ABSTRACT

Title of Dissertation: MODERN LARGE-SCALE ALGORITHMS FOR CLASSICAL GRAPH PROBLEMS

Soheil Behnezhad
Doctor of Philosophy, 2021

Dissertation Directed by: Professor MohammadTaghi Hajiaghayi
Department of Computer Science
University of Maryland

Although computing power has advanced at an astonishing rate, it has been far outpaced by the growing scale of data. This has led to an abundance of algorithmic problems where the input tends to be, by orders of magnitude, larger than the memory available on a single machine. The challenges of data processing at this scale are inherently different from those of traditional algorithms. For instance, without having the whole input properly stored in the memory of a single machine, it is unrealistic to assume that any arbitrary location of the input can be accessed at the same cost; an assumption that is essential for traditional algorithms. In this thesis, we focus on modern computational models that capture these challenges more accurately, and devise new algorithms for several classical graph problems.

Specifically, we study models of computation that only allow the algorithm to use *sublinear* resources (such as time, space, or communication). Examples include (*i*) *massively parallel computation* (à la MapReduce) algorithms where the workload is distributed among several machines each with sublinear space/communication, (*ii*) *sublinear-time* algorithms that take time sublinear in the input size, (*iii*) *streaming* algorithms that take only few passes over the input having access to a sublinear space, and (*iv*) *dynamic* algorithms that maintain a property of a dynamically changing input using a sublinear time per update.

We propose new algorithms for classical graph problems such as *maximum/maximal matching*, *maximal independent set*, *minimum vertex cover*, and *graph connectivity* in these models that substantially improve upon the state-of-the-art and are in many cases optimal. Many of our algorithms build on model-independent tools and ideas that are of independent interest and lead to improved bounds in more than one of the aforementioned settings.

Modern Large-Scale Algorithms for Classical Graph Problems

by

Soheil Behnezhad

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2021

To Faranak and Shahab, my parents.

Content

This thesis is based on the author's work during his PhD period [26, 12, 47, 27, 48, 46, 36, 39, 40, 41, 45, 37, 38, 44, 43, 28, 33, 32, 24, 29, 23, 31, 30] on the theoretical foundations of large-scale graph algorithms, and mainly covers the results in [26, 12, 48, 40, 41, 45, 37].

As we will see, various computational models can be suitable for large-scale graph problems, depending on the resources available and the specifics of the task at hand. The content of this thesis is carried out in four main parts, each corresponding to one such computational model. In Part I we cover graph algorithms in the *Massively Parallel Computations (MPC)* model which is a common abstraction of MapReduce-style computation. Part II gives graph algorithms that take *sublinear time* in the input size. Part III is on algorithms for *dynamic* graphs. Finally, Part IV discusses *streaming* graph algorithms.

Part I-Chapter 4 is based on the following publications:

- *"Exponentially Faster Massively Parallel Maximal Matching"* by Behnezhad, Hajiaghayi, and Harris [45]. In the proceedings of the 60th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2019).

- *"Massively Parallel Computation of Matching and MIS in Sparse Graphs"* by Behnezhad, Brandt, Derakhshan, Fischer, Hajiaghayi, Karp, and Uitto [37]. In the proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC 2019).

Part I-Chapter 5 is based on the following publication:

- *"Near-Optimal Massively Parallel Graph Connectivity"* by Behnezhad, Dhulipala, Esfandiari, Lacki, and Mirrokni [41]. In the proceedings of the 60th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2019).

Part II-Chapter 6 is based on the following very recent work:

- *"Time-Optimal Sublinear Algorithms for Matching and Vertex Cover"* by Behnezhad [26]. In the proceedings of the 62nd IEEE Annual Symposium on Foundations of Computer Science (FOCS 2021).

Part III-Chapters 7 and 8 are based on the following publication:

- *"Fully Dynamic Maximal Independent Set with Polylogarithmic Update Time"* by Behnezhad, Derakhshan, Hajiaghayi, Stein, and Sudan [40]. In the proceedings of the 60th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2019).

Part III-Chapter 9 is based on the following publication:

- *"Fully Dynamic Matching: Beating 2-Approximation in Δ^{ε} Update Time"* by Behnezhad, Lacki, and Mirrokni [48]. In the proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA 2020).

Part IV-Chapter 10 is based on the following publication:

- *"Beating Two-Thirds For Random-Order Streaming Matching"* by Assadi and Behnezhad [12]. In the proceedings of the 48th International Colloquium on Automata, Languages, and Programming (ICALP 2021).

Acknowledgments

First and foremost I'd like to thank my advisor MohammadTaghi Hajiaghayi. I am grateful to Mohammad for helping me find my path in research, for encouraging me to expand my collaborations, for giving me the freedom to explore what I was interested in, and for supporting me well beyond just my academic life. This thesis would not have been possible without Mohammad's guidance and I will always be indebted to him.

I cannot stress how grateful I am to Avrim Blum and Vahab Mirrokni for their support and mentorship. Avrim's generosity and his caring personality, as well as his inspiring research have had great impact on me. I am grateful to him for hosting me at TTIC and for all the career advice he's given me. I am also deeply thankful to Vahab for his faith in me, the exciting internship I had in his group at Google Research, and his support over the years.

I am also immensely thankful to Sanjeev Khanna, Christos Papadimitriou, Cliff Stein, and Madhu Sudan for their support, for being wonderful sources of advice, and for all I've learned from them during our collaborations. I am also deeply thankful to Sepehr Assadi for all the great advice he has given me at different stages of my PhD and research. I wish to also thank Jakub Lacki and Nima Reyhani for hosting me at Google Research and Upwork.

Next, I would like to express my gratitude to the members of my thesis committee Alexander Barg, Avrim Blum, MohammadTaghi Hajiaghayi, Sanjeev Khanna, David Mount, Aravind Srinivasan, and Madhu Sudan for their time and their valuable comments. I am additionally thankful to the amazing staff at UMD, especially Tom Hurst, Sharron McElroy, Jodie Gray, and Regis Cornett who do a phenomenal job and have been extremely supportive during my years at UMD. I also thank Google for my Google PhD Fellowship; majority of the results presented in this thesis were obtained while I was funded by Google.

Research would not have been as fun if it had not been for the time I got to spend with my brilliant collaborators. Thanks to Sepehr Assadi, MohammadHossein Bateni, Avrim Blum, Sebastian Brandt, Moses Charikar, Sina Dehghani, Mahsa Derakhshan, Laxman Dhulipala, Hossein Esfandiari, Alireza Farhadi, Manuela Fischer, Jon Froehlich, MohammadTaghi Hajiaghayi, David Harris, Richard Karp, Sanjeev Khanna, Raimondas Kiveris, Marina Knittel, Jakub Lacki, Silvio Lattanzi, Weiyun (Anna) Ma, Mohammad Mahdian, Vahab Mirrokni, Christos Papadimitriou, Nima Reyhani, Ronald Rivest, Hamed Saleh, Warren Schudy, Masoud Seddighin, Saeed Seddighin, Aleksandrs Slivkins, Philip Stark, Cliff Stein, Madhu Sudan, Elif Tan, Li-Yang Tan, Jara Uitto, and Hadi Yami.

My memories of my PhD are filled with the times I shared with my amazing friends at UMD to whom I owe a big thank you: Ali, Alireza, Amin, Hadi, Hamed, Hossein, Kazem, Kiana, Madeline, Mahyar, Majeed, Marina, Parsa, Saba, Saeed, Sina, and Soheil (sorry if I am forgetting a name). I would also like to thank my long-time friend Mohammad Zabolian who's had a great impact on the person that I am today.

Above all, I am forever grateful to my parents Faranak and Shahab, and my little brother Sepehr for their love and support. None of this would have been possible without them and I am so sorry that I was so far away from home for the last five years.

Finally, I would like to thank the person with whom I have had the most collaborations, both in research and otherwise, my lovely wife Mahsa Derakhshan. Thank you for all the warmth and love that you've brought to my life, and your willing ear for whatever that I find interesting. I am so incredibly lucky to have you, Mahsa!

Table of Contents

Chapter 1

Introduction

Although computing power has advanced at an astonishing rate, it has been far outpaced by the growing scale of data. This has led to an abundance of algorithmic problems where the input is massive in size. Data processing at this scale is accompanied by challenges that are inherently different from those of traditional algorithms. For example, such massive inputs are often, by orders of magnitude, larger than the memory available on a single machine. Without having the whole input properly stored in the memory of a single machine, it is unrealistic to assume that any arbitrary location of the input can be accessed at the same cost; an assumption that is essential for most traditional algorithms. As a result, we have seen a paradigm shift toward computational models that capture these challenges more accurately by allowing only *sublinear* resources such as time, space, or communication. In this thesis, we focus on these modern computational models and study classical graph problems in them.

Graphs are versatile mathematical objects for modeling pairwise relations among objects of various forms. For example, graphs can model connections in social networks, links over the world wide web, the wiring of the human brain, or transportation infrastructure. Due to their wide range of applications, graphs have been one of the most pervasive objects in Computer Science. Classical graph problems such as *maximum matching*, *maximum independent set*, *minimum vertex cover*, or *connectivity* problems have been at the cornerstone of algorithmic research, and some of the most sophisticated concepts of the field have been developed in the study these problems.[1] In this thesis, we revisit these classical problems for graphs that are massive in size.

Massive graphs are ubiquitous. For example, the facebook graph has billions of vertices (active monthly users) and trillions of edges (friendships) [70], both the human brain and the web graph are estimated to have around 100's of billions of vertices [19, 129], and the human brain is estimated to have more than 100's of trillions of edges [113].

[1]Most notably, the notion of polynomial-time solvability was first formalized by Jack Edmonds [82] in the study of the maximum matching problem.

Depending on the resources available and the specifics of the task at hand, one may choose various forms of large-scale algorithms. Examples include

- Massively Parallel Computation (MPC) algorithms,
- Sublinear-time algorithms,
- Streaming algorithms, and
- Dynamic algorithms.

We first give an overview of these models, and then discuss our contributions for each.

1.1 Overview of the Computational Models

Massively Parallel Computation (MPC): As discussed, one of the main challenges with processing large-scale data is the fact that the input is by orders of magnitude larger than the space available in a single machine. One of the most successful practical workarounds is to distribute the workload among several machines running in parallel. This is often achieved via frameworks such as MapReduce [80], Hadoop [152], Spark [155], and their variants. The *Massively Parallel Computations* (MPC) model which was first introduced by Karloff, Suri, and Vassilvitskii [109] and was further refined in the works of [98, 25, 8], captures the essence of all of these frameworks and is the standard theoretical model for studying the computational power of these systems.

In the MPC model, we have M machines each with space S. The input—which in this thesis will always be a graph G—is arbitrarily distributed among the machines. Particularly, each machine receives an arbitrary subset of the edges in the input graph. It would be insightful to think of both S and M as parameters that are sublinear in the input size. Computation then proceeds in synchronous *rounds*. In each round, each machine can perform any computation on its local data, and can send messages to other machines which will be delivered at the start of the next round. A machine can send different messages to different machines in a round, with the only constraint being that the total messages sent and received by each machine in each round should fit its memory. The main parameter to optimize is the number of rounds, which often is the main bottleneck in practice [110].

Streaming Algorithms: The streaming model is another popular model of large-scale computation. In this model, pieces of the input, e.g. the edges of a graph G, arrive one by one in a stream. The algorithm does not have enough space to store the whole input, hence, has to decide "on the fly" what information to keep in the memory in order to be able to compute the desired property of the input at the end of the stream. Streaming algorithms can be used in various ways. A common scenario in large-scale computation is to have an input that is much larger than the main (random-access) memory, but fits an external memory. Now if we have a space-efficient streaming algorithm, we can read the input line by line from the external memory and process it in a streaming fashion. We refer interested readers to the survey of Muthukrishnan [131] for more details about the streaming model.

Sublinear-time Algorithms: Linear-time algorithms have long been the gold standard in algorithm design. For massive inputs, however, even such algorithms do not run efficiently. *Sublinear-time* algorithms get around this challenge by simply spending time that is sublinear in the input size. See the survey of Czumaj and Sohler [76] for some classic such algorithms.

Since sublinear-time algorithms can only read a small fraction of the input, it is important to specify how the input can be accessed. For graph problems—the focus of this thesis—two models have been commonly considered in the literature:

- The adjacency list model: In this model, for any vertex v of its choice, the algorithm may query the degree of v in the graph and, for any $1 \leq i \leq \deg(v)$, may query the i-th neighbor of v stored in an arbitrarily ordered list.
- The adjacency matrix model: In this model, the algorithm may query, for any vertex-pair (u, v) of its choice, whether or not u and v are adjacent in the graph.

We consider both models in this thesis.

Dynamic Algorithms: In many large-scale applications, the underlying data changes dynamically over time. One can of course recompute any desired property from scratch after each update using the best known static algorithm, but this is often prohibitive. Dynamic algorithms help maintain such properties while taking a small time (often sublinear in the input size) per update. Our focus in this thesis is particularly on *fully dynamic* graphs, namely graphs that undergo both edge insertions and deletions. More precisely, we have a graph G on a fixed vertex set V. Every update either inserts an edge to the graph or removes an edge already in the graph. The goal is to maintain a property of the graph after every update while spending a small time per update, which we refer to as the *update-time*.

1.2 Our Contributions

In this thesis, we study a number of fundamental graph problems in the models discussed above and propose new algorithms that substantially improve upon the state-of-the-art and are in some cases provably optimal. The main graph problems that we study include

- *maximum (cardinality) matching* (MCM),
- *maximal matching* (MM),
- *maximal independent set* (MIS),
- *minimum vertex cover* (MVC), and
- *graph connectivity* (GC).

Readers unfamiliar with these problems can find their formal definitions as well as the (mostly standard) notation that we use throughout the thesis in Chapter 2.

Our main results in this thesis are summarized in Table 1.1. Here we give a brief overview of these results and how they compare to prior works. We denote the number of vertices of the graph by n, the number of its edges by m, and its maximum degree by Δ.

Chapter 4 – Massively Parallel Maximal Matching: In Chapter 4, we consider the *maximal matching* (MM) problem in the MPC model. For many graph problems, including maximal matching, $O(\log n)$ round MPC algorithms can be achieved in a straightforward way by simulating traditional parallel (PRAM) algorithms [126, 104, 6]. But we often desire much faster algorithms in the MPC model.

In Chapter 4 we present an $O(\log\log\Delta)$-round MPC algorithm for maximal matching using $S = O(n)$ local space and total space $M \times S = O(m)$. This exponentially improves all prior algorithms for maximal matching which either required $\log^{\Omega(1)}(n)$ rounds or $n^{1+\Omega(1)}$ space per machine. This result, published first in [45], culminated a long and exciting line of work in the literature [78, 15, 94] that started with the breakthrough result of Czumaj, Lacki, Madry, Mitrovic, Onak, and Sankowski [78]. We achieve this result by analyzing a natural algorithm that partitions the vertex set of the graph and finds a greedy matching in each partition. Our analysis of this algorithm unveils a novel application of sublinear-time algorithms for proving concentration bounds that we find to be of independent interest. This result also resolves several open problems raised by Czumaj et al. [78] who first conjectured that a variant of this algorithm might work.

Model	**Problem**	**Approx**	**Time/Pass/Round**	**Chapter**
MPC with $O(n)$ space	MM	Exact	$O(\log\log\Delta)$ rounds	4
MPC with $O(n^{\varepsilon})$ space	MM	Exact	$\sqrt{\log\lambda}\cdot\text{poly}(\log\log n)$ rounds	4
MPC with $O(n^{\varepsilon})$ space	MIS	Exact	$\sqrt{\log\lambda}\cdot\text{poly}(\log\log n)$ rounds	4
MPC with $O(n^{\varepsilon})$ space	GC	Exact	$O(\log D + \log\log n)$ rounds	5
Sublinear Time	MCM	$0.5-\varepsilon$	$\widetilde{O}(n)$ time (adjacency list)	6
Sublinear Time	MCM	$(0.5, \varepsilon n)$	$\widetilde{O}(n)$ time (adjacency matrix)	6
Sublinear Time	MVC	$2+\varepsilon$	$\widetilde{O}(n)$ time (adjacency list)	6
Sublinear Time	MVC	$(2, \varepsilon n)$	$\widetilde{O}(n)$ time (adjacency matrix)	6
Fully Dynamic	MIS	Exact	$\text{poly}(\log n)$ update-time	7
Fully Dynamic	MM	Exact	$\text{poly}(\log n)$ update-time	8
Fully Dynamic	MCM	$0.5+\Omega_{\varepsilon}(1)$	$O(\Delta^{\varepsilon}+\text{poly}\log n)$ update-time	9
Semi-Streaming	MCM	$2/3+\Omega(1)$	1 pass (random order)	10

Table 1.1: Table of the main results presented in this thesis. The considered problems are maximum cardinality matching (MCM), maximal matching (MM), maximal independent set (MIS), graph connectivity (GC), and minimum vertex cover (MVC). In these bounds n denotes the number of vertices, Δ denotes the maximum degree, λ denotes the arboricity of the graph, and D denotes the diameter of the graph — we refer readers to Chapter 2 for the formal definitions of these graph properties. The parameter ε in the results can be any constant in $(0, 1)$. All the algorithms for MM also give 2-approximations for MVC with essentially the same bounds.

