epsilon d)^2 \mathbb{E}[T_1]^2 + \epsilon d \cdot \mathbb{E}\left[T_1^2\right]\\
&\le 1 + 4\epsilon d^2 + 4\epsilon^2 d^4 + 10\epsilon d^6,
\end{aligned}$$

by Corollary 6.7.4 and Lemma 6.7.10. This can then be upper bounded by $11\epsilon d^6$ for $d \ge 5$. □

D.4 Details omitted from Section 6.8

D.4.1 Proof of Theorem 6.8.2

We now provide the formal analysis of the total variation distance between $m^{1-\epsilon}$ edge-samples from graphs sampled from our hard distributions $\mathscr{D}^{YES}$ and $\mathscr{D}^{NO}$.

Proof of Theorem 6.8.2: We begin by defining random variables A_1, A_2, B_1, and B_2 that contain partial information about the iid stream under the YES and NO cases respectively. Let A_i be a random variable in $\mathscr{A} \equiv ([r] \cup \{\star\})^{m^{1-\epsilon}} \times \mathbb{N}^r$, where the j^{th} coordinate of the first half of A_i (the part in $([r] \cup \{\star\})^{m^{1-\epsilon}}$) signifies which gadget (if any) the j^{th} edge of the stream belongs to, the coordinate being $\star$ if it belongs to the clique. The j^{th} coordinate of the second half of A_i (the part in $\mathbb{N}^r$) signifies the number of *distinct* edges from $V_j \times V_j$ sampled throughout the stream. Furthermore, let B_i be a vector of length $r+1$, where the j^{th} coordinate signifies the isomorphism class of sampled edges of the j^{th} gadget and the last coordinate signifies the isomorphism class of the subsampled clique. Let the support of B_i be $\mathscr{B}$

With slight abuse of notation, for $i \in \{1,2\}$ let

$$\begin{aligned}
p_i(a,b,c) &:= \mathbb{P}\left[A_i = a \wedge B_i = b \wedge C_i = c\right]\\
p_i(a) &:= \mathbb{P}\left[A_i = a\right]\\
p_i(b) &:= \mathbb{P}\left[B_i = b\right]\\
p_i(c) &:= \mathbb{P}\left[C_i = c\right]\\
p_i(b|a) &:= \mathbb{P}\left[B_i = b | A_i = a\right]\\
p_i(c|a,b) &:= \mathbb{P}\left[C_i = c | A_i = a \wedge B_i = b\right].
\end{aligned}$$

Again, we are interested in the total variation distance between C_1 and C_2, which satisfies

$$\begin{aligned}\|C_1 - C_2\|_{\mathrm{TV}} &\le \|(A_1, B_1, C_1) - (A_2, B_2, C_2)\|_{\mathrm{TV}} \\ &= \frac{1}{2} \sum_{(a,b,c)\in\mathscr{A}\times\mathscr{B}\times\mathscr{C}} |p_1(a,b,c) - p_2(a,b,c)| \\ &= \frac{1}{2} \sum_{(a,b,c)\in\mathscr{A}\times\mathscr{B}\times\mathscr{C}} |p_1(a)p_1(b|a)p_1(c|a,b) - p_2(a)p_2(b|a)p_2(c|a,b)|\end{aligned}$$

First, observe that there is no discrepancy between $p_1(a)$ and $p_2(a)$ as the distributions of A_1 and A_2 are identical. Notice that the probability of a given iid edge being in a specific gadget or in the clique depends only on the number of edges of that gadget or the number of edges of the cliques. The clique contains $\binom{w}{2}$ edges in both the YES and NO cases. Also G and H have the same number of edges (simply apply the guarantee of Theorem 6.8.3 with K being a single edge), so all gadgets have the same number of edges as well. $p_1(a) = p_2(a) =: p(a)$.

$$\begin{aligned}\|C_1 - C_2\|_{\mathrm{TV}} &= \frac{1}{2} \sum_{a\in\mathscr{A}} p(a) \sum_{(b,c)\in\mathscr{B}\times\mathscr{C}} |p_1(b|a)p_1(c|a,b) - p_2(b|a)p_2(c|a,b)| \\ &\le \mathbb{P}[\mathscr{E}] + \frac{1}{2} \sum_{a\in\mathscr{A}'} p(a) \sum_{(b,c)\in\mathscr{B}\times\mathscr{C}} |p_1(b|a)p_1(c|a,b) - p_2(b|a)p_2(c|a,b)|\end{aligned}$$

where $\mathscr{A}'$ is the set of outcomes of A_i in accordance with $\overline{\mathscr{E}}$. Recall that

$$\mathscr{E} \equiv \{\exists i \in [r] : \text{edges between vertices of } V_i \text{ appear more than } k \text{ times in the stream}\}.$$

Consider now the discrepancy between $p_1(b|a)$ and $p_2(b|a)$. Again, we will prove that the two distributions are equivalent, as long as the value of A_i being conditioned on is in $\mathscr{A}'$. Consider B_i to be $\left(B_i^{(1)}, B_i^{(2)}, \ldots, B_i^{(r)}, B_i^*\right)$, where $B_i^{(j)}$ represents the isomorphism class of the sampled version of the jth gadget and B_i^* is the isomorphism class of the sampled version of the clique. Note that the coordinates of B_i are independent conditioned on an outcome of A_i. Clearly, the distributions of B_1^* and B_2^* are identical. Consider now the distributions of $B_1^{(j)}$ and $B_2^{(j)}$ conditioned on $A_1 = A_2 = a \in \mathscr{A}'$. Conditioning on an outcome in $\mathscr{A}'$ fixes the size of the sampled subgraph to some $l \le k$, which means the support of $p_i(b|a)$ is some set of graphs of size l. For any specific graph K in the support, we know that the number of subgraphs of G and H isomorphic to K are equal (by the guarantee of Theorem 6.8.3); let this number be X. Also let the number of edges in a gadget be Y. Then $\mathbb{P}\left[B_1^{(j)} = [K] | A_1 = a\right] = \mathbb{P}\left[B_2^{(j)} = [K] | A_2 = a\right] = X/\binom{Y}{l}$. Thus $p_1(b|a) = p_2(b|a) =: p(b|a)$ for every $a \in \mathscr{A}'$.

$$\|C_1 - C_2\|_{\mathrm{TV}} \le \frac{1}{10} + \frac{1}{2} \sum_{(a,b)\in\mathscr{A}'\times\mathscr{B}} p(a)p(b|a) \sum_{c\in\mathscr{C}} |p_1(c|a,b) - p_2(c|a,b)|$$

Finally, consider the discrepancy between $p_1(c|a,b)$ and $p_2(c|a,b)$. We will, yet again, prove that the two distributions are identical when conditioned on any $(a,b) \in \mathscr{A}' \times \mathscr{B}$. Having conditioned on $A_i = a$ and $B_i = b$ the following are set about the stream: for every gadget, as well as the clique, we know the placement and number of the edges in the stream, and we know the isomorphidm class of the subsampled gadget (or clique). For every gadget (or clique) with subsampled isomorphism-class $[K]$, we don't know the particular embedding of K into V_j (or V_K) that produces the subsampled gadget (or clique), and we also don't know the order and multiplicity with which these edges arrive. Thanks to the fact that all gadgets were uniformly randomly permuted in their embedding into V in the construction of $\mathscr{D}^{\text{YES}}$ and $\mathscr{D}^{\text{NO}}$, the embedding of K into V_j is also uniformly random. (The clique is completely symmetric and need not be permuted.) Furthermore, since the stream is iid, conditioned on the set of edges in $V_j \times V_j$ that must appear, their order and multiplicity is drawn from the same distribution, regardless of whether we are in the YES or NO case. Therefore, for any $(a,b,c) \in \mathscr{A}' \times \mathscr{B} \times \mathscr{C}$, $p_1(c|a,b) = p_2(c|a,b) =: p(c|a,b)$.

$$\|C_1 - C_2\|_{\text{TV}} \le \frac{1}{10} + \frac{1}{2} \sum_{(a,b)\in\mathscr{A}'\times\mathscr{B}} p(a)p(b|a) \sum_{c\in\mathscr{C}} |p(c|a,b) - p(c|a,b)| = \frac{1}{10}$$

□

D.4.2 Proof of Lemma 6.8.23

Our proof of Lemma 6.8.23 is built on Theorem 3.4. of Lubotzky et al. (1988) and a result from Cullinan and Hajir (2012). We next restate the first result.

Theorem D.4.1 (Lubotzky et al. (1988))**.** *For any distinct primes p and q congruent to 1 modulo 4, there exists a group $\mathscr{G}^{p,q}$ with a set S of generator elements with the following properties: $|\mathscr{G}^{p,q}| \in [q(q^2-1)/2, q(q^2-1)]$; $|S| = p+1$; and, $\mathscr{G}^{p,q}$ has girth at least $2\log_p(q/4)$.*

Theorem D.4.2 (Cullinan and Hajir (2012))**.** *For any $x \ge 7$, the interval $(x, 2x]$ contains a prime number congruent 1 modulo 4.*

Lemma 6.8.23. *For any parameters g and l, there exists a group $\mathscr{G}$ of size $l^{O(g)}$ along with a set of generator elements S of size at least l, such that the associated Cayley graph (Definition 6.8.18) has girth at least g.*

Proof. If $l < 7$, let $p = 13$. Otherwise, if $l \ge 7$, let p be a prime number congruent 1 modulo 4 from the interval $[l, 2l]$. By Theorem D.4.2, such p exists. Let q be a prime number congruent 1 modulo 4 from the interval $[4p^g, 8p^g]$. Again by Theorem D.4.2 and recalling that $p \ge 2$, such q exists. The statement now follows by Theorem D.4.1. □

D.5 Proofs omitted from Section 6.9

Proof of Lemma 6.9.4: By symmetry we may assume that $p \le 1/2$. We consider 3 cases:

Case 1: $\epsilon \le -p/3$. With this constraint

$$D_{KL}\left(\text{Ber}\left(p+\epsilon\right) \| \text{Ber}\left(p\right)\right) \le D_{KL}\left(\text{Ber}\left(0\right) \| \text{Ber}\left(p\right)\right) = \log\left(\frac{1}{1-p}\right) \le \frac{p}{1-p}.$$

Therefore the lemma statement is always satisfied.

Case 2: $\epsilon \geq 1/4$**.** With this constraint $D_{KL}\left(\text{Ber}\left(p+\epsilon\right)\|\text{Ber}\left(p\right)\right) \leq D_{KL}\left(\text{Ber}\left(1\right)\|\text{Ber}\left(p\right)\right) = \log\left(\frac{1}{p}\right) \leq \frac{1}{p}$. Therefore, the lemma statement is always satisfied.

Case 3: $\epsilon \in [-p/3, 1/4]$**.** In that case we have

$$\begin{aligned}
D_{KL}\left(\text{Ber}\left(p+\epsilon\right)\|\text{Ber}\left(p\right)\right) &= -(p+\epsilon)\log\left(\frac{p}{p+\epsilon}\right) - (1-p-\epsilon)\log\left(\frac{1-p}{1-p-\epsilon}\right) && \text{(D.19)}\\
&= -(p+\epsilon)\log\left(1-\frac{\epsilon}{p+\epsilon}\right) - (1-p-\epsilon)\log\left(1+\frac{\epsilon}{1-p-\epsilon}\right) && \text{(D.20)}\\
&\leq -(p+\epsilon)\left(-\frac{\epsilon}{p+\epsilon} - \frac{4\epsilon^2}{(p+\epsilon)^2}\right) - (1-p-\epsilon)\left(\frac{\epsilon}{1-p-\epsilon} - \frac{4\epsilon^2}{(1-p-\epsilon)^2}\right) && \text{(D.21)}\\
&= \frac{4\epsilon^2}{(p+\epsilon)(1-p-\epsilon)} && \text{(D.22)}\\
&\leq \frac{16}{p(1-p)} && \text{(D.23)}
\end{aligned}$$

Here Eq. (D.21) follows from Taylor's theorem. Indeed, By the restriction on the range of ϵ, both $-\epsilon/(p+\epsilon)$ and $\epsilon/(1-p-\epsilon)$ are in the interval $[-1/2,\infty)$. On this interval the function $\log(1+x)$ is twice differentiable and the absolute value of its second derivative is bounded by 4, therefore

$$x - 4x^2 \leq \log(1+x) \leq x + 4x^2.$$

□

Proof of Lemma 6.9.7: We assume without loss of generality that $r \leq 1/2$. Let $\tilde{r} := \textsc{Padding}(r,\epsilon)$. Then $\tilde{r}$ is also less than half and in fact $\tilde{r} = \max(r,\epsilon)$. Let $\eta_1 = |p-q|$, $\eta_2 = |q-\tilde{r}|$ and $\eta_3 = |p-\tilde{r}|$. For simplicity we will denote $D_{KL}\left(\text{Ber}\left(x\right)\|\text{Ber}\left(y\right)\right)$ as $D_{KL}\left(x\|y\right)$ during this proof. By Lemma 6.9.4, in order to establish the result of the lemma it suffices to show that

$$\eta_3^2 \leq O(\epsilon)\tilde{r}(1-\tilde{r}). \tag{D.24}$$

Note that the term $(1-\tilde{r})$ is in $[1/2,1]$ and can be disregarded.

We will use the following facts throughout the proof:

Fact D.5.1. *For all* $x \in \mathbb{R}$,

$$\log(1+x) \leq x.$$

Fact D.5.2. *For all* $x \leq 1$,

$$\log(1+x) \leq x - \frac{x^2}{4}.$$

Fact D.5.3. *For all $x \in [0, 1/2]$,*

$$D_{KL}(0\|x) \le 2x.$$

Fact D.5.4. *For all $x \in [0, 1/2]$,*

$$D_{KL}(x\|2x) \ge \frac{x}{4}.$$

Indeed,

$$D_{KL}(x\|2x) = -x\log\left(\frac{2x}{x}\right) - (1-x)\log\left(1 - \frac{x}{1-x}\right) \ge -x\log 2 + (1-x)\cdot\frac{x}{1-x} = x\cdot(1-\log 2) \ge \frac{x}{4},$$

by Fact D.5.1.

We will differentiate six cases depending on the ordering of p, q and $\tilde{r}$. However, four of these, the ones where q is not in the middle, will be very simple.

Case 1: $p \le \tilde{r} \le q$**.**Then,

$$D_{KL}\left(p\|\tilde{r}\right) \le D_{KL}\left(p\|q\right) \le \epsilon.$$

Case 2: $\tilde{r} \le p \le q$**.**Then,

$$D_{KL}\left(p\|\tilde{r}\right) \le D_{KL}\left(q\|\tilde{r}\right) \le D_{KL}\left(q\|r\right) \le \epsilon.$$

Case 3: $\tilde{q} \le p \le \tilde{r}$**.**Then, if $\tilde{r} = r$,

$$D_{KL}\left(p\|\tilde{r}\right) \le D_{KL}\left(q\|\tilde{r}\right) = D_{KL}\left(q\|r\right) \le \epsilon.$$

On the other hand, if $\tilde{r} = \epsilon$,

$$D_{KL}\left(p\|\tilde{r}\right) \le D_{KL}(0\|\epsilon) = -\log(1-\epsilon) \le 2\epsilon,$$

by Fact D.5.3 since $\epsilon \le 1/2$.