While the space per machine of $S = O(n)$ is suitable for dense graphs, it is not quite useful when the input graph is sparse yet large-scale. For such graphs, it is unrealistic to even assume that the vertex-set fits the memory of a single machine. We further show in Chapter 4 that one can improve the local space to $S = O(n^{\varepsilon})$ for any constant $\varepsilon > 0$ and still obtain a $\text{poly}(\log\log n)$ round algorithm for maximal matching (as well as maximal independent set) so long as the input graph is *uniformly sparse* (i.e., has arboricity $\lambda = \text{poly}(\log n)$).

Chapter 5 – Massively Parallel Graph Connectivity: Identifying the connected components of a graph is another fundamental problem that has been studied in a variety of settings (see e.g. [2, 88, 74, 146, 151, 142] and the references therein). This problem is also of great practical importance [145] with a wide range of applications, e.g. in clustering [142]. Like many other graph problems, $O(\log n)$-round MPC algorithms for graph connectivity have been known for a long time. On the negative side, a widely believed 1v2-Cycle conjecture (see Conjecture 5.1) [153, 143, 118, 18] implies that this is the best possible. The 1v2-Cycle conjecture states that distinguishing whether the input is a cycle on n vertices or two cycles on $n/2$ vertices each requires $\Omega(\log n)$ rounds with $n^{1-\Omega(1)}$ space per machine.

The 1v2-Cycle conjecture and the matching upper bound, however, are far from explaining the true complexity of the problem. First, the hard example used in the conjecture is very different from what most graphs look like. Second, the empirical performance of the existing algorithms (in terms of the number of rounds) is much lower than what the upper bound of $O(\log n)$ suggests [112, 118, 149, 142, 127]. This disconnect between theory and practice has motivated the study of graph connectivity as a function of diameter D of the graph. The reason is that the vast majority of real-world graphs, indeed have very low diameter [121, 73]. This is reflected in multiple theoretical models designed to capture real-world graphs, which yield graphs with polylogarithmic diameter [61, 99, 125, 62].

We prove in Chapter 5 that for any constant $\varepsilon > 0$, there is an MPC algorithm with local space $S = O(n^{\varepsilon})$ and total space $M \times S = O(m)$ that given a graph with diameter D, identifies its connected components in $O(\log D + \log\log n)$ rounds. Note that for the wide range of values $D = \log^{\Omega(1)} n$, the this algorithm takes $O(\log D)$ rounds which can be shown to be optimal under Conjecture 5.1. Moreover, when D is not in this range the algorithm takes only $O(\log\log n)$ rounds. This algorithm improves two previous algorithms by Andoni, Song, Stein, Wang, and Zhong [9] and Assadi, Sun, and Weinstein [18].

Chapter 6 – Sublinear Time Algorithms for Matching and Vertex Cover: In Chapter 6 we focus particularly on algorithms that estimate the size of maximum matching or minimum vertex cover. We give a near-tight analysis of the "average query-complexity" of the famous random greedy maximal matching algorithm, improving upon a seminal work of Yoshida, Yamamoto, and Ito [154]. This leads to a host of sublinear-time algorithms for approximating the size of maximum matching and minimum vertex cover. Particularly, for any $\varepsilon > 0$, we get that there is a randomized algorithm that reports a

(*i*) $(\frac{1}{2} - \varepsilon)$-approximation to the size of MCM, and a $(2 + \varepsilon)$-approximation to the size of MVC using $O(n) + \widetilde{O}(\Delta/\varepsilon^2) = \widetilde{O}(n/\varepsilon^2)$ time in the adjacency list model.

(*ii*) $(\frac{1}{2}, \varepsilon n)$-approximation to the size of MCM and a $(2, \varepsilon n)$-approximation to the size of MVC using $\widetilde{O}(n/\varepsilon^3)$ time in the adjacency matrix model.

Estimating the size of matching or vertex cover has been studied extensively in the literature of sublinear time algorithms [139, 133, 154, 136, 107, 69]. Our results above resolve major open problems of the area and turn out to be time-optimal up to logarithmic factors. Part (*i*) of this result, notably, gives the *first* multiplicative estimator in the literature that runs in $\widetilde{O}(n)$ time for all graphs. For a $(2 + \varepsilon)$-approximation, in particular, no $o(n^2)$ time algorithm was known for general graphs prior to our work. Part (ii), on the other hand, improves over the previous best running time of $\widetilde{O}_\varepsilon(n\sqrt{n})$ [69].

Chapter 7 – Dynamic Maximal Independent Set: The maximal independent set (MIS) is another fundamental graph property with several theoretical and practical applications. In Chapter 7 we study the maximal independent set (MIS) problem in fully-dynamic graphs. We first overview the relevant work in the literature [64, 14, 101, 81, 137, 17] and then describe our contribution.

In static graphs with m edges, a simple greedy algorithm can find an MIS in $O(m)$ time. As such, one can trivially maintain MIS by recomputing it from scratch after each update, in $O(m)$ time. In a pioneering work, Censor-Hillel, Haramaty, and Karnin [64] presented an $O(\Delta)$ update-time algorithm for this problem. Assadi, Onak, Schieber, and Solomon [14] later gave an algorithm with $O(m^{3/4})$ update-time; thereby improving the $O(m)$ bound for all graphs. This result was further improved in a series of subsequent papers [101, 81, 137, 17]. The fastest algorithm algorithm prior to our work was a randomized algorithm of [17], which required $\widetilde{O}(\min\{\sqrt{n}, m^{1/3}\})$ amortized update-time in n-vertex graphs.

In Chapter 7, we show that it is possible to maintain an MIS of fully-dynamic graphs in polylogarithmic time. This exponentially improves over the prior algorithms, which all had polynomial update-time on general graphs.

In Chapter 8, we show how a variant of this algorithm can also maintain a randomized greedy maximal matching in polylogarithmic update-time. While algorithms with polylogarithmic update-time for maximal matching have been known previously [21, 147], none of them maintained a randomized greedy one, which turns out to be an important feature. Particularly, as we soon describe, we use this result as a black-box in Chapter 9 to obtain an improved approximation for dynamic MCM.

Chapter 9 – Fully Dynamic Approximate Matching: The problem of maintaining a large matching in the dynamic setting has also received significant attention over the last two decades (see [135, 22, 132, 102, 58, 55, 147, 57, 56, 65, 10, 53] and the references therein). One of the major achievements in the literature of dynamic graph algorithms, has been discovery of algorithms that maintain a maximal matching, and thus a $\frac{1}{2}$-approximation of maximum matching, in polylogarithmic (or constant) update time [22, 147]. In a sharp contrast, however, we have little understanding of the update-time-complexity once we go slightly above $\frac{1}{2}$-approximation. A famous open question of the area, asked first[2] in the pioneering paper of Onak and Rubinfeld [135] from 2010 is whether there exists a $(\frac{1}{2}+\Omega(1))$-approximate algorithm with polylogarithmic update-time. More than a decade later, we are still far from achieving a polylogarithmic update-time algorithm. Prior to our work, the fastest algorithm for maintaining a matching with a better-than-$\frac{1}{2}$ approximation factor was presented by Bernstein and Stein [52] which handles updates in $O(m^{1/4})$ time where m denotes the number of edges in the graph.

In Chapter 9 we prove that for any $\varepsilon \in (0,1)$ and any n-vertex graph of maximum degree Δ, one can maintain a $\frac{1}{2}+\Omega_\varepsilon(1)$ approximate maximum matching in $O(\Delta^\varepsilon)+\text{polylog}\, n$ worst-case update time. As such, the update-time can be improved to any arbitrarily small polynomial while obtaining a a strictly better-than-$\frac{1}{2}$ approximation. In this result, we use the abovementioned algorithm of Chapter 8 on maintaining a randomized greedy maximal matching as a black-box, and in a crucial way. See Chapter 9 for more details and, especially, for comparisons with the prior work of [55].

Chapter 10 – Random-Order Streaming Matching: In Chapter 10 we turn our attention to the streaming model and study the MCM problem in this model. Our focus, is particularly on the *semi-streaming* model of computation [88] where the space available is $n \cdot \text{poly}(\log n)$. The greedy algorithm for maximal matching gives a simple $1/2$-approximation algorithm to this problem in $O(n)$ space. When the stream of edges is adversarially ordered, this is simply the best result known for this problem, while it is also known that a better than $\frac{1}{1+\ln 2} \sim 0.59$-approximation is not possible [106] (see also [105, 96]). Closing the gap between these upper and lower bounds is a longstanding open problem.

Going beyond this "doubly worst case" scenario, namely, an adversarially-chosen graph and an adversarially-ordered stream, there has been an extensive interest in recent years in studying this problem on *random order streams* [116, 114, 90, 85, 15, 50]. Prior to our work, the best known approximation was the (almost) $2/3$-approximation of Bernstein [50]. This is a natural barrier for the problem [114, 50]. In particular, [50] posed obtaining a $(2/3 + \Omega(1))$-approximation to this problem as an important open question. We resolve this question in Chapter 10 and present an algorithm that for an absolute constant $\varepsilon_0 > 0$ obtains a $(\frac{2}{3} + \varepsilon_0)$-approximate maximum matching using $O(n \log n)$ space.

The importance of this result is in that it proves that $\frac{2}{3}$-approximation is not the "right" answer to this problem. This is in contrast to some other problems of similar flavor such as

[2]See also [54, Section 4], [52, Section 7] or [65, Section 1].

the one-way communication complexity of matching (on adversarial partitions) [96, 13] or the fault-tolerant matching problem [13] which are both solved using similar techniques and for both $\frac{2}{3}$-approximation is provably the best possible.

A Unified Approach for Designing Large-Scale Graph Algorithms: Many of our algorithms build on model-independent tools and ideas that are of independent interest and lead to improved bounds in more than one of the aforementioned settings. Particularly, one of the conceptual contributions of this thesis, is to show that "randomized greedy algorithms" for maximal independent set and maximal matching are extremely powerful tools for large-scale graph processing. We elaborate more on randomized greedy algorithms and their properties that we use in Chapter 3.

Chapter 2

Preliminaries

In this chapter, we overview some of the common notation and definitions that we use throughout this thesis.

2.1 Graph Notation and Basic Tools

All of the algorithms considered in this thesis are graph algorithms. A graph $G = (V, E)$ has a vertex-set V and an edge-set E each element of which is a pair of vertices. All the graphs considered in this thesis are simple, undirected, and unweighted. That is, all the edges are unordered pairs between distinct vertices (i.e., no self-loops), and that a vertex-pair belongs to E at most once (i.e., no parallel edges).

For a vertex $v \in V$, we use the neighbor-set $N_G(v)$ of v to denote the set of vertices u that are connected to v (i.e., there is an edge $\{u, v\} \in E$). Moreover, we use $\Gamma_G(v) := N_G(v) \cup \{v\}$ and use $\deg_G(v) := |N_G(v)|$ to denote the *degree* of v in G.

A path P of G is an ordered set of distinct vertices $(v_1, \ldots, v_k)$ such that $\{v_i, v_{i+1}\} \in E$ for all $i \in \{1, \ldots, k-1\}$. The length of a path $(v_1, \ldots, v_k)$ is the number of edges in it which is $k-1$. For any two vertices u and v, we use $\text{dist}_G(u, v)$ to denote the length of the shortest path connecting u and v; if no such path exists, then $\text{dist}_G(u, v) = \infty$.

For any $V' \subseteq V$, we use $G[V']$ to denote the induced subgraph of G on V'; that is, $G[V']$ contains edge e in E if and only if both of its endpoints are in V'. For two disjoint subsets $A, B \subseteq V$, we use $G[A \times B]$ to denote a bipartite subgraph of G with one part being A, the other part being B, and the edge-set including any edge of E with one endpoint in A and the other endpoint in B. The *line graph* of a graph $G = (V, E)$, is a graph $L(G) = (V_L, E_L)$ whose vertex-set V_L equals the edge-set E of G, and there is an edge $\{e, e'\} \in E_L$ iff the edges e and e' of G share one endpoint.

In all the notation define above, we may drop the subscript G if the graph G is clear from the context.

Throughout the thesis, we use $G = (V, E)$ to denote the input graph. Unless otherwise stated, we use the following notation to refer to the main properties of G:

- n: the number of vertices in G, i.e., $n := |V|$.
- m: the number of edges in G, i.e., $m := |E|$.
- Δ: the maximum degree in G, i.e., $\Delta := \max_{v \in V} \deg(v)$.
- $\bar{d}$: the average degree in G, i.e., $\bar{d} := \left(\sum_{v \in V} \deg(v)\right)/n = 2m/n$.
- $\mu(G)$: the size of the maximum matching of G.
- $\nu(G)$: the size of the minimum vertex cover of G.
- D: the diameter of G defined as $\max_{u,v \in V : \text{dist}(u,v) \neq \infty} \text{dist}(u, v)$.
- λ: this is the *arboricity* of the graph defined as $\max_{U \subseteq V} \lceil m_U/(|U| - 1) \rceil$ where m_U denotes the number of edges in the graph with both endpoints in U.

Graph problems. We mainly study the following graph problems in this thesis:

- **Maximum/maximal matching:** An edge subset $M \subseteq E$ is a *matching* if no two edges in M share an endpoint. A matching M of a graph G is a *maximal matching* if it is not a subset of another matching. A matching M is a *maximum matching* if every matching in G has size at most $|M|$. Note that a maximum matching is a maximal matching but the converse is not necessarily true.
- **Maximal independent set:** A vertex subset $I \subseteq V$ is a *maximal independent set* (MIS) of G if no two vertices in I are connected and any vertex in $V \setminus I$ has a neighbor in I.
- **Vertex cover:** A vertex subset $C \subseteq V$ is a *vertex cover* of G if any edge in the graph has at least one endpoint in U. We are often interested in the minimum size vertex cover.
- **Graph connectivity:** A connected component of G is a maximal subset $U \subseteq V$ such that there is a path between any pairs of vertices in U. The graph connectivity problem asks to identify all connected components of G.

Alternating and augmenting paths. Given a matching M, an *alternating path* P for M is a path whose edges alternatively belong to M and do not belong to M. An *augmenting path* for M is an alternative path that starts and ends with edges that do not belong to M. Given an augmenting path P for M, we use

$$M \oplus P := (M \setminus P) \cup (P \setminus M)$$

to denote the matching obtained by flipping the containment of edges of P in M. Given two matchings M and M', their *symmetric difference* $M \Delta M'$ is a graph including only the edges that belong to exactly one of M and M'.

2.2 Non-graph Notation

For any positive integer k, we use $[k]$ to denote the set $\{1, \ldots, k\}$. We use the term with high probability, abbreviated "w.h.p." to denote probability at least $1 - 1/n^{\Omega(1)}$ where n is the number of vertices in the input graph G. We also use the term with exponentially high probability, abbreviated "w.e.h.p." to denote probability at least $1 - e^{-n^{\Omega(1)}}$.