Case 4: $\tilde{q} \le \tilde{r} \le p$**.**Then,

$$D_{KL}\left(p\|\tilde{r}\right) \le D_{KL}\left(p\|q\right) \le \epsilon.$$

Case 5: $p \le q \le \tilde{r}$**.**We consider two subcases.

(a.) $p \le 4\epsilon$**.** Then q cannot be greater than 8ϵ. Indeed this would mean by Fact D.5.4 that

$$D_{KL}\left(p\|q\right) > D_{KL}(4\epsilon\|8\epsilon) \ge \epsilon,$$

which is a contradiction. Similarly, r cannot be greater than 16ϵ. Indeed this would mean by Fact D.5.4 that

$$D_{KL}\left(q \| r\right) > D_{KL}\left(8\epsilon \| 16\epsilon\right) \geq 2\epsilon,$$

which is also a contradiction. Ultaminately, $\widetilde{r} \leq 16\epsilon$, so

$$D_{KL}\left(p \| \widetilde{r}\right) \leq D_{KL}\left(0 \| 16\epsilon\right) \leq 32\epsilon$$

by Fact D.5.3 since $16\epsilon \leq 1/2$.

(b.) $p \geq 4\epsilon$**.** Note that $\widetilde{r} = r$. Let us bound η_1. First note that η_1 cannot be greater than p due to Fact D.5.4. We will further show that η_1 in fact cannot be greater than $2\sqrt{p\epsilon}$.

$$\begin{aligned}
\epsilon &\geq D_{KL}\left(p \| q\right) \\
&= -p\log\left(1+\frac{\eta_1}{p}\right) - (1-p)\log\left(1-\frac{\eta_1}{1-p}\right) \\
&\geq -p\left(\frac{\eta_1}{p} - \frac{\eta_1^2}{4p^2}\right) - (1-p)\left(-\frac{\eta_1}{1-p}\right) && \text{By Facts D.5.1 and D.5.2, since } \eta_1/p \leq 1, \\
&= \frac{\eta_1^2}{4p}.
\end{aligned}$$

An identical calculation shows that $\eta_2 \leq 2\sqrt{q\epsilon}$. Ultimately,

$$\begin{aligned}
\eta_3 &= \eta_1 + \eta_2 \\
&\leq 2\sqrt{p\epsilon} + 2\sqrt{q\epsilon} \\
&\leq 2\sqrt{p\epsilon} + 2\sqrt{\left(p + 2\sqrt{p\epsilon}\right)\cdot\epsilon} \\
&\leq 6\sqrt{p\epsilon} && \text{Since } p \geq \epsilon, \\
&\leq 6\sqrt{\widetilde{r}\epsilon}.
\end{aligned}$$

From here Eq. (D.24) follows immediately.

Case 6: $\widetilde{r} \leq q \leq p$**.**In this case, let us first bound η_2. We will show that η_2 cannot be greater than $2\sqrt{q\epsilon}$.

$$\begin{aligned}
\epsilon &\geq D_{KL}\left(q \| r\right) \\
&\geq D_{KL}\left(q \| \widetilde{r}\right) \\
&= -q\log\left(1-\frac{\eta_2}{q}\right) - (1-q)\log\left(1+\frac{\eta_2}{1-q}\right) \\
&\geq -q\left(-\frac{\eta_2}{q} - \frac{\eta_2^2}{4q^2}\right) - (1-q)\left(\frac{\eta_2}{1-q}\right) && \text{By Facts D.5.1 and D.5.2,} \\
&= \frac{\eta_2^2}{4q}.
\end{aligned}$$

This also implies that q is at most $6\widetilde{r}$. Indeed, suppose $q = \gamma\widetilde{r}$. Then

$$\begin{aligned}\widetilde{r} &= q - \eta_2\\ &\geq q - 2\sqrt{q\epsilon}\\ &= \gamma\widetilde{r} - 2\sqrt{\gamma\widetilde{r}\epsilon}\\ &\geq (\gamma - 2\sqrt{\gamma})\cdot\widetilde{r},\end{aligned}$$

since $\widetilde{r} \geq \epsilon$. Therefore, $1 \geq \gamma - 2\sqrt{\gamma}$, so $\gamma \leq 6$. We conclude that $\eta_2 \leq 2\sqrt{6\widetilde{r}\epsilon}$. An identical calculation shows that $\eta_1 \leq 2\sqrt{6q\epsilon} \leq 12\sqrt{\widetilde{r}\epsilon}$. Ultimately,

$$\begin{aligned}\eta_3 &= \eta_1 + \eta_2\\ &\leq 2\sqrt{6\widetilde{r}\epsilon} + 12\sqrt{\widetilde{r}\epsilon}\\ &\leq 18\sqrt{\widetilde{r}\epsilon}.\end{aligned}$$

From here Eq. (D.24) follows immediately.

This concludes the proof of the lemma under all possible orderings of p, q and $\widetilde{r}$. $\square$

Bibliography

Abraham, I., Durfee, D., Koutis, I., Krinninger, S., and Peng, R. (2016). On fully dynamic graph sparsifiers. In *57th Annual Symposium on Foundations of Computer Science*, pages 335–344.

Achlioptas, D. (2003). Database-friendly random projections: Johnson-lindenstrauss with binary coins. *J. Comput. Syst. Sci.*, 66(4):671–687.

Agarwal, A. and Chakrabarti, S. (2007). Learning random walks to rank nodes in graphs. In *Proceedings of the 24th international conference on Machine learning*, pages 9–16.

Ahn, K., Guha, S., and McGregor, A. (2012a). Analyzing graph structure via linear measurements. In *Proceedings of the 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 459–467.

Ahn, K., Guha, S., and McGregor, A. (2012b). Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st symposium on Principles of Database Systems*, pages 5–14. ACM.

Ahn, K. J. and Guha, S. (2011a). Laminar families and metric embeddings: Non-bipartite maximum matching problem in the semi-streaming model. *arXiv preprint arXiv:1104.4058*.

Ahn, K. J. and Guha, S. (2011b). Linear programming in the semi-streaming model with application to the maximum matching problem. In *International Colloquium on Automata, Languages, and Programming*, pages 526–538. Springer.

Ahn, K. J., Guha, S., and McGregor, A. (2013). Spectral sparsification in dynamic graph streams. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 1–10. Springer.

Alev, V. L., Anari, N., Lau, L. C., and Gharan, S. O. (2017). Graph clustering using effective resistance. *arXiv preprint arXiv:1711.06530*.

Alon, N. and Klartag, B. (2017). Optimal compression of approximate inner products and dimension reduction. In *Proceedings of the IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 639–650.

Alon, N., Matias, Y., and Szegedy, M. (1999). The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147.

Alon, N., Rubinfeld, R., Vardi, S., and Xie, N. (2012). Space-efficient local computation algorithms. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1132–1139. Society for Industrial and Applied Mathematics.

Alon, N. and Spencer, J. H. (2008). *The Probabilistic Method, Third Edition*. Wiley-Interscience series in discrete mathematics and optimization. Wiley.

Alvisi, L., Clement, A., Epasto, A., Lattanzi, S., and Panconesi, A. (2014). Communities, random walks, and social sybil defense. *Internet Math.*, 10(3-4):360–420.

Andersen, R., Chung, F., and Lang, K. (2006). Local graph partitioning using pagerank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 475–486.

Andoni, A., Chen, J., Krauthgamer, R., Qin, B., Woodruff, D. P., and Zhang, Q. (2016). On sketching quadratic forms. In *Proceedings of the 2016 Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 311–319.

Andoni, A. and Indyk, P. (2006). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 459–468.

Andoni, A., Indyk, P., Nguyen, H. L., and Razenshteyn, I. P. (2014). Beyond locality-sensitive hashing. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1018–1028.

Andoni, A. and Razenshteyn, I. P. (2015). Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 793–801.

Assadi, S., Bateni, M., Bernstein, A., Mirrokni, V. S., and Stein, C. (2019a). Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1616–1635.

Assadi, S. and Bernstein, A. (2019). Towards a unified theory of sparsification for matching problems. In *2nd Symposium on Simplicity in Algorithms, SOSA 2019, January 8-9, 2019, San Diego, CA, USA*, volume 69 of *OASIcs*, pages 11:1–11:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Assadi, S. and Khanna, S. (2017). Randomized composable coresets for matching and vertex cover. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2017, Washington DC, USA, July 24-26, 2017*, pages 3–12.

Assadi, S., Khanna, S., and Li, Y. (2017). On estimating maximum matching size in graph streams. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1723–1742.

Assadi, S., Khanna, S., Li, Y., and Yaroslavtsev, G. (2016). Maximum matchings in dynamic graph streams and the simultaneous communication model. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1345–1364.

Assadi, S., Sun, X., and Weinstein, O. (2019b). Massively parallel algorithms for finding well-connected components in sparse graphs. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 461–470.

Badanidiyuru, A., Mirzasoleiman, B., Karbasi, A., and Krause, A. (2014). Streaming submodular maximization: massive data summarization on the fly. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 671–680. ACM.

Badanidiyuru, A. and Vondrák, J. (2013). Fast algorithms for maximizing submodular functions. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1497–1514.

Bahmani, B., Chakrabarti, K., and Xin, D. (2011). Fast personalized pagerank on mapreduce. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 973–984.

Balcan, M., Ehrlich, S., and Liang, Y. (2013). Distributed k-means and k-median clustering on general communication topologies. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 1995–2003.

Bansal, N., Svensson, O., and Trevisan, L. (2019). New notions and constructions of sparsification for graphs and hypergraphs. In *Proceedings of the IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 910–928.

Bateni, M., Bhaskara, A., Lattanzi, S., and Mirrokni, V. S. (2014). Distributed balanced clustering via mapping coresets. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2591–2599.

Batson, J. D., Spielman, D. A., and Srivastava, N. (2012). Twice-Ramanujan sparsifiers. *SIAM Journal on Computing*, 41(6):1704–1721.

Batson, J. D., Spielman, D. A., Srivastava, N., and Teng, S. (2013). Spectral sparsification of graphs: theory and algorithms. *Commun. ACM*, 56(8):87–94.

Beame, P., Gharan, S. O., and Yang, X. (2018). Time-space tradeoffs for learning finite functions from random evaluations, with applications to polynomials. In *Conference On Learning Theory, COLT 2018, Stockholm, Sweden, 6-9 July 2018.*, pages 843–856.

Beame, P., Koutris, P., and Suciu, D. (2013). Communication steps for parallel query processing. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, pages 273–284. ACM.

Benczúr, A. A. and Karger, D. R. (1996). Approximating S-T minimum cuts in $\tilde{O}(n^2)$ time. In *38th Annual Symposium on Theory of Computing*, pages 47–55.

Benczúr, A. A. and Karger, D. R. (2015). Randomized approximation schemes for cuts and flows in capacitated graphs. *SIAM Journal on Computing*, 44(2):290–319.

Benson, A. R., Gleich, D. F., and Leskovec, J. (2016). Higher-order organization of complex networks. *CoRR*, abs/1612.08447.

Berkhin, P. (2007). Bookmark-coloring approach to personalized pagerank computing. Technical report, Internet Mathematics.

Bernstein, A., Holm, J., and Rotenberg, E. (2018). Online bipartite matching with amortized replacements. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 947–959. SIAM.

Bhattacharya, S., Henzinger, M., Nanongkai, D., and Tsourakakis, C. E. (2015). Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 173–182. ACM.

Bollobás, B. and Frieze, A. M. (1985). On matchings and hamiltonian cycles in random graphs. In *Random Graphs '83*, volume 118 of *North-Holland Mathematics Studies*, pages 23 – 46. North-Holland.

Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Comput. Networks*, 30(1-7):107–117.

Bulteau, L., Froese, V., Kutzkov, K., and Pagh, R. (2016). Triangle counting in dynamic graph streams. *Algorithmica*, 76(1):259–278.

Bury, M., Grigorescu, E., McGregor, A., Monemizadeh, M., Schwiegelshohn, C., Vorotnikova, S., and Zhou, S. (2019). Structural results on matching estimation with applications to streaming. *Algorithmica*, 81(1):367–392.

Bury, M. and Schwiegelshohn, C. (2015). Sublinear estimation of weighted matchings in dynamic data streams. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 263–274.

Carlson, C., Kolla, A., Srivastava, N., and Trevisan, L. (2019). Optimal lower bounds for sketching graph cuts. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2565–2569.

Chan, T.-H. H., Louis, A., Tang, Z. G., and Zhang, C. (2018). Spectral properties of hypergraph Laplacian and approximation algorithms. *Journal of the ACM*, 65(3):1–48.

Charikar, M., Chen, K. C., and Farach-Colton, M. (2004). Finding frequent items in data streams. *Theor. Comput. Sci.*, 312(1):3–15.

Chen, Y., Khanna, S., and Nagda, A. (2020). Near-linear size hypergraph cut sparsifiers. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 61–72. IEEE.

Cheng, D., Cheng, Y., Liu, Y., Peng, R., and Teng, S. (2015). Spectral sparsification of random-walk matrix polynomials. *CoRR*, abs/1502.03496.

Cheng, Y., Panigrahi, D., and Sun, K. (2020). Sparsification of balanced directed graphs. *CoRR*, abs/2006.01975.

Chiplunkar, A., Kapralov, M., Khanna, S., Mousavifar, A., and Peres, Y. (2018). Testing graph clusterability: Algorithms and lower bounds. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 497–508. IEEE.

Chitnis, R., Cormode, G., Esfandiari, H., Hajiaghayi, M., McGregor, A., Monemizadeh, M., and Vorotnikova, S. (2016). Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1326–1344.

Chitnis, R. H., Cormode, G., Esfandiari, H., Hajiaghayi, M., McGregor, A., Monemizadeh, M., and Vorotnikova, S. (2015). Kernelization via sampling with applications to dynamic graph streams. *CoRR*, abs/1505.01731.

Chu, T., Gao, Y., Peng, R., Sachdeva, S., Sawlani, S., and Wang, J. (2018). Graph sparsification, spectral sketches, and faster resistance computation, via short cycle decompositions. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 361–372.

Chung, F. and Simpson, O. (2015). Distributed algorithms for finding local clusters using heat kernel pagerank. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 177–189. Springer.

Cohen, M. B., Kyng, R., Miller, G. L., Pachocki, J. W., Peng, R., Rao, A. B., and Xu, S. C. (2014). Solving SDD linear systems in nearly $m\log^{1/2} n$ time. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 343–352.

Cohen, M. B., Musco, C., and Pachocki, J. (2016). Online row sampling. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2016)*.

Cormode, G. and Firmani, D. (2014). A unifying framework for ℓ_0-sampling algorithms. *Distributed and Parallel Databases*, 32(3):315–335. Preliminary version in ALENEX 2013.

Cormode, G., Jowhari, H., Monemizadeh, M., and Muthukrishnan, S. (2017). The sparse awakens: Streaming algorithms for matching size estimation in sparse graphs. In *25th Annual European Symposium on Algorithms, ESA 2017, September 4-6, 2017, Vienna, Austria*, pages 29:1–29:15.

Cullinan, J. and Hajir, F. (2012). Primes of prescribed congruence class in short intervals. *Integers*, 12:A56.