2.3 Probabilistic Tools

Throughout the thesis, we will rely on a number of concentration bounds. The first one is the standard Chernoff bound for sum of independent Bernoulli random variables:

Proposition 2.1 (Chernoff Bound; cf. [4]). *Let $X_1, \ldots, X_n$ be independent Bernoulli random variables and define $X := \sum_{i=1}^{n} X_i$. For any $\lambda > 0$,*

$$\mathbf{Pr}[|X - \mathbf{E}[X]| \geq \lambda] \leq 2 \exp\left(-\frac{\lambda^2}{3\,\mathbf{E}[X]}\right).$$

In some cases, we will need concentration bounds on more general functions of independent random variables. For this purpose, we use two main bounds both of which concern functions $f(x_1, \ldots, x_n)$ which have *Lipschitz properties*, namely, that changing each coordinate x_i has a relatively small change to the value of f. The first one is the Efron-Stein inequality which bounds the variance:

Proposition 2.2 (Efron-Stein inequality [148]). *Fix an arbitrary function $f : \{0,1\}^n \to \mathbb{R}$ and let $X_1, \ldots, X_n$ and $X'_1, \ldots, X'_n$ be $2n$ i.i.d. Bernoulli random variables. For $\vec{X} := (X_1, \ldots, X_n)$ and $\vec{X}^{(i)} := (X_1, \ldots, X_{i-1}, X'_i, X_{i+1}, \ldots, X_n)$, we have*

$$\mathbf{Var}(f(\vec{X})) \leq \frac{1}{2} \cdot \mathbf{E}\Big[\sum_{i=1}^{n} \big(f(\vec{X}) - f(\vec{X}^{(i)})\big)^2\Big].$$

We also use the following form the of the *bounded differences inequality*. We say a function f is λ-Lipschitz if changing each coordinate of its input changes the value of f by at most λ.

Proposition 2.3 (Bounded differences inequality)**.** *Let f be a λ-Lipschitz function on k variables, and let $\vec{X} = (X_1, \ldots, X_k)$ be a vector of k independent (not necessarily identically distributed) random variables. Then,*

$$\mathbf{Pr}\left[f(\vec{X}) \geq \mathbf{E}[f(\vec{X})] + t\right] \leq \exp\left(\frac{-2t^2}{k\lambda^2}\right).$$

We will use the following direct corollary of Proposition 2.3.

Corollary 2.4. *Let f be a λ-Lipschitz function on k variables, and let $\vec{X} = (X_1, \ldots, X_k)$ be a vector of k independent (not necessarily identically distributed) random variables. Then with probability $1 - e^{-n^{\Omega(1)}}$:*

$$f(\vec{X}) \leq \mathbf{E}[f(\vec{X})] + \lambda n^{0.01}\sqrt{k}.$$

We also need Lovász Local Lemma (LLL) in our proofs.

Proposition 2.5 (Lovász Local Lemma; cf. [4])**.** *Let $p \in (0,1)$ and $d \geq 1$. Suppose $\mathcal{E}_1, \ldots, \mathcal{E}_t$ are t events such that $\mathbf{Pr}(\mathcal{E}_i) \leq p$ for all $i \in [t]$ and each $\mathcal{E}_i$ is mutually independent of all but (at most) d other events $\mathcal{E}_j$. If $p \cdot (d+1) < 1/e$ then $\mathbf{Pr}[\cap_{i=1}^{n} \overline{\mathcal{E}_i}] > 0$.*

Chapter 3

Randomized Greedy as a Common Tool

As discussed in the introduction, one of the conceptual contributions of this thesis, is to show that randomized greedy algorithms for maximal independent set and maximal matching are extremely useful for large-scale graph processing. Particularly we use these algorithms in the following chapters of this thesis, across various models of large-scale computation:

- In Chapter 4 we use properties of randomized greedy maximal matching (RGMM) to design an exponentially faster MPC algorithm for maximal matching.
- In Chapters 7 and 8 we respectively show how to maintain randomized greedy maximal independent set (RGMIS) and RGMM efficiently in the fully dynamic setting. In Chapter 9 we further build on this approach and show how to maintain a better-than-two approximate matching in the fully dynamic setting.
- In Chapter 6 we use RGMM to design time-optimal sublinear algorithms for estimating the size of maximum matching and minimum vertex cover.

In this chapter, we present some of the useful properties of randomized greedy algorithms that we will use in the forthcoming parts of the thesis.

3.1 Definitions

Let us start with the formal definition of a greedy maximal independent set which processes the vertices in some given order and greedily adds them to the independent set.

$\mathsf{GMIS}(G, \pi_V)$: Greedy maximal independent set with respect to ordering π of vertices.

Input: Graph $G = (V, E)$ and a permutation π_V over the vertex-set V.

Initialize $I \leftarrow \emptyset$. Iterate over the vertices in the order of π_V. Upon visiting a vertex v, if no neighbor of v belongs to I, add v to I.
We denote the final value of I, which is a maximal independent set of G, by $\mathsf{GMIS}(G, \pi_V)$.

A greedy maximal matching is defined similarly, except that we iterate over the edges instead of the vertices, and greedily add them to a matching.

$\mathsf{GMM}(G, \pi_E)$: Greedy maximal matching with respect to ordering π_E of edges.

Input: Graph $G = (V, E)$ and a permutation π_E over the edge-set E.

Initialize $M \leftarrow \emptyset$. Iterate over the edges in the order of π_E. Upon visiting an edge e, if no edge incident to e belongs to M, add e to M.
We denote the final value of M, which is a maximal matching of G, by $\mathsf{GMM}(G, \pi_E)$.

Observation 3.1. *The greedy maximal matching algorithm is equivalent to running the greedy MIS algorithm on the line graph. That is, for any graph G and any permutation π over the edge-set of G,*

$$\mathsf{GMM}(G, \pi) = \mathsf{GMIS}(L(G), \pi).$$

The two algorithms above are particularly useful if π_V and π_E are chosen uniformly at random from all possible permutations of V and E respectively. We call the resulting objects randomized greedy maximal independent set (RGMIS) and randomized greedy maximal matching (RGMM) respectively.

Random ranks instead of random permutations: One useful way to draw permutations π_V and π_E uniformly at random from all possibilities, is to draw independent ranks. That is, suppose that for each vertex $v \in V$, we draw a real ρ_v independently and uniformly from $[0, 1]$. We can then obtain π_V by sorting the vertices in the increasing order of their ranks. It is not hard to see that the resulting permutation π_V is chosen uniformly from all possible permutations. We can obtain π_E similarly by drawing random ranks on the edges and sorting them with respect to these ranks.

Drawing independent random ranks instead of directly picking a random permutation has many benefits. For instance, by conditioning on the rank of an element (say a vertex or an edge), the ranks of all other elements remain independent. On the contrary, however, once we condition on the location of an element in a permutation, other elements cannot be in this location and so they are not entirely independent. This approach is also useful for dynamic graphs, where the edges of the graph change over time, or for sublinear-time algorithms where we do not have access to all the edges.

3.2 Local Simulation Oracles and Query-Complexity

Let us first focus on the RGMM algorithm. Suppose that we would like to determine if a given vertex (or edge) belongs to the produced matching for some given ordering π. Of course we can run $\mathsf{GMM}(G, \pi)$ in $\Theta(m+n)$ time and answer this. But can we do better? Due to the greedy structure of the algorithm, we can indeed answer such queries much more efficiently. Intuitively, an edge e belongs to $\mathsf{GMM}(G, \pi)$ if and only if none of the other incident edges to e with a lower rank than e appear in $\mathsf{GMM}(G, \pi)$. This naturally suggests a *local simulation oracle* to determine if a given edge belongs to $\mathsf{GMM}(G, \pi)$ without constructing $\mathsf{GMM}(G, \pi)$ on the whole graph.

The following local procedure, which formalizes the intuition above, was first proposed by Nguyen and Onak [133]. We note that the algorithm here is slightly different from the algorithm of [133] in that it "caches" recursive calls.

Algorithm 1: "edge oracle" $\mathsf{EO}(e, \pi)$: determines if $e \in \mathsf{GMM}(G, \pi)$.

```
1 if we have already computed $\mathsf{EO}(e, \pi)$ then return the computed answer.;
2 Let $e_1, \ldots, e_k$ be all the edges incident to $e$ such that $\pi(e_1) < \ldots < \pi(e_k) < \pi(e)$.
3 for $i$ in $1 \ldots k$ do
4     if $\mathsf{EO}(e_i, \pi) = \mathsf{TRUE}$ then return FALSE;
5 return TRUE
```

Algorithm 2: "vertex oracle" $\mathsf{VO}(v, \pi)$: determines if a given vertex v is matched by $\mathsf{GMM}(G, \pi)$.

```
1 Let $e_1, \ldots, e_k$ be the edges incident to $v$ with $\pi(e_1) < \ldots < \pi(e_k)$.
2 for $i$ in $1 \ldots k$ do
3     if $\mathsf{EO}(e_i, \pi) = \mathsf{TRUE}$ then return TRUE;
4 return FALSE
```

We note that a similar local procedure can also be defined for the related Greedy MIS algorithm. The intuition is the same: If a vertex has no lower rank neighbor in the greedy MIS, it must itself be in the greedy MIS.

Algorithm 3: "MIS oracle" $\mathsf{MISO}(v, \pi)$: determines if $v \in \mathsf{GMIS}(G, \pi)$.

```
1 if we have already computed $\mathsf{MISO}(v, \pi)$ then return the computed answer.;
2 Let $u_1, \ldots, u_k$ be all the neighbors of $v$ such that $\pi(u_1) < \ldots < \pi(u_k) < \pi(v)$.
3 for $i$ in $1 \ldots k$ do
4     if $\mathsf{MISO}(u_i, \pi) = \mathsf{TRUE}$ then return $\mathsf{FALSE}$;
5 return $\mathsf{TRUE}$
```

Having defined these local simulation oracles, the next question is are they really faster than the naive approach that runs the whole algorithm? For some pathological permutation π, the local oracles above can be as slow as the naive approach. But when the permutations are random, the local simulations tend to be much faster.

The number of recursive calls needed to compute the oracle values is often called the "query-complexity" of these randomized greedy algorithms and if the starting vertex/edge is chosen as random, it is called the "average query-complexity."

For the maximal matching oracle, in particular, and for a vertex v (resp. edge e) let us use $T(v, \pi)$ (resp. $T(e, \pi)$) to denote the number of recursive calls to the edge oracle $\mathsf{EO}(\cdot, \pi)$ over the course of answering $\mathsf{VO}(v, \pi)$ (resp. $\mathsf{EO}(e, \pi)$). We emphasize that for some edge e', $\mathsf{EO}(e', \pi)$ may be called multiple times during the execution and we count all of these in $T(v, \pi)$, though only the first call to $\mathsf{EO}(e', \pi)$ may generate new recursive calls due to caching.

We prove the following result on the average query-complexity of RGMM in Chapter 6:

Theorem 3.2. *For a vertex $v \sim V$ chosen uniformly at random and for a permutation π chosen independently and uniformly at random,*

$$\mathop{\mathbf{E}}_{v \sim V, \pi}[T(v, \pi)] = O(\bar{d} \cdot \log n),$$

where recall that $n := |V|$ and $\bar{d}$ is the average degree of the graph.

Remark 3.3. *In a complete bipartite graph with $\Theta(\bar{d})$ vertices in one part and $\Theta(n)$ vertices in the other, for $n \gg \bar{d}$, the average query-complexity is $\Omega(\bar{d})$. Hence Theorem 3.2 is tight up to a logarithmic factor.*

The average query-complexity of RGMM and RGMIS were studied extensively prior to our work following the paper of Nguyen and Onak [133]. For the RGMIS algorithm, in particular, Yoshida, Yamamoto, and Ito [154] proved in a beautiful paper that for a vertex v chosen uniformly at random and for a random permutation π, the MIS oracle $\mathsf{MISO}(v, \pi)$

leads to a total of $O(\bar{d}+1)$ expected recursive calls to $\mathsf{MISO}(\cdot, \pi)$.

The result of Yoshida et al. [154] can also be adapted to the RGMM algorithm, but the bound would be weaker than Theorem 3.2. First observe that due to Observation 3.1, which asserts RGMM is equivalent to running RGMIS on the line-graph, the result of Yoshida et al. [154] as black-box implies the following bound on the query-complexity of a random *edge* (different from Theorem 3.2 which is for a random vertex):

Proposition 3.4 ([154]). *For an edge $e \sim E$ chosen uniformly at random and for a permutation π chosen independently and uniformly at random,*

$$\mathop{\mathbf{E}}_{e \sim E, \pi}[T(e, \pi)] = O(1 + L/m)$$

where L is the number of edges in the line-graph.

While it is not a black-box reduction, we note that the proof of [154] can be adapted to also work for random vertices. Particularly, it implies that for a random vertex v, $\mathbf{E}_{v \sim V, \pi}[T(v, \pi)] = O(1 + L/n)$. In general, L/n can be upper bounded by $O(\bar{d} \cdot \Delta)$, and there are graphs[1] for which $L/n = \Omega(\bar{d} \cdot \Delta)$. Hence, Theorem 3.2 improves the result of [154] by a factor of nearly Δ. This improvement is crucial for our time-optimal sublinear algorithms in Chapter 6.

3.3 Robustness Property

The next property of the randomized greedy algorithms that we highlight is their robustness. Namely, that the outputs of RGMM and RGMIS do not change much under changes to the underlying graph.

A Deterministic Robustness Property of GMM: We start with the greedy maximal matching algorithm and prove a robustness property for it which holds for all permutations (i.e., not necessarily a random permutation). The property is that modifying a single vertex or edge of G does not change the set of matched vertices too much. Note that that the set of *edges* in the matching can still change significantly.

Lemma 3.5 (Deterministic Lipschitz Properties of GMM). *Fix some graph $G = (V, E)$ and let $\rho : E \to [0, 1]$ be an associated list of priorities:*

[1] Consider a complete bipartite graph with $\Delta + 1$ vertices in one part and $\Theta(\bar{d})$ vertices in the other.

1. *If graph G' is derived by removing a vertex of G, then there are at most 2 vertices that are matched in exactly one of $\mathsf{GMM}(G,\rho)$ and $\mathsf{GMM}(G',\rho)$.*
2. *If graph G' is derived by removing an edge of G, then there are at most 2 vertices that are matched in exactly one of $\mathsf{GMM}(G,\rho)$ and $\mathsf{GMM}(G',\rho)$.*
3. *If ρ' is derived by changing a single entry of ρ, then there are at most 2 vertices that are matched in exactly one of $\mathsf{GMM}(G,\rho)$ and $\mathsf{GMM}(G,\rho')$.*

Proof. We start with the proof of the first part. Suppose that G' is obtained by removing some vertex v from G. Let $M := \mathsf{GMM}(G,\rho)$ and $M' := \mathsf{GMM}(G',\rho)$. Furthermore, let $D := M \oplus M'$ denote the symmetric difference of M and M', i.e. the set $(M \setminus M') \cup (M' \setminus M)$. Note that the match-status of a vertex v differs in M and M' if and only if its degree in D is one. Therefore, it suffices to show that there are at most two such vertices in D.

We first claim that D has at most one connected component (apart from singleton vertices). For sake of contradiction, suppose that D has multiple such connected components; fix one component C that does not contain v. Let e be the edge in C with the lowest rank. The fact that no lower rank edge that is connected to e is part of M or M' (otherwise e would not be the highest priority edge in C) shows that e has to belong to both M and M'. By definition of D, this means that $e \notin D$ which is a contradiction. Next, observe that since D is composed of the edges of two matchings, its maximum degree is at most 2 and thus its unique component is either a path or a cycle. The latter has no vertex of degree one and the former has two; proving part 1 of Lemma 3.5.

The proof of the other two parts of Lemma 3.5 follows from a similar argument. If an edge e is removed from G or its entry in ρ is changed, then for the same argument, the symmetric difference $M \oplus M'$ of the two greedy matchings M and M' that are obtained would contain only one connected component which has to contain e. Since this component is a cycle or a path, the match-statuses of at most two vertices are different in the two matchings. $\square$

While Lemma 3.5 holds for all permutations, if we in addition assume that the permutation is random, much more can be said about how the outputs of RGMM and RGMIS change. For example, let G' be obtained by removing a vertex or edge from a graph $G = (V,E)$. Then it is known that the outputs of $\mathsf{GMIS}(G,\rho)$ and $\mathsf{GMIS}(G',\rho)$ for a random rank function $\rho : V \to [0,1]$ differ in $O(1)$ vertices in expectation [64, Theorem 1]. We further show in Chapter 7 (Theorem 7.10) that even the number of vertices whose *eliminator* changes can be bounded by $O(\log n)$ where the eliminator of a vertex is its lowest-rank neighbor (or itself) that joins GMIS. (Note that the eliminator of a vertex may change without changing its MIS-status.) Applying these results on RGMIS to the line-graph, we can strengthen Lemma 3.5 and even bound the expected number of *edges* that leave/join GMM after an edge insertion/deletion by $O(1)$.