Czumaj, A., Lacki, J., Madry, A., Mitrovic, S., Onak, K., and Sankowski, P. (2018). Round compression for parallel matching algorithms. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 471–484. ACM.

Czumaj, A., Peng, P., and Sohler, C. (2015). Testing cluster structure of graphs. In *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*, pages 723–732.

Das Sarma, A., Gollapudi, S., and Panigrahy, R. (2011). Estimating pagerank on graph streams. *Journal of the ACM (JACM)*, 58(3):1–19.

Das Sarma, A., Molla, A. R., Pandurangan, G., and Upfal, E. (2015). Fast distributed pagerank computation. *Theoretical Computer Science*, 561:113–121.

Das Sarma, A., Nanongkai, D., Pandurangan, G., and Tetali, P. (2013). Distributed random walks. *Journal of the ACM (JACM)*, 60(1):1–31.

Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. S. (2004). Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20th ACM Symposium on Computational Geometry, Brooklyn, New York, USA, June 8-11, 2004*, pages 253–262.

Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

Dinur, I. and Nissim, K. (2003). Revealing information while preserving privacy. In *Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 202–210.

Eggert, S., Kliemann, L., and Srivastav, A. (2009). Bipartite graph matchings in the semi-streaming model. In *European Symposium on Algorithms*, pages 492–503. Springer.

Epstein, L., Levin, A., Mestre, J., and Segev, D. (2011). Improved approximation guarantees for weighted matching in the semi-streaming model. *SIAM Journal on Discrete Mathematics*, 25(3):1251–1265.

Esfandiari, H., Hajiaghayi, M., and Monemizadeh, M. (2016). Finding large matchings in semi-streaming. In *IEEE International Conference on Data Mining Workshops, ICDM Workshops 2016, December 12-15, 2016, Barcelona, Spain.*, pages 608–614.

Esfandiari, H., Hajiaghayi, M. T., Liaghat, V., Monemizadeh, M., and Onak, K. (2015). Streaming algorithms for estimating the matching size in planar graphs and beyond. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1217–1233.

Even, G., Medina, M., and Ron, D. (2014). Deterministic stateless centralized local algorithms for bounded degree graphs. In *European Symposium on Algorithms*, pages 394–405. Springer.

Feigenbaum, J., Kannan, S., McGregor, A., Suri, S., and Zhang, J. (2005). On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2-3):207–216.

Filtser, A., Kapralov, M., and Nouri, N. (2020). Graph spanners by sketching in dynamic streams and the simultaneous communication model. *CoRR*, abs/2007.14204.

Fischer, E., Lehman, E., Newman, I., Raskhodnikova, S., Rubinfeld, R., and Samorodnitsky, A. (2002). Monotonicity testing over general poset domains. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 474—-483.

Fogaras, D. and Rácz, B. (2004). Towards scaling fully personalized pagerank. In *Algorithms and Models for the Web-Graph: Third International Workshop, WAW 2004, Rome, Italy, October 16, 2004, Proceeedings*, volume 3243 of *Lecture Notes in Computer Science*, pages 105–117. Springer.

Fujii, K., Soma, T., and Yoshida, Y. (2018). Polynomial-time algorithms for submodular Laplacian systems.

Gamlath, B., Kale, S., Mitrović, S., and Svensson, O. (2018). Weighted matchings via unweighted augmentations. *arXiv preprint arXiv:1811.02760*.

Garg, S., Raz, R., and Tal, A. (2018). Extractor-based time-space lower bounds for learning. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 990–1002.

Gargi, U., Lu, W., Mirrokni, V., and Yoon, S. (2011). Large-scale community detection on youtube for topic discovery and exploration. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 5.

Ghaffari, M. (2016). An improved distributed algorithm for maximal independent set. In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 270–277. Society for Industrial and Applied Mathematics.

Ghaffari, M., Gouleakis, T., Konrad, C., Mitrović, S., and Rubinfeld, R. (2018). Improved massively parallel computation algorithms for mis, matching, and vertex cover. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 129–138. ACM.

Ghaffari, M. and Uitto, J. (2019). Sparsifying distributed algorithms with ramifications in massively parallel computation and centralized local computation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1636–1653. SIAM.

Ghaffari, M. and Wajc, D. (2019). Simplified and space-optimal semi-streaming (2+epsilon)-approximate matching. In *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019, January 8-9, 2019 - San Diego, CA, USA*, pages 13:1–13:8.

Gibson, D., Kleinberg, J., and Raghavan, P. (2000). Clustering categorical data: an approach based on dynamical systems. *The International Journal on Very Large Data Bases*, 8(3-4):222–236.

Gleich, D. F. and Seshadhri, C. (2012). Vertex neighborhoods, low conductance cuts, and good seeds for local community methods. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 597–605.

Gluch, G., Kapralov, M., Lattanzi, S., Mousavifar, A., and Sohler, C. (2021). Spectral clustering oracles in sublinear time. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1598–1617. SIAM.

Goel, A., Kapralov, M., and Khanna, S. (2012). On the communication and streaming complexity of maximum bipartite matching. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 468–485.

Goodrich, M. T., Sitchinava, N., and Zhang, Q. (2011). Sorting, searching, and simulation in the mapreduce framework. In *International Symposium on Algorithms and Computation*, pages 374–383. Springer.

Guha, S., McGregor, A., and Tench, D. (2015). Vertex and hyperedge connectivity in dynamic graph streams. In *Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS 2015, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 241–247.

Hall, P. (1935). On representatives of subsets. *Journal of the London Mathematical Society*, s1-10(1):26–30.

Halperin, S. and Zwick, U. (1996). An optimal randomised logarithmic time connectivity algorithm for the erew pram. *Journal of Computer and System Sciences*, 53(3):395–416.

Harvey, N. J. A., Liaw, C., and Liu, P. (2018). Greedy and local ratio algorithms in the mapreduce model. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, Vienna, Austria, July 16-18, 2018*, pages 43–52. ACM.

Haveliwala, T. (2003). Topic-sensitive pagerank: a context-sensitive ranking algorithm for web search. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):784–796.

Hein, M., Setzer, S., Jost, L., and Rangapuram, S. S. (2013). The total variation on hypergraphs - learning on hypergraphs revisited. In *Advances in Neural Information Processing Systems 26 (NIPS)*, pages 2427–2435.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30.

Hoskins, J. G., Musco, C., Musco, C., and Tsourakakis, C. E. (2018). Inferring networks from random walk-based node similarities. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 3708–3719.

Huang, Y., Liu, Q., and Metaxas, D. (2009). Video object segmentation by hypergraph cut. In *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1738–1745.

Ikeda, M., Kitabeppu, Y., and Takai, Y. (2021). Coarse Ricci curvature of hypergraphs and its generalization.

Ikeda, M., Miyauchi, A., Takai, Y., and Yoshida, Y. (2019). Finding Cheeger cuts in hypergraphs via heat equation.

Ikeda, M. and Tanigawa, S. (2018). Cut sparsifiers for balanced digraphs. In *Proceedings of the 16th International Workshop on Approximation and Online Algorithms (WAOA)*, pages 277–294.

Indyk, P., Mahabadi, S., Mahdian, M., and Mirrokni, V. S. (2014). Composable core-sets for diversity and coverage maximization. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS'14, Snowbird, UT, USA, June 22-27, 2014*, pages 100–108. ACM.

Indyk, P. and Motwani, R. (1998). Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 604–613.

Isard, M., Budiu, M., Yu, Y., Birrell, A., and Fetterly, D. (2007). Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS operating systems review*, volume 41, pages 59–72. ACM.

Jambulapati, A. and Sidford, A. (2018). Efficient $\widetilde{O}(n/\epsilon)$ spectral sketches for the laplacian and its pseudoinverse. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2487–2503. SIAM.

Jeh, G. and Widom, J. (2003). Scaling personalized web search. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, page 271–279, New York, NY, USA. Association for Computing Machinery.

Jin, C. (2019). Simulating random walks on graphs in the streaming model. In *10th Innovations in Theoretical Computer Science Conference, ITCS 2019, January 10-12, 2019, San Diego, California, USA*, pages 46:1–46:15.

Jindal, G., Kolev, P., Peng, R., and Sawlani, S. (2017). Density independent algorithms for sparsifying k-step random walks. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, pages 14:1–14:17.

Jowhari, H., Sağlam, M., and Tardos, G. (2011). Tight bounds for Lp samplers, finding duplicates in streams, and related problems. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 49–58. ACM.

Kapralov, M. (2013). Better bounds for matchings in the streaming model. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 1679–1697. SIAM.

Kapralov, M., Khanna, S., and Sudan, M. (2014). Approximating matching size from random streams. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 734–751. SIAM.

Kapralov, M., Krauthgamer, R., Tardos, J., and Yoshida, Y. (2021a). Spectral hypergraph sparsifiers of nearly linear size. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1159–1170. IEEE.

Kapralov, M., Krauthgamer, R., Tardos, J., and Yoshida, Y. (2021b). Towards tight bounds for spectral sparsification of hypergraphs. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 598–611. ACM.

Kapralov, M., Lattanzi, S., Nouri, N., and Tardos, J. (2021c). Efficient and local parallel random walks. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 21375–21387.

Kapralov, M., Lee, Y. T., Musco, C. N., Musco, C. P., and Sidford, A. (2017a). Single pass spectral sparsification in dynamic streams. *SIAM Journal on Computing*, 46(1):456–477.

Kapralov, M., Maystre, G., and Tardos, J. (2021d). Communication efficient coresets for maximum matching. In *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 156–164. SIAM.

Kapralov, M., Mitrovic, S., Norouzi-Fard, A., and Tardos, J. (2020a). Space efficient approximation to maximum matching size from uniform edge samples. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1753–1772. SIAM.

Kapralov, M., Mousavifar, A., Musco, C., Musco, C., and Nouri, N. (2019a). Faster spectral sparsification in dynamic streams. *CoRR*, abs/1903.12165.

Kapralov, M., Mousavifar, A., Musco, C., Musco, C., Nouri, N., Sidford, A., and Tardos, J. (2020b). Fast and space efficient spectral sparsification in dynamic streams. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1814–1833. SIAM.

Kapralov, M., Nelson, J., Pachocki, J., Wang, Z., Woodruff, D. P., and Yahyazadeh, M. (2017b). Optimal lower bounds for universal relation, and for samplers and finding duplicates in streams. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 475–486.

Kapralov, M., Nouri, N., Sidford, A., and Tardos, J. (2019b). Dynamic streaming spectral sparsification in nearly linear time and space. *CoRR*, abs/1903.12150.

Kapralov, M. and Woodruff, D. P. (2014). Spanners and sparsifiers in dynamic streams. *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 272–281.

Karger, D. R. (1994). Random sampling in cut, flow, and network design problems. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 648–657.

Karger, D. R., Nisan, N., and Parnas, M. (1992). Fast connected components algorithms for the erew pram. In *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures*, pages 373–381.

Karloff, H., Suri, S., and Vassilvitskii, S. (2010). A model of computation for mapreduce. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 938–948. SIAM.

Kelner, J. A. and Levin, A. (2013). Spectral sparsification in the semi-streaming setting. *Theory of Computing Systems*, 53(2):243–262.

Kelner, J. A. and Madry, A. (2009). Faster generation of random spanning trees. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 13–21.

Kelner, J. A., Orecchia, L., Sidford, A., and Allen Zhu, Z. (2013). A simple, combinatorial algorithm for solving SDD systems in nearly-linear time. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 911–920.

Klimt, B. and Yang, Y. (2004). Introducing the enron corpus. In *CEAS*.

Kogan, D. and Krauthgamer, R. (2015). Sketching cuts in graphs and hypergraphs. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 367–376.

Kol, G., Raz, R., and Tal, A. (2017). Time-space hardness of learning sparse parities. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 1067–1080.

Komura, Y. (1967). Nonlinear semi-groups in Hilbert space. *Journal of the Mathematical Society of Japan*, 19(4):493–507.

Konrad, C. (2015). Maximum matching in turnstile streams. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, pages 840–852. Springer.

Konrad, C. (2018). A simple augmentation method for matchings with applications to streaming algorithms. In *43rd International Symposium on Mathematical Foundations of Computer Science (MFCS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Konrad, C., Magniez, F., and Mathieu, C. (2012). Maximum matching in semi-streaming with few passes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 231–242. Springer.

Koutis, I., Levin, A., and Peng, R. (2016). Faster spectral sparsification and numerical algorithms for SDD matrices. *ACM Trans. Algorithms*, 12(2):17:1–17:16.

Koutis, I., Miller, G. L., and Peng, R. (2010). Approaching optimality for solving SDD linear systems. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 235–244.

Koutis, I., Miller, G. L., and Peng, R. (2011). A nearly-m log n time solver for SDD linear systems. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 590–598.

Kuhn, F., Moscibroda, T., and Wattenhofer, R. (2016). Local computation: Lower and upper bounds. *Journal of the ACM (JACM)*, 63(2):17.

Kyng, R., Lee, Y. T., Peng, R., Sachdeva, S., and Spielman, D. A. (2016). Sparsified cholesky and multigrid solvers for connection laplacians. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 842–850.

Kyng, R., Pachocki, J., Peng, R., and Sachdeva, S. (2017). A framework for analyzing resparsification algorithms. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2032–2043.

Kyng, R. and Sachdeva, S. (2016). Approximate gaussian elimination for laplacians - fast, sparse, and simple. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 573–582.

Łącki, J., Mitrović, S., Onak, K., and Sankowski, P. (2020). Walking randomly, massively, and efficiently. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 364–377.

Lattanzi, S., Moseley, B., Suri, S., and Vassilvitskii, S. (2011). Filtering: a method for solving graph problems in mapreduce. In *SPAA 2011: Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures, San Jose, CA, USA, June 4-6, 2011 (Co-located with FCRC 2011)*, pages 85–94. ACM.

Lee, Y. T. and Sidford, A. (2013). Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 147–156.

Lee, Y. T. and Sidford, A. (2014). Path finding methods for linear programming: Solving linear programs in õ(vrank) iterations and faster algorithms for maximum flow. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 424–433.

Lee, Y. T. and Sun, H. (2015). Constructing linear-sized spectral sparsification in almost-linear time. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 250–269. IEEE Computer Society.

Lee, Y. T. and Sun, H. (2017). An sdp-based algorithm for linear-sized spectral sparsification. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 678–687. ACM.

Leskovec, J., Kleinberg, J. M., and Faloutsos, C. (2007). Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data*, 1(1):2.

Leskovec, J. and Krevl, A. (2014). SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data.

Leskovec, J., Lang, K. J., Dasgupta, A., and Mahoney, M. W. (2008). Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *CoRR*, abs/0810.1355.

Levi, R., Rubinfeld, R., and Yodpinyanee, A. (2017). Local computation algorithms for graphs of non-constant degrees. *Algorithmica*, 77(4):971–994.

Li, H. and Schild, A. (2018). Spectral subspace sparsification. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 385–396.

Li, M., Miller, G. L., and Peng, R. (2013). Iterative row sampling. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 127–136.

Li, P. and Milenkovic, O. (2018). Submodular hypergraphs: p-Laplacians, Cheeger inequalities and spectral clustering. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80, pages 3020–3029.