3.4 Sparsification Property

The next sparsification property asserts that if we run RGMIS (resp. RGMM) on the first p fraction of the vertices (resp. edges) in the random permutation, then in the remaining graph, the maximum degree can be bounded by $O(\frac{\log n}{p})$ w.h.p. This property has been used in various models of computation. Here we present a proof for RGMIS and the bound for RGMM follows as a corollary.

Lemma 3.6. *Fix a graph $G = (V, E)$, let $\rho : V \to [0, 1]$ be a random rank function, and let $p \in [0, 1]$ be a parameter. Let I_p be the subset of vertices v in $\mathsf{GMIS}(G, \rho)$ with $\rho(v) \le p$. Let G' be the graph obtained by removing the vertices in I_p and their neighbors from G'. With probability $1 - 1/n^4$, G' has maximum degree $\frac{5 \ln n}{p}$.*

Proof. Fix a vertex v; we prove that

$$\mathbf{Pr}\left[v \text{ belongs to } G' \text{ and } \deg_{G'}(v) > (5 \ln n)/p\right] \le 1/n^5. \tag{3.1}$$

The lemma then follows by a union bound over all n vertices.

To prove Eq (3.1), let us define $G^v := G[V \setminus \Gamma_G(v)]$ to be the graph obtained by removing v and all of its neighbors from G and let $I^v_p := \{v \in \mathsf{GMIS}(G^v, \rho) \mid \rho(v) \le p\}$. We prove Eq (3.1) holds even if the ranks of vertices in G^v are picked adversarially.

Let $L = \{u \in N_G(v) \mid \Gamma_G(u) \cap I^v_p = \emptyset\}$ be the set of neighbors of v that do not have a neighbor in I^v_p. Since the ranks are independent, we have

$$\mathbf{Pr}[\text{there exists } u \in L \text{ with } \rho(u) \le p] = 1 - (1 - p)^{|L|}. \tag{3.2}$$

Next, we claim that if there is a vertex $u \in L$ with $\rho(u) \le p$, then $I_p \ne I^v_p$. Suppose for the sake of contradiction that such u exists and $I_p = I^v_p$. The fact that $\rho(u) \le p$ implies $\Gamma_G(u) \cap I_p \ne \emptyset$ since I_p is a maximal independent set of vertices with rank $\le p$. Combined with $\Gamma_G(u) \cap I^v_p = \emptyset$ which follows from $u \in L$, this implies that there must be a vertex in $\Gamma_G(u)$ that belongs to I_p but not I^v_p, contradicting $I_p = I^v_p$. As such, if there is any vertex in

L with rank $\leq p$ then $I_p^v \neq I_p$ which combined with Eq (3.2) implies:

$$\mathbf{Pr}[I_p^v = I_p] \leq (1-p)^{|L|}. \tag{3.3}$$

Our next crucial observation is that if v belongs to G', then $I_p = I_p^v$. This follows from the fact that if $v \in G'$, then no vertex in $\Gamma_G(v)$ belongs to I_p and so removing $\Gamma_G(v)$ from the graph should not change the independent set up to rank p. Hence, by Eq (3.3),

$$\mathbf{Pr}[v \text{ belongs to } G'] \leq (1-p)^{|L|}. \tag{3.4}$$

To conclude the proof of Eq (3.1), note that if $|L| > (5\ln n)/p$, then by equation above the probability of v belonging to G' is at most $(1-p)^{(5\ln n)/p} \leq 1/n^5$. On the other hand, if $|L| \leq (5\ln n)/p$, then under the event that $v \in G'$, $I_p = I_p^v$ by the discussion above and thus $\deg_{G'}(v) = |L| \leq (5\ln n)/p$. □

As a corollary of Lemma 3.6, we also get the same degree reduction lemma for RGMM.

Lemma 3.7. *Fix a graph $G = (V, E)$, let $\rho : E \to [0,1]$ be a random rank function, and let $p \in [0,1]$ be a parameter. Let M_p be the subset of edges e in $\mathsf{GMIS}(G, \rho)$ with $\rho(e) \leq p$. Let G' be the graph obtained by removing all the edges connected to at least one edge in M_p. With probability $1 - 1/n^4$, G' has maximum degree $\frac{5\ln n}{p} + 1 = O(\frac{\ln n}{p})$.*

Proof. Since RGMM on G is equivalent to RGMIS on $L(G)$ by Observation 3.1, Lemma 3.6 bounds the maximum degree in $L(G')$ by $\frac{5\ln n}{p}$ with probability $1 - 1/n^4$. The lemma follows since the maximum degree in the line-graph is always at least as large as the maximum degree of the original graph minus one. □

Part I

Massively Parallel Computation

As we discussed in Chapter 1, modern parallel frameworks such as MapReduce [80], Hadoop [152], Spark [155], and their variants provide one of the most successful and widely applied approaches for processing massive data. But *what is the true computational power of modern parallel frameworks?*

To answer the question above, we first need a computational model. The *Massively Parallel Computations* (MPC) model which was first introduced by Karloff, Suri, and Vassilvitskii [109] and was further refined in the works of [98, 25, 8], serves as a clean abstraction of all of these frameworks without getting into the implementation details of these systems. Indeed, the MPC model has become the standard theoretical model for this purpose. We first give a slightly more formal definition of the model and then delve into our contributions.

The MPC Model: In the Massively Parallel Computations (MPC) model, an input of size N is initially distributed among M machines, each with a local space of size S. Computation proceeds in synchronous rounds in which each machine can perform an arbitrary local computation on its data for free, and can send messages to other machines. The messages are delivered at the start of the next round. Furthermore, the total messages sent or received by each machine in each round should not exceed its memory. The main parameter to optimize is the number of rounds that the algorithm takes while using a sublinear in N space per machine (i.e., $S = N^{1-\Omega(1)}$) and an, ideally, linear in N total space (i.e., $M \times S = O(N)$).

Our focus is particularly on graph problems in the MPC model. As before, we denote the input graph by $G = (V, E)$ and use n and m to denote the number of vertices and edges in G. The edge-set of G is initially distributed arbitrarily among the machines, meaning that $N = \Theta(m)$ words (or $\Theta(m \log n)$ bits). While the input size relates to m, it is actually often easier to parametrize the space per machine S by the number of vertices n instead of m. Two of the most popular regimes of parameters for graph algorithms in the MPC model include:

Near-linear Regime: Here the space per machine is $S = O(n)$. This regime of space is particularly useful for dense graphs whose edges do not fit the memory of a single machine, but the vertices do.

Strictly Sublinear Regime: Here the space per machine is $S = n^{\delta}$ for a constant $\delta < 1$. This regime of space is most useful for sparse graphs where even the vertex-set is too large and does not fit into one machine.

Chapter 4

Massively Parallel Maximal Matching

In this chapter, we consider the *maximal matching* problem in the MPC model. For many graph problems, including maximal matching, $O(\log n)$ round MPC algorithms can be achieved in a straightforward way by simulating traditional parallel (PRAM) algorithms [126, 104, 6] using $n^{\Omega(1)}$ space. But can we do better? Particularly, many graph problems are known to admit $\text{poly}(\log\log n)$ or $O(1)$ round MPC algorithms. Can we achieve the same for maximal matching? This is the main question that we attempt to address in this chapter.

Related Work: One of the first algorithms for the maximal matching problem in the MPC model was given by Lattanzi, Moseley, Suri, and Vassilvitskii [119]. Allowing a super-linear in n space of $n^{1+\delta}$, the algorithm of [119] terminates in $O(1/\delta)$ rounds. Note, however, that with $O(n)$ space, this algorithm still requires $\Omega(\log n)$ rounds. For the related problem of approximate matching (i.e., relaxing maximality) Czumaj, Lacki, Madry, Mitrovic, Onak, and Sankowski [78] in a breakthrough result gave a $\text{poly}(\log\log n)$-round algorithm that uses $O(n)$ local space. The round-complexity was soon after strengthened to $O(\log\log n)$ with the same space requirement [93, 15]. Unfortunately, however, this progress on approximate matching offers no help for maximal matching. The reason, roughly speaking, is that these algorithms leave a (small) constant fraction of the remaining (matchable) vertices unmatched in each round and thus they also require up to $\Omega(\log n)$ rounds to ensure maximality of the solution. As a result, despite this remarkable progress the $O(\log n)$-round algorithms of the 1980's remained the fastest for maximal matching in the MPC model with $O(n)$ space. An improvement over this bound was later achieved by Ghaffari and Uitto [92] who presented an $\widetilde{O}(\sqrt{\log n})$ round algorithm in the strictly sublinear regime.

Our Contribution: Our main result in this section is an exponentially faster algorithm for maximal matching:

Theorem 4.1 ([45])**.** *Given an n-vertex graph G with m edges and max degree Δ, there exists a randomized* MPC *algorithm for computing a maximal matching that*

(1) takes $O(\log\log\Delta)$ rounds using $O(n)$ space per machine,

(2) or takes $O(\log\frac{1}{\delta})$ rounds using $O(n^{1+\delta})$ space per machine, for any $\delta \in (0,1)$.

The algorithm succeeds with probability $1-e^{-n^{\Omega(1)}}$ and requires $O(m)$ total space.

Theorem 4.1 part (1) provides the first $\log^{o(1)} n$ round MPC algorithm for maximal matching that does not require a super-linear space in n. In fact, it improves exponentially over the prior algorithms in this regime, which all take $\log^{\Omega(1)} n$ rounds [126, 119, 92]. Furthermore, Theorem 4.1 part (2) exponentially improves over the δ-dependency of Lattanzi *et al.*'s algorithm [119] which requires $O(1/\delta)$ rounds using $O(n^{1+\delta})$ space.

We further show in [45] that the space can be slightly improved to $n/2^{\Omega(\sqrt{\log n})}$ while still taking $O(\log\log n)$ rounds. For simplicity, however, we will not go through the details of this latter result in this thesis.

While Theorem 4.1 requires a space close to linear in n, we further show in the following theorem that a poly$(\log\log n)$-round algorithm also exists in the strictly sublinear regime provided that the arboricity λ of the graph is not too large. This result is particularly interesting because the main motivation behind the strictly sublinear space regime is indeed the case of sparse graphs, and low arboricity graphs include most sparse graphs of interest such as planar graphs, minor-free graphs, graphs of bounded degree, bounded genus, bounded treewidth, etc.

Theorem 4.2 ([37])**.** *Given an n-vertex graph G with m edges and arboricity λ, there exists an $O(\sqrt{\log\lambda}\cdot\log\log\lambda+\log^2\log n)$ round randomized* MPC *algorithm for computing a maximal matching (and a maximal independent set) of G. The algorithm can be adapted to use a local space of $O(n^\delta)$ for any fixed $\delta > 0$ and requires a total space of $O(m)$.*

Our main focus in this chapter will be on proving Theorem 4.1, and will only mention the key ideas behind the proof of Theorem 4.2 in Section 4.5.

Conceptual Contribution

We prove Theorem 4.1 by providing a novel analysis of an extremely simple and natural algorithm. The algorithm edge-samples the graph, randomly partitions the vertices into disjoint subsets, and finds a greedy maximal matching within the induced subgraph of each partition. This partitioning is useful since each induced subgraph can be sent to a different machine. We show that if we commit the edges of each of these greedy matchings to the final output, the vertex degrees in the residual graph are drastically dropped. This resolves a conjecture of Czumaj, Lacki, Madry, Mitrovic, Onak, and Sankowski [78] who suspected that a variant of this algorithm might work and left its analysis as one of their main open problems:

> "*Finally, we suspect that there is a simpler algorithm for the problem* [...] *by simply greedily matching high-degree vertices on induced subgraphs* [...] *in every phase. Unfortunately, we do not know how to analyze this kind of approach.*"

We give a high-level overview of the analysis in Section 4.1.

Other Implications

Our algorithm also has a few other implications when used as a black-box.

Corollary 4.3. *By a well-known reduction, the set of matched vertices in a maximal matching is a 2-approximation of minimum vertex cover. As such, algorithms of Theorem 4.1 can also be applied to 2-approximation of minimum vertex cover.*

The problem of whether an approximate vertex cover can be found faster in MPC with $O(n)$ space was first asked by Czumaj *et al.* [78]. Subsequent works showed that indeed $O(\log\log n)$ algorithms are achievable and the approximation factor has been improved from $O(\log n)$ to $O(1)$ to $(2+\varepsilon)$ [11, 93, 15]. Corollary 4.3 reaches a culminating point: If we restrict the machines to run a polynomial-time algorithm, which is a standard assumption (see [109, 8]), no algorithm can achieve a better approximation under the Unique Games Conjecture [111].

Corollary 4.4. *By known reductions [34, 120], Theorem 4.1 implies an $O(\log\log\Delta)$ round algorithm for maximal matching in the congested clique model. It also leads to $O(\log\log\Delta)$ round congested clique algorithms for 2-approximate vertex cover, $(1+\varepsilon)$ approximate maximum matching, and $(2+\varepsilon)$ approximate maximum weighted matching by known reductions.*

Prior to our work, the fastest known algorithm for maximal matching in the congested clique model required $\widetilde{O}(\sqrt{\log n})$ rounds [92]. Corollary 4.4 exponentially improves this.

Corollary 4.5. *For any constant $\varepsilon \in (0, 1)$, Theorem 4.1 can be used to give algorithms for $(1 + \varepsilon)$ approximate matching and $(2 + \varepsilon)$ approximate maximum weighted matching in asymptotically the same number of rounds and space.*

The reduction from maximal matching to $(1 + \varepsilon)$ approximate matching is due to an algorithm by McGregor [128] (see [15]) and the reduction to $(2 + \varepsilon)$ approximate weighted matching is due to an algorithm by Lotker, Patt-Shamir, and Rosén [124] (see [78]). We also note that if the space is $O(n \operatorname{polylog} n)$, then our algorithm can be used in a framework of Gamlath, Kale, Mitrovic, and Svensson [90] to get an $O(\log\log\Delta)$ round algorithm for $(1+\varepsilon)$ approximate maximum weighted matching.

Finally, we note that Corollary 4.5 strengthens the round-complexity of the results in [78, 93, 15] from $O(\log\log n)$ to $O(\log\log\Delta)$ using $O(n)$ space. To our knowledge, the algorithms of [78, 93, 15] do require $\Omega(\log\log n)$ rounds even when $\Delta = \operatorname{poly}\log n$ since they switch to an $O(\log\Delta)$ round algorithm at this threshold. Corollary 4.5, however, implies an $O(\log\log\log n)$ round algorithm on such graphs.

Recent Development

After the first publication of Theorem 4.1 in [45], Nowicki and Onak [134] proved that the same algorithm can be used to maintain a maximal matching efficiently in a dynamic variant of the MPC model, where batches of edge updates (insertions/deletions) have to be handled.

4.1 High Level Technical Overview

As discussed above, if the space per machine is $n^{1+\Omega(1)}$, we already know how to find a maximal matching efficiently [119]. The main problem, roughly speaking, is that once the space becomes $O(n)$, the computational power of *a single machine* alone does not seem to be sufficient to have a significant effect on the whole graph. More concretely, the known algorithms that work based on ideas such as edge-sampling the graph into a single machine and finding a matching there [119, 30, 1], all require $\Omega(\log n)$ rounds of repeating this procedure if the space is $O(n)$.

Vertex partitioning [109, 23, 78, 15, 93], which in the context of matching was first used by [78], helps in utilizing several machines. The general idea is to randomly partition the vertices and find a matching in the induced subgraph of each partition individually in a different machine. It turns out that the choice of the internal matching algorithm over these induced subgraphs, has a significant effect on the global progress made over the whole graph. This is, in fact, the fundamental way that the algorithms within this framework differ [78, 15, 93].

For the internal matching algorithm, we use randomized greedy maximal matching (see Chapter 3). Recall that this procedure iterates over the edges in a random order, and at the time of processing each edge, adds it to the matching iff none of its incident edges are part of the matching so far. We give a brief overview of our algorithm first, then describe the key ideas behind its analysis.

The algorithm. Our main algorithm, which is formalized as Algorithm 5, uses three randomization steps, all of which are necessary for the analysis:

- An ordering π over the edges is chosen uniformly at random.
- Each edge of the graph is sampled independently with some probability p.
- For some k, the vertex set V is partitioned into disjoint subsets $V_1, \dots, V_k$ where the partition of each vertex is chosen independently at random.

After these steps, for any $i \in [k]$, we put the edge-sampled induced subgraph of V_i into machine i and compute a greedy maximal matching M_i according to ordering π. We note that the choice of k and p in Algorithm 5 ensure that the induced subgraphs fit the memory of a machine.

The analysis outline. Observe that $M = \bigcup_{i \in [k]} M_i$ is a valid matching since the partitions are vertex disjoint. The key to our results is to show that if we commit the edges of M to the final maximal matching, then the degree of almost all vertices drops to $\Delta^{1-\Omega(1)}$ in the residual graph. The main challenge here is to bound the vertex degrees across the partitions.

To do this, for any vertex v and any partition $i \in [k]$, we let $Z_{v,i}$ denote the number of neighbors of v in partition i that remain unmatched in greedy matching M_i. Note that $Z_{v,i}$ is a random variable of the three randomizations involved in the algorithm, and that $\sum_{i \in [k]} Z_{v,i}$ is precisely equal to the remaining degree of vertex v. We show the abovementioned degree reduction guarantee through a concentration bound on random variable $Z_{v,i}$.

Let us first outline how a concentration bound on $Z_{v,i}$ can be useful. Suppose, wishfully thinking, that $Z_{v,i} = (1 \pm o(1))\, \mathbf{E}[Z_{v,i}]$ for every $i \in [k]$ with high probability. By symmetry of the partitions, we have $\mathbf{E}[Z_{v,i}] = \mathbf{E}[Z_{v,1}]$ for every $i \in [k]$. This means that all random variables $Z_{v,1}, \dots, Z_{v,k}$ take on the same values ignoring the lower terms. Now, if $\mathbf{E}[Z_{v,1}]$ is small enough that $k \cdot \mathbf{E}[Z_{v,1}] < \Delta^{1-\Omega(1)}$, we get the desired bound on residual degree of v. Otherwise, due to the huge number of unmatched neighbors in its own partition, we show that v must have been matched and, thus, cannot survive to the residual graph!

Unfortunately, $Z_{v,i}$ is a rather complicated function and it is not straightforward to prove such sharp concentration bounds on it. Recall that Chernoff-Hoeffding bounds work only on sum of independent random variables. Furthermore, concentration bounds obtained by Azuma's or other "dimension dependent" inequalities seem useless for our purposes: because the partition of every vertex in the graph may potentially affect $Z_{v,i}$, these would give bounds on the order of $Z_{v,i} = \mathbf{E}[Z_{v,i}] \pm \widetilde{O}(\sqrt{n})$. As $\mathbf{E}[Z_{v,i}]$ should be on the order of Δ, this is useless when Δ is small.

Instead of an exponential concentration bound, we aim for a weaker concentration bound by proving an upper bound on the variance of $Z_{v,i}$. To achieve this upper bound, we use a method known as the Efron-Stein inequality (Proposition 2.2) which plays a central role in our analysis. On one hand, this weaker concentration bound is still strong enough for our purpose of degree reduction. On the other hand, since we are only bounding the variance, the required conditions are much more relaxed and can be shown to be satisfied by the algorithm.

Recall that to bound the variance using the Efron-Stein inequality, we have to show that sum of changes to the function by re-drawing its entries is not too large. To do this, we use the bounds on the average query-complexity of RGMM (see Section 3.2). Specifically, if we can determine whether vertex v is matched by querying only few others, modifying other parts of the graph should not affect v's status. This application of average query-complexity is rather surprising and in the contrary to how it has been used in the literature (mostly for sublinear time algorithms). Indeed, our main conceptual contribution of this chapter is to show that:

sublinear-time algorithms are useful for proving concentration bounds too!

4.2 The Degree Reduction Algorithm

As discussed in Section 4.1, the key to proving Theorem 4.1 is an algorithm to reduce the graph degree by a polynomial factor. The precise statement of this lemma is as follows:

Lemma 4.6 (degree reduction)**.** *There is an $O(1)$ round* MPC *algorithm to produce a matching M, with the following behavior w.e.h.p.: it uses $n/\Delta^{\Omega(1)}$ space per machine and $O(m)$*

space in total, and the residual graph $G[V \setminus M]$ has maximum degree $\Delta^{1-\Omega(1)}$.

Our main result, and the technical core of our analysis in proving Lemma 4.6 lies in showing that the following Algorithm 5 significantly reduces the degree of most vertices.

Algorithm 4: This algorithms gets a graph $G = (V, E)$ with maximum degree Δ and outputs a matching M of G.

```
1 Permutation: Choose a permutation $\pi$ uniformly at random over the edges in $E$.
2 Edge-sampling: Let $G^L(V, L)$ be an edge-sampled subgraph of $G$ where each edge in
    $E$ is sampled independently with probability $p := \Delta^{-0.85}$.
3 Vertex partitioning: Partition the vertices of $V$ into $k := \Delta^{0.1}$ groups $V_1, \dots, V_k$ such
    that the partition of every vertex in $V$ is chosen independently and uniformly at
    random.
4 Each machine $i \in [k]$ receives the graph $G^L[V_i]$ and finds the greedy maximal matching
    $M_i := \mathsf{GMM}(G^L[V_i], \pi)$.
5 Return matching $M := \bigcup_{i=1}^{k} M_i$.
```

Some notation: When it is clear from the context, we abuse notation to use M for the vertex set of matching M. In particular, we use $G[V \setminus M]$ to denote the graph obtained by removing every vertex of M from G. Furthermore, for any vertex $v \in V$ and matching M, we define the *residual degree* $\deg_M^{\text{res}}(v)$ to be zero if $v \in M$, and otherwise $\deg_M^{\text{res}}(v) := \deg_{G[V \setminus M]}(v)$. Finally, we define the *match-status* of vertex v according to some matching M to be the indicator for the event that $v \in M$.

Specifically, we prove the following properties for Algorithm 5.

Lemma 4.7. *Algorithm 5 has the following desirable behavior:*

1. *W.e.h.p., it uses $n/\Delta^{\Omega(1)}$ space per machine.*
2. *W.e.h.p., it uses $O(n) + m/\Delta^{\Omega(1)}$ space in total (aside from storing the input graph.)*
3. *The expected number of vertices $v \in V$ such that $\deg_M^{\mathit{res}}(v) > \Delta^{0.99}$ is $O(n/\Delta^{0.03})$.*

We will prove Lemma 4.7 in Section 4.3 and we will prove Lemma 4.6 in Section 4.4. Before this, let us show how the degree reduction algorithm of Lemma 4.6 can be used to prove Theorem 4.1.

Proof of Theorem 4.1. The algorithm consists of r iterations that each commits a number of edges to the final maximal matching using the algorithm of Lemma 4.6. In each iteration, the maximum degree in the remaining graph is reduced from Δ to $\Delta^{1-\alpha}$ given that $\Delta > c$

for some constant c and α. This ensures that by the end of iteration r, maximum degree is at most $\max\{c, \Delta^{(1-\alpha)^r}\}$.

To get the first result, take $r = \Theta(\log\log\Delta)$; at the end of this process, the residual graph has degree $O(1)$. At this point, we put the entire residual graph onto a single machine, and compute its maximal matching. To get the second result, take $r = \Theta(\log(1/\delta))$; at the end of this process, the residual graph has degree n^δ. At this point, we again put the entire residual graph onto a single machine, and compute its maximal matching. □

4.3 Matching Almost All High-Degree Vertices

We now turn to proving Lemma 4.7. We first need some notation for the analysis of Algorithm 5. For simplicity, we write G_i for the graph $G[V_i]$. Note that G_i is different from G_i^L in that G_i^L includes only a subset of the edges in G_i; those that were sampled in Line 2 of Algorithm 5. We let L_i be the set of edges $\{u, v\} \in L$ with $u, v \in V_i$; that is, L_i is the edge-set of G_i^L. We further define χ to be the partition function of the vertices; that is, each vertex select a value $\chi(v)$ u.a.r from $[k]$, and then we set $V_i = \chi^{-1}(i)$. We also note that throughout the proof, we assume $m \geq n^{0.9}$. This assumption comes w.l.o.g. since otherwise one can put all the edges into one machine with even sublinear memory of $O(n^{0.9})$ and find a maximal matching there.

We begin by analyzing the residual degree of a vertex within its own partition, which are some simple consequences of the method used to generate L.

Claim 4.8. *The following bounds on the edge set L hold w.e.h.p.:*

1. *Every $i \in [k]$ has $|V_i| = \Theta(n/\Delta^{0.1})$.*
2. *The graph G^L contains $O(m/\Delta^{0.85})$ edges.*
3. *Each graph G_i^L contains $O(n/\Delta^{0.05})$ edges.*

Proof. The first property follows from a straightforward Chernoff bound, noting that $\mathbf{E}[V_i] = n/k = n/\Delta^{0.1} \geq \text{poly}(n)$. For the second property, observe that the expected number of edges in G^L is $m \cdot p = m/\Delta^{0.85}$. As we have discussed above, we can assume that $m \geq n^{0.9}$ and we also know that $\Delta \leq n$; therefore, $m/\Delta^{0.85} \geq n^{0.05}$ and by Chernoff's bound the number of such edges is $O(m/\Delta^{0.85})$ w.e.h.p. For the third property, we consider two cases where $\Delta \geq n^{0.01}$ and $\Delta < n^{0.01}$ separately.

Case 1: $\Delta \geq n^{0.01}$. For each vertex $v \in V_i$, its incident edge $e = \{u, v\}$ will belong to G_i^L if e is sampled in L and vertex u also belongs to V_i. Both of these events occur at the same time with probability $p \cdot k^{-1} = \Delta^{-0.95}$. This means that the expected number of neighbors of v in G_i^L will be $\Delta \cdot \Delta^{-0.95} = \Delta^{0.05}$. Since we assumed $\Delta \geq n^{0.01}$, a simple Chernoff bound can show that this random variable is concentrated around $O(\Delta^{0.05})$ w.e.h.p. Combined with the first property, the number of edges in each G_i^L will be $O(n/\Delta^{0.1}) \cdot O(\Delta^{0.05}) = O(n/\Delta^{0.05})$ w.e.h.p.

Case 2: $\Delta < n^{0.01}$. Let U denote the number of edges in G_i^L. For the arguments discussed above, we still have $\mathbf{E}[U] \leq O(n/\Delta^{0.05})$. Furthermore, U can be regarded as a function of the vertex partition χ and the edge set L. There are $O(n\Delta)$ such random variables, and each of these can change U by at most Δ. Therefore, by Corollary 2.4, w.e.h.p., we have

$$U \leq \mathbf{E}[U] + \Delta \cdot n^{0.01} \cdot \sqrt{O(n\Delta)};$$

as $\Delta \leq n^{0.01}$ this in turn implies that $U \leq O(n/\Delta^{0.05})$ w.e.h.p. □

These allow us to prove the first two parts of Lemma 4.7:

Proof of Lemma 4.7 part 1 and 2. For the space bounds, Claim 4.8 shows that for each G_i^L, we require $O(n/\Delta^{0.05})$ space for its edges and $O(n/\Delta^{0.1})$ for its vertices. Since Δ is larger than any constant, this is smaller than $n/\Delta^{\Omega(1)}$. To show the bounds on total space usage note that the total edge count of all the graphs G_i^L is clearly at most $|L|$, since each edge lives on at most one machine, and this is at most $m/\operatorname{poly}(\Delta)$. Furthermore, storing partition of each vertex requires only $O(n)$ total space. □

As we have discussed before, for any vertex $v \in V$ and any $i \in [k]$, we define the random variable

$$Z_{v,i} := \left|V_i \cap N_{G[V \setminus M]}(v)\right|,$$

to be the degree of vertex v in the i^{th} partition of the residual graph $G[V \setminus M]$. Note here that v does not necessarily belong to V_i. With this definition, if a vertex v is not matched in M, we have $\deg_M^{\text{res}}(v) = Z_{v,1} + \cdots + Z_{v,k}$. We further define the related random variable Z'_v as:

$$Z'_v := \begin{cases} Z_{v,\chi(v)} & \text{if } v \notin M \\ 0 & \text{if } v \in M, \end{cases}$$

which is equivalent to the residual degree of v in its own partition.

Claim 4.9. *For any vertex v, we have* $\mathbf{Pr}(Z'_v > \Delta^{0.86}) \leq \exp(-\operatorname{poly}(\Delta))$.

Proof. We will show that this bound holds, even after conditioning on the random variables χ and π. Suppose now that $v \in V_i$ and so we need to bound the probability that $Z_{v,i} > \Delta^{0.86}$. Note, here, that $Z'_v = \deg^{\text{res}}_{M_i}(v)$. Also, M_i is formed by performing independent edge sampling on $G[V_i]$ and then taking the greedy maximal matching. Applying the degree reduction property of random greedy maximal matching (Lemma 3.7), the probability that $Z'_v > \frac{\ln(1/\beta)}{p}$ can be bounded by β.[1] Setting $\beta = e^{-\Delta^{0.01}}$, we have $Z'_v > \Delta^{0.86}$ with probability at most $\exp(-\operatorname{poly}(\Delta))$. □

4.3.1 Analysis of the Inter-partition Degrees

The key to analyzing Algorithm 5 is to show that for most vertices v, the values of $Z_{v,i}$ take on similar values across all possible indices i. We had sketched how this leads to the desired bound on vertex degrees in Section 4.1; let us provide some more technical details here.

Recall from Section 4.1 that our concentration inequalities should not have an additive factor depending on n or they become too weak to be useful as Δ gets smaller. To overcome this, we show that with careful analysis, the Efron-Stein inequality (Proposition 2.2) yields our desired concentration bound; in particular, it gives concentration on the order $Z_{v,i} = \mathbf{E}[Z_{v,i}] \pm \Delta^{1-\Omega(1)}$. However, we emphasize that this concentration bound is *not* with exponentially high probability, or even with high probability: it only holds with a relatively small probability $1 - 1/\operatorname{poly}(\Delta)$. This is the reason that we can only show that the number of high-degree vertices reduces by a $1/\operatorname{poly}(\Delta)$ factor, and not that Algorithm 5 reduces the maximum degree outright.

Due to symmetry, we may consider showing a concentration bound for $Z_{v,1}$. Let us furthermore assume that L and π have been fixed. Therefore, $Z_{v,1}$ becomes only a function of the vertex partitioning χ, or more precisely, a function of the set of vertices that belong to partition V_1. Let us define the vector $\vec{x}$, by setting $x_v = 1$ if $\chi(v) = 1$, and $x_v = 0$ otherwise. We may write $Z_{v,1}(\vec{x})$ to emphasize that $Z_{v,1}$ is merely a function of $\vec{x}$. Observe that $\vec{x}$ is a vector of n i.i.d. Bernoulli-$1/k$ random variables. To use the Efron-Stein inequality for bounding the variance, we have to upper bound the right-hand-side of inequality

$$\mathbf{Var}(Z_{v,1}) \leq \frac{1}{2} \mathop{\mathbf{E}}_{\vec{x}} \Big[\sum_{w \in V} \big(Z_{v,1}(\vec{x}) - Z_{v,1}(\vec{x}^{(w)}) \big)^2 \Big], \tag{4.1}$$

[1] We note that Lemma 3.7 is proved specifically for $\beta = 1/\operatorname{poly}(n)$ and the proof has to be slightly modified for this guarantee; but the idea is essentially the same.

where $\vec{x}^{(w)}$ is obtained by replacing the value of x_w in $\vec{x}$ with x'_w which is drawn independently from the same distribution. In other words, the w summand of (4.1) corresponds to the effect of repartitioning vertex w on the value of $Z_{v,1}$. Thus, we need to show that for most of the vertices in V, whether they belong to V_1 or not does not affect $Z_{v,1}$.

To show this, consider a game where we determine $Z_{v,1}(\vec{x})$ by querying entries of $\vec{x}$. The queries can be conducted adaptively, i.e., each query can depend on the answers to previous queries. If we show an upper bound β_v on the number of queries required to determine $Z_{v,1}(\vec{x})$, then no matter what the other $n - \beta_v$ entries of $\vec{x}$ are, $Z_{v,1}(\vec{x})$ remains unchanged and so clearly $Z_{v,1}(\vec{x}) - Z_{v,1}(\vec{x}^{(w)}) = 0$ for all such unqueried vertices w. (The subscript v in β_v is used to emphasize that the upper bound can be different for different choices of v.) Therefore, one way to show that most vertices of V do not affect $Z_{v,1}$ is to design an efficient query process. We also note a particularly useful property of the Efron-Stein inequality in (4.1) is that even an upper bound on the expected number (taken over choice of $\vec{x}$) of queries suffices.