Li, Y., Nguyen, H. L., and Woodruff, D. P. (2014). Turnstile streaming algorithms might as well be linear sketches. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 174–183.

Louis, A. (2015). Hypergraph Markov operators, eigenvalues and approximation algorithms. In *Proceedings of the 47th Annual ACM on Symposium on Theory of Computing (STOC)*, pages 713–722.

Lovász, L. (1993). Random walks on graphs: A survey. In *Combinatorics, Paul ErdHos is eighty. Vol. 2*, pages 353–397. János Bolyai Mathematical Society.

Lubotzky, A., Phillips, R., and Sarnak, P. (1988). Ramanujan graphs. *Combinatorica*, 8(3):261–277.

Luby, M. (1986). A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15(4):1036–1053.

Madry, A., Straszak, D., and Tarnawski, J. (2015). Fast generation of random spanning trees and the effective resistance metric. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 2019–2036.

Mansour, Y., Rubinstein, A., Vardi, S., and Xie, N. (2012). Converting online algorithms to local computation algorithms. In *International Colloquium on Automata, Languages, and Programming*, pages 653–664. Springer.

Mansour, Y. and Vardi, S. (2013). A local computation approximation scheme to maximum matching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 260–273. Springer.

McGregor, A. (2005). Finding graph matchings in data streams. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 170–181. Springer.

McGregor, A. (2014). Graph stream algorithms: A survey. *SIGMOD Rec.*, 43(1):9–20. Preliminary version in the 31st International Colloquium on Automata, Languages and Programming (ICALP), 2004.

McGregor, A. (2017). Graph sketching and streaming: New approaches for analyzing massive graphs. In *Computer Science - Theory and Applications - 12th International Computer Science Symposium in Russia, CSR 2017, Kazan, Russia, June 8-12, 2017, Proceedings*, pages 20–24.

McGregor, A., Tench, D., Vorotnikova, S., and Vu, H. T. (2015). Densest subgraph in dynamic graph streams. In *Mathematical Foundations of Computer Science 2015 - 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part II*, volume 9235 of *Lecture Notes in Computer Science*, pages 472–482. Springer.

McGregor, A. and Vorotnikova, S. (2016). Planar matching in streams revisited. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2016, September 7-9, 2016, Paris, France*, pages 17:1–17:12.

McGregor, A. and Vorotnikova, S. (2018). A simple, space-efficient, streaming algorithm for matchings in low arboricity graphs. In *1st Symposium on Simplicity in Algorithms, SOSA 2018, January 7-10, 2018, New Orleans, LA, USA*, pages 14:1–14:4.

Meila, M. and Shi, J. (2000). Learning segmentation by random walks. *Advances in neural information processing systems*, 13:873–879.

Meyer, C. (1973). Generalized inversion of modified matrices. *SIAM Journal on Applied Mathematics*, 24(3):315–323.

Mirrokni, V. S. and Zadimoghaddam, M. (2015). Randomized composable core-sets for distributed submodular maximization. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 153–162. ACM.

Mirzasoleiman, B., Karbasi, A., Sarkar, R., and Krause, A. (2013). Distributed submodular maximization: Identifying representative elements in massive data. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 2049–2057.

Miyadera, I. (1992). *Nonlinear Semigroups*, volume 109. American Mathematical Soc.

Monemizadeh, M., Muthukrishnan, S., Peng, P., and Sohler, C. (2017). Testable bounded degree graph properties are random order streamable. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 131:1–131:14.

Munro, J. I. and Paterson, M. (1980). Selection and sorting with limited storage. *Theor. Comput. Sci.*, 12:315–323.

Nelson, J. and Yu, H. (2019). Optimal lower bounds for distributed and streaming spanning forest computation. *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1844–1860.

Newman, I. and Rabinovich, Y. (2013). On multiplicative λ-approximations and some geometric applications. *SIAM Journal on Computing*, 42(3):855–883.

Nguyen, H. N. and Onak, K. (2008). Constant-time approximation algorithms via local improvements. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 327–336. IEEE.

Nisan, N. (1990). Pseudorandom generators for space-bounded computation. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 204–212.

Onak, K., Ron, D., Rosen, M., and Rubinfeld, R. (2012). A near-optimal sublinear-time algorithm for approximating the minimum vertex cover size. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1123–1131. Society for Industrial and Applied Mathematics.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab. Previous number = SIDL-WP-1999-0120.

Parnas, M. and Ron, D. (2007). Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theoretical Computer Science*, 381(1-3):183–196.

Paz, A. and Schwartzman, G. (2017). A $(2+\epsilon)$-approximation for maximum weight matching in the semi-streaming model. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2153–2161.

Pemmaraju, S. V. (2001). Equitable coloring extends chernoff-hoeffding bounds. In *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques*, pages 285–296. Springer.

Peng, P. and Sohler, C. (2018). Estimating graph parameters from random order streams. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2449–2466.

Peng, R. and Spielman, D. A. (2014). An efficient parallel solver for SDD linear systems. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 333–342.

Raz, R. (2016). Fast learning requires good memory: A time-space lower bound for parity learning. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 266–275.

Raz, R. (2017). A time-space lower bound for a large class of learning problems. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 732–742.

Rubinfeld, R., Tamir, G., Vardi, S., and Xie, N. (2011). Fast local computation algorithms. *arXiv preprint arXiv:1104.1377*.

Rudelson, M. and Vershynin, R. (2007). Sampling from large matrices: An approach through geometric functional analysis. *Journal of the ACM*, 54(4):21.

Ruzsa, I. Z. and Szemerédi, E. (1978). Triple systems with no six points carrying three triangles. *Combinatorics (Keszthely, 1976), Coll. Math. Soc. J. Bolyai*, 18:939–945.

Schild, A. (2018). An almost-linear time algorithm for uniform random spanning tree generation. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 214–227.

Shun, J., Roosta-Khorasani, F., Fountoulakis, K., and Mahoney, M. W. (2016). Parallel local graph clustering. *Proc. VLDB Endow.*, 9(12):1041–1052.

Soma, T. and Yoshida, Y. (2019). Spectral sparsification of hypergraphs. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2570–2581.

Spielman, D. A. and Srivastava, N. (2011). Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926.

Spielman, D. A. and Teng, S. (2013). A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM J. Comput.*, 42(1):1–26.

Spielman, D. A. and Teng, S.-H. (2004). Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 81–90.

Spielman, D. A. and Teng, S.-H. (2011). Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025.

Takai, Y., Miyauchi, A., Ikeda, M., and Yoshida, Y. (2020). Hypergraph clustering based on PageRank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*, pages 1970–1978.

Teng, S.-H. (2016). Scalable algorithms for data and network analysis. *Foundations and Trends® in Theoretical Computer Science*, 12(1-2):1–274.

Tropp, J. A. (2011). User-friendly tail bounds for sums of random matrices. *Foundations of Computational Mathematics*, 12(4):389–434.

Vishnoi, N. K. (2013). Lx = b. *Foundations and Trends® in Theoretical Computer Science*, 8(1–2):1–141.

Wasserman, S. and Faust, K. (1994). *Social Network Analysis: Methods and Applications*. Cambridge University Press.

Whang, J. J., Gleich, D. F., and Dhillon, I. S. (2013). Overlapping community detection using seed set expansion. In *Proceedings of the 22nd ACM international conference on information & knowledge management*, pages 2099–2108.

White, T. (2012). *Hadoop - The Definitive Guide: Storage and Analysis at Internet Scale (3. ed., revised and updated)*. O'Reilly.

Wong, E., Baur, B., Quader, S., and Huang, C. (2012). Biological network motif detection: principles and practice. *Briefings Bioinform.*, 13(2):202–215.