In addition to showing that most vertices do not affect $Z_{v,1}$, we also need to show that the query process yields an appropriate Lipschitz property on $Z_{v,1}$ as well. That is, even if the query process can guarantee $Z_{v,1}(\vec{x}) - Z_{v,1}(\vec{x}^{(w)}) = 0$ for most vertices w, we still have to bound the value of $(Z_{v,1}(\vec{x}) - Z_{v,1}(\vec{x}^{(w)}))^2$ on those vertices w where $Z_{v,1}(\vec{x}) \neq Z_{v,1}(\vec{x}^{(w)})$. This also follows from the nice structure of the greedy maximal matching algorithm.

Claim 4.10 (Lipschitz property)**.** *For any vertex partitioning $\vec{x}$, let $\vec{x}^{(w)}$ be obtained by changing the w index of $\vec{x}$. Then $(Z_{v,1}(\vec{x}) - Z_{v,1}(\vec{x}^{(w)}))^2 \leq 4$.*

Proof. Suppose that $x_w = 0$ which means $x_w^{(w)} = 1$. Let V_1 and V'_1 denote the vertex partitions due to $\vec{x}$ and $\vec{x}^{(w)}$ respectively, i.e., $V_1 = \{u \mid x_u = 1\}$ and $V'_1 = \{u \mid x_u^{(w)} = 1\}$. Observe that V_1 and V'_1 differ in only one vertex w which belongs to V'_1 but not V_1. Define $M_1 := \mathsf{GMM}(G[V_1], \pi)$ and $M'_1 := \mathsf{GMM}(G[V'_1], \pi)$. By Lemma 3.5 part 1, there are at most two vertices in V whose match-status differs between M_1 and M'_1. Even if these two vertices happen to be neighbors of v, we still have $|Z_{v,1}(\vec{x}) - Z_{v,1}(\vec{x}^{(w)})| \leq 2$ and thus get the desired bound. The case with $x_w = 1$ and $x_w^{(w)} = 0$ follows from a similar argument. □

The Lipschitz property can be plugged directly into (4.1) to show $\mathbf{Var}(Z_{v,1}) \leq O(n\Delta^{-0.1})$. In what follows, however, we describe a query process which significantly reduces this upper bound to poly(Δ) for nearly all the vertices, i.e., removes the dependence on n.

The query process. We start with a query process to determine whether a given edge belongs to matching $M_1(\vec{x})$ – where here we write $M_1(\vec{x})$ to emphasize that the parameters π, L should be regarded as fixed and so matching M_1 is only a function of the vertex partitioning $\vec{x}$. This process is very similar to a generic edge oracle for the greedy matching (which we discussed in Section 3.2), except that instead of querying the edges, it queries the entries of the vector $\vec{x}$.

Suppose that we have to determine whether a given edge $e \in L$ belongs to the matching $M_1(\vec{x})$. Instead of revealing the whole vector $\vec{x}$, first note that if one of the end-points of e does not belong to V_1, then e cannot be in the induced subgraph G_1^L and thus we can answer NO immediately. Suppose that e appears in G_1^L. Since the greedy maximal matching algorithm processes the edges in the order of π, it suffices to recursively determine whether any of the incident edges to e belongs to $M_1(\vec{x})$ in the order of their priorities. At any point that we find such incident edge to e, we immediately return NO as e certainly cannot join $M_1(\vec{x})$. Otherwise e has to join $M_1(\vec{x})$, thus we return YES. We summarize the resulting query process as $\mathsf{EO}_\pi(e, \vec{x})$:

Algorithm 5: $\mathsf{EO}_\pi(e, \vec{x})$: A query-process to determine whether $e \in M_1(\vec{x})$.

```
1 Let $e = \{u, v\}$. Query $x_u$ and $x_v$; if $x_u = 0$ or $x_v = 0$, then return FALSE.
2 Let $e_1, \dots, e_d$ be the incident edges to $e$ in $G^L$ sorted as $\pi(e_1) < \pi(e_2) < \cdots < \pi(e_d)$.
3 for $i = 1, \dots, d$ do
4     if $\pi(e_i) < \pi(e)$ then
5         if $\mathsf{EO}_\pi(e_j, \vec{x})$ = YES then return FALSE
6 return TRUE
```

We also define a *degree oracle* $\mathcal{DO}_\pi(v, \vec{x})$ to determine the value of $Z_{v,1}(\vec{x})$. This checks whether each $w \in N_G(v)$ appears in V_1 and is matched, which in turn requires checking whether every edge incident to w appears in matching of $G^L[V_1]$:

Algorithm 6: $\mathcal{DO}_\pi(v, \vec{x})$: A query process to determine the value of $Z_{v,1}(\vec{x})$.

```
1 $c \leftarrow 0$
2 for all vertices $u \in N_G(v)$ do
3     Query $x_u$. if $x_u = 1$ then
4         Execute $\mathsf{EO}_\pi((u, w), \vec{x})$ for all vertices $w \in N_{G^L}(u)$.
5         if $\mathsf{EO}_\pi((u, w), \vec{x})$ = NO for all such vertices $w$ then
6             $c \leftarrow c + 1$                              ▷ $u$ is unmatched in $M_1$
7 return $c$
```

Analysis of the query complexity. We now analyze the *query complexity* of the oracle $\mathcal{DO}_\pi$, i.e., the number of indices in $\vec{x}$ that it queries. For any vertex v, we let $B(v)$ denote the number of vertices that are queried when running $\mathcal{DO}_\pi(v)$. This is precisely the quantity that we need to bound for arguing that $\mathbf{Var}(Z_{v,1})$ is small according to (4.1). Formally:

Claim 4.11. *Fix any $\vec{x}, \pi, L$ and and let $\vec{x}^{(w)}$ be a vector obtained by resampling the index x_w. Then*

$$\sum_{w \in V} (Z_{v,1}(\vec{x}) - Z_{v,1}(\vec{x}^{(w)}))^2 \leq 4B(v).$$

Proof. By definition, the value of $Z_{v,1}(\vec{x})$ can be uniquely determined by only revealing indices of $\vec{x}$ which are quered by $\mathcal{DO}_\pi(v, \vec{x})$. Therefore, changing other indices w of $\vec{x}$ cannot affect $Z_{v,1}$ and so $Z_{v,1}(\vec{x}) - Z_{v,1}(\vec{x}^{(w)}) = 0$. There are $B(v)$ indices queries by v. For any such index w, Claim 4.10 shows that $(Z_{v,1}(\vec{x}) - Z_{v,1}(\vec{x}^{(w)}))^2 \leq 4$. □

To bound $B(v)$, let us first define $A(e)$ for an edge $e \in L_1$ to be the number of edges in L_1, on which the edge oracle is called (recursively) in the course of running $\mathsf{EO}_\pi(e, \vec{x})$. Note that when running EO, only edges that are in L_1 can generate new recursive calls; other edges are checked, but immediately discarded.

Claim 4.12. *We have $\mathbf{E}_{\chi,L,\pi}[\sum_{e \in L_1} A(e)] \leq O(n)$.*

Proof. Let us first suppose that the random variables L and χ are fixed. Thus also G_i^L is determined. The only randomness remaining is the permutation π. As we are only interested in edges of L_1, the edges outside L_1 have no effect on the behavior of EO_π. Thus, $A(e)$ is essentially the query complexity of $\mathsf{GMM}(G_1, \pi)$ under a random permutation. By Proposition 3.4, we have:

$$\mathop{\mathbf{E}}_{\pi}\Big[\sum_{e \in L_1} A(e) \mid L, \chi\Big] \leq O(|L_1| + |R_1|),$$

where R_1 is the set of intersecting edge pairs in G_1. Integrating now over the random variables L and χ, we get:

$$\mathbf{E}\Big[\sum_{e \in L_1} A(e)\Big] \leq O(\mathbf{E}[|L_1| + |R_1|]).$$

Each edge $e \in E$ goes into L_1 with probability $p/k^2 = \Delta^{-1.05}$, and so $\mathbf{E}[|L_1|] = m\Delta^{-1.05}$. Likewise, G contains at most $m\Delta/2$ pairs of intersecting edges and each of these survives to R_1 with probability $p^2/k^3 = \Delta^{-2}$. Therefore, $\mathbf{E}[|R_1|] \leq m\Delta^{-1}$. Since $m \leq n\Delta$, we therefore get $\mathbf{E}[\sum_{e \in L_1} A(e)] \leq O(n)$. □

Claim 4.13. *Suppose that we condition on the event that when running $\mathcal{DO}_\pi(v, \vec{x})$, we make a total of t calls to $\mathsf{EO}_\pi(e, \vec{x})$ with $e \in L_1$. Then the expected number of total entries of $\vec{x}$ queried during $\mathcal{DO}_\pi(v, \vec{x})$ is at most $O(\Delta^{1.15} + t\Delta^{0.15})$.*

Proof. Let us condition on the random variables χ, L_1 and π. This determines the full listing of all edges in L_1 that are queried during the execution of $\mathcal{DO}_\pi(v)$, because only such edges can generate new recursive calls to EO_π. Thus, if we show that this bound holds conditioned on χ, L_1, π it will also show that it holds conditioned on the value t. The only remaining randomness at this point is the set $L \setminus L_1$.

Let J denote the set of edges in L_1 queried during $\mathcal{DO}_\pi(v, \vec{x})$, with $|J| = t$. Then $\mathcal{DO}_\pi(v, \vec{x})$ will query x_u for all $u \in N_G(v)$, and it will query w for all $w \in N_{G^L}(u)$ for all such $u \in N_G(v)$. Finally, whenever it encounters edge $e \in J$, it will call $\mathsf{EO}_\pi(f, \vec{x})$ for some edges $f \in L \setminus L_1$ which touch e; each of these will query two vertices, but the query process will not proceed further when they are discovered to lie outside L_1.

The number of vertices $u \in N_G(v)$ queried is clearly at most Δ. Now let us fix some $u \in N_G(v)$ and count the number of vertices $w \in N_{G^L}(u)$ queried. This is precisely $\deg_L(u)$, and for any fixed u, the expected number of such vertices w is at most $\Delta p = \Delta^{0.15}$. Thus, the expected number of queried vertices in the first two categories is at most $\Delta^{1.15}$.

Finally, let us consider some edge $e = (a, b) \in J$. The number of corresponding queried edges of $L \setminus L_1$ is at most $\deg_{L \setminus L_1}(a) + \deg_{L \setminus L_1}(b)$. Clearly again, for any fixed e we have $\mathbf{E}[\deg_{L \setminus L_1}(a)] \leq \Delta p = \Delta^{0.15}$ and similarly for b. Thus, the expected number of queried entries of $\vec{x}$ corresponding to edge e is at most $4\Delta^{0.15}$.

Putting all these together, the expected number of queried entries of $\vec{x}$ can be bounded by $O(\Delta^{1.15} + t\Delta^{0.15})$. □

Lemma 4.14. *We have $\mathbf{E}[\sum_{v \in V} B(v)] \leq O(n\Delta^{1.15})$ where the expectation is taken over χ, L, π.*

Proof. For any vertex $v \in V$, let us first define $B'(v)$ to the number of edges in L_1 that are queried in the course of running $\mathcal{DO}_\pi(v)$. This can be bounded by:

$$B'(v) \leq \sum_{u \in N_G(v) \cap V_1} \sum_{w : (u,w) \in L_1} A(u, w).$$

Summing over $v \in V$, we get:

$$\sum_v B'(v) \leq \sum_v \sum_{u \in N_G(v) \cap V_1} \sum_{w:(u,w) \in L_1} A(u,w) \leq \sum_{(u,w) \in L_1} A(u,w) \Big(\sum_{v \in N_G(u)} 1 + \sum_{v \in N_G(w)} 1 \Big)$$
$$\leq 2\Delta \sum_{e \in L_1} A(e).$$

Taking expectations and applying Claim 4.12, we therefore have

$$\mathbf{E}\Big[\sum_v B'(v)\Big] \leq 2\Delta\, \mathbf{E}\Big[\sum_{e \in L_1} A(e)\Big] \leq O(\Delta n).$$

By Claim 4.13, we have $\mathbf{E}[B(v) \mid B'(v) = t] \leq O(\Delta^{1.15} + t\Delta^{0.15})$ for any vertex v. This further implies that $\mathbf{E}[B(v)] \leq O(\Delta^{1.15} + \mathbf{E}[B'(v)]\Delta^{0.15})$; thus

$$\mathbf{E}\Big[\sum_v B(v)\Big] \leq O\Big(\Delta^{0.15}\, \mathbf{E}\Big[\sum_v B'(v)\Big] + \Delta^{1.15} n\Big) \leq O(\Delta^{1.15} n). \quad \square$$

We now say that a vertex v is *bad* if $\mathbf{E}_{\vec{x}}[B(v) \mid \pi, L] > \Delta^{1.4}$ (i.e., $\Omega(\Delta^{0.25})$ times larger than the average value given by Lemma 4.14) and *good* otherwise. Let us define $\mathcal{B}$ to be the set of bad vertices. Note that, because $\mathcal{B}$ is based on a conditional expectation, it is determined solely by the random variables π, L.

Claim 4.15. *The expected size of $\mathcal{B}$ satisfies $\mathbf{E}_{\pi,L}[|\mathcal{B}|] \leq O(n/\Delta^{0.25})$.*

Proof. Observe that we have $\sum_{v \in V} \mathbf{E}_{\chi}[B(v) \mid \pi, L] \geq |\mathcal{B}| \cdot \Delta^{1.4}$ with probability one since for each bad vertex $v \in \mathcal{B}$, by definition the expected value of $B(v)$ is at least $\Delta^{1.4}$. Taking expectations over π and L, we therefore get

$$\mathop{\mathbf{E}}_{\pi,L}[|\mathcal{B}|] \leq \Delta^{-1.4} \cdot \mathop{\mathbf{E}}_{\pi,L}\Big[\sum_{v \in V} \mathop{\mathbf{E}}_{\vec{x}}[B(v)] \mid \pi, L\Big] = \Delta^{-1.4} \sum_{v \in V} \mathbf{E}[B(v)].$$

By Lemma 4.14, we have $\sum_{v \in V} \mathbf{E}[B(v)] \leq O(\Delta^{1.15} n)$. Putting these two bounds together gives $\mathbf{E}[|\mathcal{B}|] \leq O(n\Delta^{-0.25})$. $\square$

Claim 4.16. *For any π, L, any good vertex v has $\mathbf{Var}(Z_{v,1} \mid \pi, L) \leq O(\Delta^{1.4})$.*

Proof. By Claim 4.11, for any vertex partitioning $\vec{x}$, we have $\sum_{w \in V}(Z_{v,1}(\vec{x}) - Z_{v,1}(\vec{x}^{(w)}))^2 \leq 4B(v)$, where $\vec{x}^{(w)}$ is obtained by changing the w entry of $\vec{x}$. If we fix π, L and take expectations over $\vec{x}$, this gives

$$\mathop{\mathbf{E}}_{\vec{x}}\Big[\sum_{w \in V}(Z_{v,1}(\vec{x}) - Z_{v,1}(\vec{x}^{(w)}))^2 \mid \pi, L\Big] \leq 4 \mathop{\mathbf{E}}_{\vec{x}}[B(v) \mid \pi, L].$$

On the other hand, by (4.1), any vertex v has

$$\mathbf{Var}(Z_{v,1} \mid \pi, L) \leq \frac{1}{2} \mathop{\mathbf{E}}_{\vec{x}} \Big[\sum_{w \in V} \big(Z_{v,1}(\vec{x}) - Z_{v,1}(\vec{x}^{(w)})\big)^2 \mid \pi, L \Big].$$

Combining the two inequalities gives $\mathbf{Var}(Z_{v,1} \mid \pi, L) \leq 2\,\mathbf{E}[B(v) \mid \pi, L]$. Since v is good with respect to π, L, it satisfies $\mathbf{E}_{\vec{x}}[B(v) \mid \pi, L] \leq O(\Delta^{1.4})$ by definition. Thus

$$\mathbf{Var}(Z_{v,1} \mid \pi, L) = O(\Delta^{1.4}). \quad \square$$

We are now ready to prove the main part of Lemma 4.7.

Proof of Lemma 4.7, part (3). For each vertex $v \in V$, define the random variable y_v to be the indicator function that $\deg^{\text{res}}_M(v) > \Delta^{0.99}$ after running Algorithm 5. We need to show that $\mathbf{E}[\sum_{v \in V} y_v] \leq O(n/\Delta^{0.03})$.

Depending on π and L, let us partition the vertices in V into two subsets $\mathcal{B}$ and $\mathcal{G}$ of respectively bad and good vertices as defined before. Furthermore, fix $\tau = 2\Delta^{0.86}$ and partition the set $\mathcal{G}$ of good vertices into two subsets $\mathcal{H}$ and $\mathcal{L}$ where for any vertex $v \in \mathcal{H}$, $\mathbf{E}_{\vec{x}}[Z_{v,1} \mid \pi, L] \geq \tau$ and for any $v \in \mathcal{L}$, $\mathbf{E}_{\vec{x}}[Z_{v,1} \mid \pi, L] < \tau$. We have:

$$\sum_{v \in V} y_v = \sum_{v \in \mathcal{B}} y_v + \sum_{v \in \mathcal{L}} y_v + \sum_{v \in \mathcal{H}} y_v.$$

By Claim 4.15, we know directly that $\mathbf{E}[|\mathcal{B}|] \leq O(n/\Delta^{0.25})$. Since $y_v \leq 1$ for any vertex v, we have $\mathbf{E}[\sum_{v \in \mathcal{B}} y_v] \leq \mathbf{E}[|\mathcal{B}|] \leq O(n/\Delta^{0.25})$.

Now, for any fixed $v \in V$, we compute the probability of the event that $v \in \mathcal{L}$ and $y_v = 1$ (respectively, $v \in \mathcal{H}$ and $y_v = 1$); we show that each event has probability $O(\Delta^{-0.03})$.

Good vertices of type $\mathcal{L}$. Recall that $\deg^{\text{res}}_M(v) \leq Z_{v,1} + \ldots + Z_{v,k}$ where $k = \Delta^{0.1}$ denotes the number of partitions. Taking expectations we get

$$\mathbf{E}[\deg^{\text{res}}_M(v) \mid \pi, L] \leq \mathbf{E}[Z_{v,1} + \ldots + Z_{v,k} \mid \pi, L] = k\,\mathbf{E}[Z_{v,1} \mid \pi, L]$$

where the latter equality for symmetry of the partitions. If $v \in \mathcal{L}$, then $\mathbf{E}[Z_{v,1} \mid \pi, L] < \tau$, thus, $\mathbf{E}[\deg^{\text{res}}_M(v) \mid \pi, L] \leq k\tau = \Delta^{0.1} \cdot 2\Delta^{0.86} = 2\Delta^{0.96}$. By Markov's inequality, $\mathbf{Pr}[\deg^{\text{res}}_M(v) > \Delta^{0.99} \mid \pi, L] < O(\Delta^{-0.03})$. Therefore, $\mathbf{Pr}[y_v = 1 \wedge v \in \mathcal{L} \mid \pi, L] \leq O(\Delta^{-0.03})$. Integrating over π, L also $\mathbf{Pr}[y_v = 1 \wedge v \in \mathcal{L}] \leq O(\Delta^{-0.03})$ as desired.

Good vertices of type $\mathcal{H}$. We show that good vertices of type $\mathcal{H}$ are highly likely to be matched in their own partition and thus not too many of them will remain in the graph. For such a vertex v, one of the following two events must occur: either $Z'_v \geq \Delta^{0.86}$ or $Z'_v < \Delta^{0.86}$. The first of these events has probability $\exp(-\operatorname{poly}(\Delta)) \ll \Delta^{-0.03}$ by Claim 4.9. We next need to bound the probability of having $v \in \mathcal{H}$ and also having $Z'_v \leq \Delta^{0.86}$. If this occurs, by definition of Z'_v, we have at least one index $j \in [k]$ with $Z_{v,j} < \Delta^{0.86}$. We bound the occurrence probability of this event.

Since $v \in \mathcal{H}$, by definition it is a good vertex and thus Claim 4.16 shows that $\mathbf{Var}(Z_{v,i} \mid \pi, L) \leq O(\Delta^{1.4})$. Also, $\mathbf{E}[Z_{v,i} \mid \pi, L] \geq 2\Delta^{0.86}$. Therefore, by Chebyshev's inequality, for any fixed $i \in [k]$,

$$\begin{aligned}\mathbf{Pr}\left[Z_{v,i} < \Delta^{0.86} \mid \pi, L\right] &\leq \mathbf{Pr}\left[\left|Z_{v,i} - \mathbf{E}[Z_{v,i} \mid \pi, L]\right| \geq 2\Delta^{0.86} - \Delta^{0.86}\right] \\ &\leq O\left(\frac{\mathbf{Var}(Z_{v,i} \mid \pi, L)}{(\Delta^{0.86})^2}\right) \leq O\left(\frac{\Delta^{1.4}}{\Delta^{1.72}}\right) \leq O(\Delta^{-0.32}).\end{aligned}$$

By a union bound over the $k = \Delta^{0.1}$ choices of j, we have

$$\mathbf{Pr}\left[v \in \mathcal{H} \text{ and there exists some } j \in [k] \text{ with } Z_{v,j} \leq \Delta^{0.86} \mid \pi, L\right] \leq O(\Delta^{-0.22}).$$

This means that the probability that $y_v = 1$ and $v \in \mathcal{H}$ is $O(\Delta^{-0.22}) \ll O(\Delta^{-0.03})$. □

4.4 Putting Everything Together

We now prove Lemma 4.6, showing that Algorithm 5 can be used to reduce the overall graph degree. There are two parts to doing this. First, we need to amplify the success probability of Lemma 4.7, which only showed a degree reduction in expectation, into one holding w.e.h.p. Next, we need to remove the remaining high-degree vertices.

Claim 4.17. *There is an algorithm to generate a matching M which w.e.h.p. uses $n/\Delta^{\Omega(1)}$ space per machine and $O(m)$ total space, such that there are at most $n/\Delta^{0.02}$ vertices v with $\deg^{res}_M(v) > \Delta^{0.99}$.*

Proof. We may assume that the original graph has as least $n/\Delta^{0.02}$ vertices with $\deg(v) > \Delta^{0.99}$, as otherwise there is nothing to do. This implies that $m \geq n\Delta^{0.97}$.

Now consider running Algorithm 5 to generate a matching M. Let us define Y to be the number of vertices $v \in V$ with $\deg^{\text{res}}_M(v) > \Delta^{0.99}$. Line 5 has shown that $\mathbf{E}[Y] \leq O(n/\Delta^{0.03})$, and so we need to show concentration for Y. There are two cases depending on Δ.

Case 1: $\Delta > n^{0.1}$**.** In this case, Markov's inequality applied to Y shows that $\mathbf{Pr}[Y > n\Delta^{-0.02}] \leq O(\Delta^{-0.01}) \leq 1/2$. Now consider running $t = n^a$ parallel iterations of Algorithm 5 for some constant $a > 0$, generating matchings $M_1, \dots, M_t$. Since they are independent, there is a probability of at least $1 - 2^{-t}$ that at least one matching M_i has the property that its residual set of high-degree vertices satisfies $Y > n\Delta^{-0.02}$. Thus, w.e.h.p., this algorithm satisfies the condition on the high-degree vertices. Each application of Algorithm 5 separately uses $O(n) + m/\operatorname{poly}(\Delta)$ space. Therefore, the t iterations in total use $O(n^{1+a}) + n^a m/\operatorname{poly}(\Delta)$ space. Since $\Delta > n^{0.1}$ and $m \geq \Delta n^{0.97} > n^{1.07}$, this is $O(m)$ for a a sufficiently small constant.

Case 2: $\Delta < n^{0.1}$. We can regard Y as being determined by $O(n\Delta)$ random variables, namely, the values ρ, χ, L. By Lemma 3.5, modifying each entry of ρ, χ, or L can only change the match-status of at most $O(1)$ vertices. Each of these, in turn, has only Δ neighbors, which are the only vertices whose degree in $G[V \setminus M]$ is changed. Thus, changing each of the underlying random variables can only change Y by $O(\Delta)$. By Corollary 2.4, therefore, w.e.h.p. we have

$$Y \leq \mathbf{E}[Y] + O(\Delta) n^{0.01} \sqrt{n\Delta} \leq O(n\Delta^{-0.03}) + O(n^{0.51}\Delta^{1.5}).$$

As $\Delta \leq n^{0.1}$ and Δ is larger than any needed constants, this is at most $n\Delta^{-0.02}$. Therefore, already a single application of Algorithm 5 suceeds w.e.h.p. □

Having slightly reduced the number of high-degree vertices, we next use the following Algorithm 7, which significantly decreases the number of high-degree vertices.

Algorithm 7:

```
1 Let $\mathcal{Y}$ be the set of vertices in $G[V \setminus M]$ with degree greater than $\tau = \Delta^{0.999}$.
2 Sample each edge with at least one end-point in $\mathcal{Y}$ with probability $q := \Delta^{-0.99}$ and let
    $L$ be the set of sampled edges.
3 Put $G^L = (V, L)$ in machine 1, choose an arbitrary permutation $\pi$ over its edges and
    return matching $M' := \mathsf{GMM}(G^L, \pi)$.
```

Claim 4.18. *Given a graph G, suppose we apply Claim 4.17; let M be the resulting matching and $G' = G[V \setminus M]$. Suppose we next run Algorithm 7 on G' and let M' denote the resulting matching. Let $\mathcal{Y}'$ denote the set of vertices v with $\deg^{res}_{M \cup M'}(v) > \tau$. Then, w.e.h.p., $|\mathcal{Y}'| \leq n/\Delta^{1.01}$.*

Proof. Let $\mathcal{Y}$ be the set of vertices with $\deg^{\text{res}}_M(v) > \tau$ and $Y = |\mathcal{Y}|$. By Claim 4.17, w.e.h.p. $Y \leq n/\Delta^{0.02}$. For the remainder of this proof, we assume that M (and hence Y) is fixed and it satisfies this bound.

We first analyze $\mathbf{E}[Y']$ where we define $Y' = |\mathcal{Y}'|$. Consider some vertex $v \in \mathcal{Y}$. By the sparsification property of RGMM, with probability at least $1 - \beta$ the vertex v has

$\deg^{\text{res}}_{M\cup M'} \leq O(\frac{\log 1/\beta}{q})$. Setting $\beta = e^{-\Delta^{0.001}}$, we get that $\deg^{\text{res}}_{M\cup M'}(v) \leq O(\Delta^{0.991})$ with probability $1-\exp(-\Delta^{\Omega(1)})$. Since this holds for any vertex $v \in \mathcal{Y}$, we have shown that

$$E[Y'] \leq Y \cdot \exp(-\Delta^{\Omega(1)}) \leq ne^{-\Delta^{\Omega(1)}}.$$

We next need to show concentration for Y'. For this, note that if $\Delta > n^{0.01}$, then the above bound on $\mathbf{E}[Y']$ already implies (by Markov's inequality) that $Y' < 1$ w.e.h.p.

If $\Delta < n^{0.01}$, then we use the bounded differences inequality. Here, Y' can be regarded as a function of $n\Delta$ random variables (the membership of each edge in L). By Lemma 3.5, each such edge can affect the match-status of $O(1)$ vertices. Each such vertex w, in turn, can only change the membership in $\mathcal{Y}'$ of its neighbors. Hence, each random variable changes Y' by at most $O(\Delta)$. By Corollary 2.4, we therefore have w.h.p.

$$Y' \leq \mathbf{E}[Y'] + O(\Delta \times \sqrt{n\Delta} \times n^{0.01}) \leq n\exp(-\Delta^{\Omega(1)}) + O(\Delta^{1.5}n^{0.51}).$$

By our assumption that $\Delta \leq n^{0.01}$, this is easily seen to be smaller than $n/\Delta^{1.01}$. □

Proof of Lemma 4.6. When we apply Claim 4.17 and then apply Claim 4.18, this w.e.h.p. gives matchings M, M' respectively such that $G[V \setminus (M \cup M')]$ has at most $n/\Delta^{1.01}$ vertices of degree larger than $\Delta^{0.999}$. Claim 4.17 already obeys the stated space bounds. For Algorithm 7, observe that $|\mathcal{Y}| \leq n/\Delta^{0.02}$, and so there are at most $n\Delta^{0.98}$ edges incident to $\mathcal{Y}$. This means $\mathbf{E}[|L|] \leq n/\Delta^{0.01}$ and a simple Chernoff bound thus shows that $L \leq n/\Delta^{\Omega(1)}$ w.e.h.p. Finally, we place all vertices with degree at least $\Delta^{0.999}$ and their incident edges onto a single machine; this clearly takes $O(n/\Delta^{0.01})$ space. Since Δ is larger than any needed constant, this is at most $n/\Delta^{\Omega(1)}$. We thus expand $M \cup M'$ to a maximal matching M'' of $G[V \setminus (M \cup M')]$. At the end of this process, all remaining vertices of G must have degree less than $\Delta^{0.999}$. □

4.5 Maximal Matching for Bounded Arboricity Graphs

In this section, we overview the key ideas behind the proof of Theorem 4.2.

The insufficiency of space to store all the vertices in one machine in the strictly sublinear regime of MPC imposes challenges similar to those faced by algorithms in the LOCAL [140] model: There is one processor on each of the nodes of the input graph and two processors can communicate in each round if and only if there is an edge between their corresponding vertices. The fact that the vertices, in the strictly sublinear regime of MPC, have to make decisions (such as joining the MIS) based solely on a *small* neighborhood that they observe around them, makes the algorithmic challenges of the two models similar. We need to keep in mind, however, that the constraints that impose such locality in the two models are fundamentally different. Roughly, in LOCAL, the radius that a vertex sees around is small but in MPC, it is the size of this subgraph that is restricted to be sublinear.

However, a key difference between the two models that makes us hope for faster MPC algorithms is the possibility of all-to-all communications. To illustrate this over a simple example, consider a directed path $(v_1, v_2, \ldots, v_d)$. It is not hard to see that in the LOCAL model, it takes at least $d-1$ rounds for v_1 to send one bit of message to v_d. However, thanks to all-to-all communications, it can be done in only $O(\log d)$ rounds of MPC using the well-known *pointer jumping* technique: Initially, for any $i < d$, set $p(v_i) := v_{i+1}$ and in each round update it to be $p(v_i) \leftarrow p(p(v_i))$. In only $O(\log d)$ rounds $p(v_1)$ will point to v_d. This is possible since vertex v_i can directly communicate with vertex $u = p(v_i)$ and ask for the value of $p(u)$. Achieving such exponential improvements, however, is typically much more intricate for other problems due to the space restrictions of MPC.

To further demonstrate the relevance of the above *graph exponentiation* idea to our problems, we recall a beautiful (and well-known) property of LOCAL algorithms. In any r-round LOCAL algorithm, the final state of each node/edge is merely a function of its r-hop (i.e., the nodes/edges that are at distance at most r). This has been extensively used in the literature to prove lower bounds, but has also given rise to a few algorithmic ideas (see e.g., [138, 7] and the follow-up work or [91]). Combined with the $O(\log n)$ round LOCAL algorithm of Luby [126] for MIS, or that of Israeli and Itai [104] for maximal matching, this property implies that if in MPC, we manage to collect the $O(\log n)$-hop of each vertex in a machine responsible for it, we can locally simulate these algorithms without any further communications.[2] Using the graph exponentiation idea, we hope to be able to do this in much faster than $O(\log n)$ rounds. There are however two fundamental barriers for this:

Local memory barrier. The $O(\log n)$-hop of a vertex may be as large as $\Omega(m)$, exceeding the local space of a machine. Even the 1-hop of a vertex with degree higher than $\omega(n^\delta)$ cannot be stored in one machine.

Global memory barrier. Storing the neighborhood of each vertex on its corresponding machine leads to multiple copies of each vertex and thus a total aggregated memory of

[2]We note that since these algorithms are randomized, one also needs to collect the tape of random bits of each vertex as well so that the results computed on different machines are compatible.

significantly larger than the input size, m.

Let us forget the global memory barrier for now and focus on handling the local memory problem. We can safely assume for $t = \delta \log_\Delta n$, that the t-hop of every vertex fits the memory of one machine since $\Delta^{\delta \log_\Delta n} = n^\delta$. This implies that we can indeed simulate t rounds of a LOCAL algorithm in one round if we first collect the t-hops (which can be done in $O(\log t)$ rounds). However, t is usually smaller than the actual number of rounds that the algorithm takes. A way to overcome this is to share the *states*. That is, having the state of each vertex by the end of round t, we can share these states with all other machines in one round of communication and simulate the next t rounds of the algorithm to obtain the states by round $2t$. We can repeat this to simulate r rounds of our LOCAL algorithm in $O(r/t + \log t)$ rounds. Note that for this idea to work, the states of the LOCAL algorithm have to be crucially small so that they can be shared and stored on the machines. However, even incorporating this simulation does not help when the maximum degree is large. For instance, when $\Delta = \Omega(n^\delta)$, even the direct neighbors of a vertex may not fit the memory of a single machine.