Yadati, N., Nimishakavi, M., Yadav, P., Nitin, V., Louis, A., and Talukdar, P. P. (2019). HyperGCN: A new method for training graph convolutional networks on hypergraphs. In *Advances in Neural Information Processing Systems 32*, pages 1509–1520.

Yadati, N., Nitin, V., Nimishakavi, M., Yadav, P., Louis, A., and Talukdar, P. (2020). NHP: Neural hypergraph link prediction. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management (CIKM)*, pages 1705–1714.

Yang, J. and Leskovec, J. (2012). Defining and evaluating network communities based on ground-truth. *CoRR*, abs/1205.6233.

Yoshida, Y. (2016). Nonlinear Laplacian for digraphs and its applications to network analysis. In *Proceedings of the 9th ACM International Conference on Web Search and Data Mining (WSDM)*, pages 483–492.

Yoshida, Y. (2019). Cheeger inequalities for submodular transformations. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2582–2601.

Yoshida, Y., Yamamoto, M., and Ito, H. (2009). An improved constant-time approximation algorithm for maximum. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 225–234. ACM.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010). Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95.

Zelke, M. (2012). Weighted matching in the semi-streaming model. *Algorithmica*, 62(1-2):1–20.

Zhang, C., Hu, S., Tang, Z. G., and Chan, T.-H. H. (2020). Re-revisiting learning on hypergraphs: Confidence interval, subgradient method, and extension to multiclass. *IEEE Transactions on Knowledge and Data Engineering*, 32(3):506–518.

Zhu, Z. A., Liao, Z., and Orecchia, L. (2015). Spectral sparsification and regret minimization beyond matrix multiplicative updates. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 237–245. ACM.

Jakab Tardos

EPFL
Lausanne, Switzerland
+41 78 734 38 08
jakab.tardos@epfl.ch

RESEARCH INTERESTS

- Sublinear algorithms for massive graph analysis.
- Spectral graph theory.
- Approximation algorithms in streaming and MPC models.
- Submodular maximization.

EDUCATION

PhD in Computer Science September 2017 - Present
École Polytechinque Fédéral de Lausanne (EPFL)
- Supervised by Michael Kapralov.
- Teaching experience
  - Theory of Computation, EPFL - Spring semester of 2018 and 2019
  - Algorithms, EPFL - Fall semesters of 2018 and 2019
  - Advanced Algorithms, EPFL - Spring semesters of 2020 and 2021
  - Topics in Theory of Computation, EPFL - Fall semester of 2020

MASt in Pure Mathematics October 2016 - June 2017
University of Cambridge, Trinity College
- Graduated with distinction. Top 10% of the year.
- Master's essay in the field of quantum computation, titled ”The HHL algorithm”, on solving systems of linear equations in logarithmic time using quantum computers.

BSc in Mathematics September 2013 - June 2016
Eötvös Loránd University, Budapest, Hungary
- Received a final grade of 5.0, overall 4.95 GPA (both out of a maximum of 5.0) – highest GPA of the year.
- Undergraduate thesis in the field of operations research, titled ”Combinatorial Batch Codes”.

WORK EXPERIENCE

Research Internship June 2021 - August 2021
Google Brain Amsterdam
- 13 week internship contributing to the design and development of a static analysis tool for JAX and Flax (high performance machine learning libraries).
- Studied the behavior of the XLA compiler (used to improve the performance of TensorFlow and JAX code).
- Designed and implemented a system predicting certain behaviors of the XLA compiler in Python.

Research Software Engineering Internship May 2020 - July 2020
Google Zürich

- 13 week internship with a group focused on algorithm design and graph processing.
- Collaborated on research projects
  - on submodular maximization,
  - and community detection in graphs.
- Designed and implemented a framework for dynamic community detection algorithms in C++. This included a data input pipeline, a generic algorithm from which implementations would be derived, and a method for evaluating the efficacy of algorithms.

PUBLICATIONS[1]

- Michael Kapralov, Mikhail Makarov, Sandeep Silwal, Christian Sohler, Jakab Tardos. *Motif Cut Sparsifiers.* In submission.
- Michael Kapralov, Amulya Musipatla, Jakab Tardos, David P. Woodruff, Samson Zhou. *Noisy Boolean Hidden Matching with Applications.* Appeared in **ITCS 2022**.
- Michael Kapralov, Silvio Lattanzi, Navid Nouri, Jakab Tardos. *Efficient and Local Parallel Random Walks.* Appeared in **NeurIPS 2021**.
- Yuchen Wu, Jakab Tardos, MohammedHossein Bateni, Andre Linhares, Filipe Miguel Goncalves de Almeida, Andreas Montanari, Ashkan Norouzi-Fard.[2] *Streaming Belief Propagation for Community Detection.* Appeared in **NeurIPS 2021**.
- Michael Kapralov, Robert Krauthgamer, Jakab Tardos, Yuichi Yoshida. *Spectral Hypergraph Sprasifiers of Nearly Linear Size.* Appeared in **FOCS 2021**.
- Michael Kapralov, Robert Krauthgamer, Jakab Tardos, Yuichi Yoshida. *Towards Tight Bounds for Spectral Sparsification of Hypergraphs.* Appeared in **STOC 2021**.
- Michael Kapralov, Gilbert Maystre, Jakab Tardos. *Communication Efficient Coresets for Maximum Matching.* Appeared in **SOSA 2021**.
- Marwa El Halabi, Slobodan Mitrović, Ashkan Norouzi-Fard, Jakab Tardos, Jakub Tarnawski. *Fairness in Streaming Submodular Maximization: Algorithms and Hardness.* Appeared in **NeurIPS 2020**
- Michael Kaprlaov, Slobodan Mitrović, Ashkan Norouzi-Fard, Jakab Tardos. *Space Efficient Approximation to Maximum Matching Size from Uniform Edge Samples.* Appeared in **SODA 2020**.
- Michael Kapralov, Aida Mousavifar, Cameron Musco, Christopher Musco, Navid Nouri, Aaron Sidford, Jakab Tardos. *Fast and Space Efficient Spectral Sparsification in Dynamic Streams.* Appeared in **SODA 2020**.

[1]Authors appear in alphabetical order, unless otherwise stated.
[2]Author appear in alphabetical order, except for the first two author.

AWARDS AND SCHOLARSHIPS

Scholarships

- Santander Cambridge Scholarship - 2016
- Trinity College Eastern European Bursary - 2016
- Scholarship for Excellent Academic Performance - 2014-2016
- Hungarian IMO Scholarship - 2014

Major competition results

- 2nd prize at IMC (International Mathematics Competition) - 2016
- 3rd place at Vǒjtech Jarník International Mathematics Competition - 2016
- Bronze medal at IMO (International Mathematical Olympiad) - 2013
- 1st place at OKTV (Hungarian National Olympiad) in mathematics - 2013
- 4th place at OKTV (Hungarian National Olympiad) in mathematics - 2012
- Silver Medal at MEMO (Middle European Mathematics Olympiad) - 2012

ADDITIONAL SKILLS

Programming languages

- **C++** – Several university courses, several hobby projects and a research project implemented in C++, as well as a 13 week internship developing in C++.
- **Python** – University course, and a 13 week internship developing in Python.
- **Java** (+ Hadoop) – University course, and a research project on the efficient massively parallel simulation of random walks implemented in Java, using Hadoop.

Mathematics

- I have a deep understanding of many fields of mathematics; the areas I am most familiar with include algorithms theory, combinatorics, operations research, and probability theory.
- Throughout high school and university I have participated in many prestigious mathematics competitions. As a result, I am experienced at dealing creatively with difficult problems both as part of a groups and on my own.

Languages

- Proficiency in **English** – Overall score of 8.5/9.0 on the Academic IELTS exam.
- Intermediate level **French**.
- Native **Hungarian**.