To convey the main intuitions, let us assume that the arboricity of the input graph is $O(1)$. We borrow a subroutine first introduced by Barenboim, Elkin, Pettie, and Schneider [20, Theorem 7.2] for the LOCAL model and use it in our simulation. This algorithm, with slight modifications, guarantees that for any $\tau \geq \log^{O(1)} n$ (that is also sufficiently larger than arboricity,) one can reduce the maximum degree to τ in $O(\log_\tau n)$ rounds by committing a subset of edges (or vertices) to the maximal matching (or MIS).[3] Call a vertex v *high-degree* if $\deg(v) > \tau$ and *low-degree* otherwise. This round complexity is achieved since the algorithm removes $\tau^{\Omega(1)}$ fraction of high-degree vertices in each round by matching them to their low-degree neighbors (or by adding their low-degree neighbors to MIS). The algorithm turns out to be very simple to implement and intuitive. For instance for maximal matching, in each round, after discarding a subset of edges, each low-degree vertex proposes to one of its high-degree neighbors uniformly at random and then each high-degree vertex gets matched to one of its proposing neighbors (if any) arbitrarily. The intuition behind the analysis of this subroutine is roughly as follows: Fix a high-degree vertex v and suppose it is likely to survive ℓ rounds and remain high-degree. For this to happen, not only almost all neighbors of v have to be high-degree, but the neighbors of its neighbors should also be high-degree and this should continue for roughly ℓ levels. Due to the small arboricity of the graph, these high-degree vertices cannot be highly inter-connected (otherwise we have a dense subgraph)

[3]Since we assume that arboricity is constant in this section, we have hidden the actual dependence of the running time on the arboricity.

and thus each level requires $\tau^{\Omega(1)}$ additional nodes. Therefore, ℓ cannot exceed $O(\log_\tau n)$.

Now we can use this subroutine as follows in the MPC model. Let Δ be the current maximum degree in the graph and let $\tau = \sqrt{\Delta}$. The algorithm of [20] then takes $O(\log_{\sqrt{\Delta}} n) = O(\log_\Delta n)$ rounds to reduce the maximum degree polynomially to $\sqrt{\Delta}$ in the LOCAL model. Simulating it with the graph exponentiation technique discussed above, this can be done in $O(\log\log n)$ rounds of MPC. Since each of these simulations reduces the current maximum degree polynomially, repeating it for $O(\log\log n)$ phases can be shown to reduce the maximum degree to a constant where the problem becomes trivial to solve using known algorithms. Hence, the overall round complexity would be $O(\log^2 \log n)$. In case the graph has a super-constant arboricity, this idea can be used to reduce the maximum degree in $O(\log^2 \log n)$ rounds to $\text{poly}(\lambda)$ and at that point we can switch to the algorithm of [92] to solve the remaining low degree graph in $\widetilde{O}(\sqrt{\log \lambda})$ rounds.

Optimizing the global memory. The challenge in optimizing the global memory is due to the simulation step. Recall that in the simulation algorithm, we allocate $O(n^\delta)$ space to each vertex. Hence, we inevitably need $\Omega(m + n^{1+\delta})$ total space instead of $O(m)$. It does not seem possible to give a general simulation algorithm that avoids the $n^{1+\delta}$ additive term in the total space. The algorithm of [20], however, can be simulated more efficiently since roughly speaking the algorithm "ignores" low-degree vertices in a large number of rounds and so we do not need to collect their neighbors.

See [37] and [35] for more details about this simulation technique and the bound of Theorem 4.2.

Chapter 5

Massively Parallel Graph Connectivity

Identifying the connected components of a graph is a fundamental problem that has been studied in a variety of settings (see e.g. [2, 88, 74, 146, 151, 142] and the references therein). This problem is also of great practical importance [145] with a wide range of applications, e.g. in clustering [142]. In this chapter, we study this problem in the Massively Parallel Computations (MPC) model.

As before, we use S and M to respectively denote the space per machine and the number of machines. We also that we use n and m to respectively denote the number of vertices and edges in the input graph $G = (V, E)$. Our focus in this chapter is particularly on the strictly sublinear regime of MPC. Namely, we assume that the space per machine is $S = n^{\delta}$ for some constant $\delta \in (0, 1)$ that can be made arbitrarily small. Furthermore, we assume that $M \cdot S = O(m)$ so that there is only enough total space to store the input. This only leaves one parameter to optimize: the number of rounds.

Like many other graph problems, $O(\log n)$-round MPC algorithms for graph connectivity have been known for a long time. On the negative side, a widely believed conjecture [153, 143, 118, 18] implies that this is the best possible in the strictly sublinear MPC:

Conjecture 5.1 (1v2-Cycle Conjecture)**.** *Distinguishing whether the input is a cycle on n vertices or two cycles on $n/2$ vertices with $n^{1-\Omega(1)}$ space per machine and* $\text{poly}(n)$ *total space requires $\Omega(\log n)$ rounds.*

The 1v2-Cycle conjecture and the matching upper bound, however, are far from explaining the true complexity of the problem. First, the hard example used in the conjecture is very different from what most graphs look like. Second, the empirical performance of the existing algorithms (in terms of the number of rounds) is much lower than what the upper bound of $O(\log n)$ suggests [112, 118, 149, 142, 127]. This disconnect between theory and practice has motivated the study of graph connectivity as a function of diameter D of the graph. The reason is that the vast majority of real-world graphs, indeed have very low diameter [121, 73]. This is reflected in multiple theoretical models designed to capture real-world graphs, which yield graphs with polylogarithmic diameter [61, 99, 125, 62].

Our main contribution in this chapter is the following algorithm:

Theorem 5.2 (main result). *There is a strongly sublinear* MPC *algorithm with $O(m)$ total space that given a graph with diameter D, identifies its connected components in $O(\log D + \log\log_{m/n} n)$ rounds. The algorithm is randomized, succeeds with high probability, and does not require prior knowledge of D.*

Note that for the wide range of values $D = \log^{\Omega(1)} n$, the algorithm of Theorem 5.2 takes $O(\log D)$ rounds which can be shown to be optimal under Conjecture 5.1. Moreover, when D is not in this range the algorithm takes only $O(\log\log n)$ rounds.

Related Work

Theorem 5.2 improves over a previous algorithm of Andoni, Song, Stein, Wang, and Zhong [9] that takes $O(\log D \cdot \log\log_{m/n} n)$ rounds. Note that the algorithm of [9] matches the $\Omega(\log D)$ lower bound for a very specific case: if the graph is extremely dense, i.e., $m = n^{1+\Omega(1)}$. In practice, this is usually not the case [71, 121, 86]. In fact, it is worth noting that the main motivation behind the MPC model with sublinear in n space per machine is the case of sparse graphs [109]. We also note that for the particularly important case when $D = \operatorname{poly}\log n$, the algorithm of Theorem 5.2 requires only $O(\log\log n)$ rounds which quadratically improves upon a bound of $O(\log^2 \log n)$ rounds, which follows from the result of [9]. Theorem 5.2 also provides a number of other qualitative advantages. For instance it succeeds with high probability as opposed to the constant success probability of [9]. Furthermore, the running time required for identifying each connected component depends on its own diameter only. The diameter D in the result of [9] is crucially the largest diameter.

We note that another algorithm by Assadi, Sun, and Weinstein [18] implies an $O(\log \frac{1}{\lambda} + \log\log n)$ round algorithm for graphs with $\widetilde{O}(n)$ edges that have *spectral gap* at least λ. By a well-known bound, $D = O(\frac{\log n}{\lambda})$. Therefore, our algorithm requires $O(\log D + \log\log n) = O(\log(\frac{\log n}{\lambda}) + \log\log n) = O(\log \frac{1}{\lambda} + \log\log n)$ rounds for graphs with spectral gap at least λ. As a result, the running time bound of our algorithm is never worse than the bound due to Assadi *et al.* However, as shown in [9], there are graphs with $\frac{1}{\lambda} \geq D \cdot n^{\Omega(1)}$ making our algorithm more general.

Recent Developments

After the first publication of the result of this chapter in [41], Liu, Tarjan, and Zhong [123] showed that the same algorithm can also be efficiently implemented in the PRAM model, leading to an $O(\log D + \log\log n)$-round algorithm for graph connectivity. Moreover, Coy and Czumaj [72] showed that the algorithm can be efficiently derandomized, essentially achieving all the guarantees of Theorem 5.2 deterministically.

5.1 High-Level Overview of Techniques

Recall that we assume the regime of MPC with strictly sublinear space of n^δ with δ being a constant in $(0, 1)$. This local space, roughly speaking, is usually not sufficient for computing any meaningful global property of the graph within a machine. As such, most algorithms in this regime proceed by performing *local operations* such as contracting edges/vertices, adding edges, etc. Note that even the direct neighbors of a high-degree vertex may not fit in the memory of a single machine, however, using standard techniques most basic local operations can be implemented in $O(1/\delta) = O(1)$ rounds of MPC. Intuitively, if the degree of a vertex v is larger than n^δ, one can construct a "virtual" n^δ-ary regular tree of depth $O(1/\delta)$ out of v whose leaves correspond to the original neighbors of v. This way, the (virtual) degree of every vertex is bounded by $O(n^\delta)$ and fits the memory. Additionally the distance of each vertex to its original neighbors is at most $O(1/\delta)$. For the purpose of this section, however, we do not get into such technicalities and discuss the high level intuition behind the algorithm. We start with a brief overview of some of the relevant techniques and results, then proceed to describe the new ingredients of our algorithm and its analysis.

Graph exponentiation. Consider a simple algorithm (a similar variant of which was also discussed in Section 4.5) that connects every vertex to vertices within its 2-hop (i.e., vertices of distance 2) by adding edges. It is not hard to see that the distance between any two vertices shrinks by a factor of 2. By repeating this procedure, each connected component becomes a clique within $O(\log D)$ steps. The problem with this approach, however, is that the total space required can be up to $\Omega(n^2)$, which for sparse graphs is much larger than $O(m)$. Andoni, Song, Stein, Wang, and Zhong [9] managed to improve the total space to the optimal bound of $O(m)$, at the cost of increasing the round complexity to $O(\log D \cdot \log\log_{m/n} n)$. We briefly overview this result below.

Overview of [9]. Suppose that *every* vertex has degree at least $d \gg \log n$. Select each vertex as a *leader* independently with probability $\Theta(\log n/d)$. Then contract every non-leader vertex to a leader neighbor (which w.h.p. exists). This shrinks the number of vertices from n to $\widetilde{O}(n/d)$. As a result, the amount of space available per remaining vertex increases to $\widetilde{\Omega}(\frac{m}{n/d}) = \widetilde{\Omega}(\frac{nd}{n/d}) \approx d^2$. At this point, a variant of the aforementioned graph exponentiation technique can be used to increase vertex degrees to d^2 (but not more), which implies that another application of leader contraction decreases the number of vertices by a factor of $\Omega(d^2)$. Since the available space per remaining vertex increases doubly exponentially, $O(\log\log n)$ phases of leader contraction suffice to increase it to n. Moreover, each phase requires $O(\log D)$ iterations of graph exponentiation, thus the round complexity is $O(\log D \log\log n)$.

5.1.1 Our Connectivity Algorithm: The Roadmap

The main shortcoming of Andoni *et al.*'s algorithm is that within a phase, where the goal is to increase the degree of every vertex to d, those vertices that have already reached degree d are *stalled* (i.e., do not connect to their 2-hops) until all other vertices reach this degree. Because of the stalled vertices, the only guaranteed outcome of the graph exponentiation operation is increasing vertex degrees. In particular, the diameter of the graph may remain unchanged. This is precisely why their algorithm may require up to $O(\log D \cdot \log\log n)$ applications of graph exponentiation. We note that this is not a shortcoming of their analysis. Indeed, we remark that there are family of graphs on which Andoni *et al.*'s algorithm takes $\Theta(\log D \cdot \log\log n)$ rounds.

Instead of describing our algorithm, we focus in this section on some of the properties that we expect it to satisfy, and how they suffice to get our desired round complexity. This overview should be helpful when reading the description of the algorithm in Section 5.2.1.

Our algorithm assigns *budgets* to vertices. Intuitively, a budget controls how much space a vertex can use, i.e., how much it can increase its degree. To bound the space complexity, we will bound the sum of all vertex budgets. In our algorithm vertices may have different budgets (differently from the algorithm of Andoni *et al.*). This allows us to prevent the vertices from getting stalled behind each other. Overall, we have $L = \Theta(\log\log n)$ possible budgets $\beta_0, \beta_1, \ldots, \beta_L$ where $\beta_0 = O(1)$, $\beta_L = \Omega(n)$, and $\beta_i = (\beta_{i-1})^{1.25}$. We say a vertex v is at *level* $\ell(v) = i$, if its budget $b(v)$ equals β_i. The algorithm executes a single loop until each connected component becomes a clique. We call a single execution of this loop an *iteration* which can be implemented in $O(1)$ rounds.

Property 1 (see Lemma 5.9)**.** For any two vertices u and v at distance exactly 2 at the beginning of an iteration of the algorithm, after the next 4 iterations, either their distance decreases to 1, or the level of both vertices increases by at least one.

We call every 4 iterations of the algorithm a *super-iteration*. Property 1 guarantees that if a vertex does not get connected to every vertex in its 2-hop within a super-iteration, its level must increase.[1] Recall, on the other hand, that the maximum level of any vertex is at most $O(\log\log n)$. As such, every vertex resists getting connected to those in its 2-hop for at most $O(\log\log n)$ super-iterations. However, somewhat counter-intuitively, this observation is (provably) not sufficient to guarantee an upper bound of $O(\log D + \log\log n)$ rounds. Our main tool in resolving this, is maintaining another property.

Property 2 (see Observation 5.7)**.** If a vertex v is neighbor of a vertex u with $\ell(u) > \ell(v)$, then by the end of the next iteration, the level of v becomes at least $\ell(u)$.

The precise proof of sufficiency of Properties 1 and 2 is out of the scope of this section. Nonetheless, we provide a proof sketch with the hope to make the actual arguments easier to understand. See Lemma 5.12 for the formal statement and its proof.

Proof sketch of round complexity. Fix two vertices u and v in one connected component of the original graph and let P_1 be the shortest path between them. As the vertices connect to their 2-hops and get closer to each other, we drop some of the vertices of P_1 while ensuring that the result is also a valid path from u to v. That is, we maintain a path P_i by the end of each super-iteration i which is obtained by replacing some subpaths of P_{i-1} of length at least two by single edges. We say that the interior vertices of the replaced subpaths are dropped.

Our goal is to show that for $R := O(\log D + \log\log n)$, path P_R has $O(1)$ length, thus dropping the diameter of the whole graph to $O(1)$ which is trivially solved in $O(1)$ rounds by our algorithm. To show this, we require a potential-based argument. Suppose that we initially put one coin on every vertex of P_1, thus we have at most $D+1$ coins. Let $P_i = (u_1, \dots, u_j, \dots, u_k)$ be the path at the end of super-round i. As we construct P_{i+1} from P_i, any vertex u_j that is dropped from P_i, passes its coins evenly to vertices in $\{u_{j-2}, u_{j-1}, u_{j+1}, u_{j+2}\}$ that exist and survive to P_{i+1} (if none of them survive the coins are discarded). Moreover, we construct P_{i+1} from P_i such that it satisfies the following property,

[1]In this section, for simplicity and to convey the main intuitions, we ignore the changes to the graph's structure caused by vertices being merged together.

which we call invariant 1: If the level of a vertex $u_j \in P_i$ within super-iteration $i+1$ does not increase, there is a vertex in $\{u_{j-2}, u_{j-1}, u_j, u_{j+1}, u_{j+2}\}$ that is dropped. This is guaranteed to be possible due to Property 1: Observe that at least one of u_{j-2} and u_{j+2} should belong to P_i otherwise the path has length ≤ 2 and is already small. Let us suppose w.l.o.g. that u_{j-2} exists. Now if either of u_{j-2} or u_j is dropped from P_i the condition is satisfied, otherwise by Property 1, u_{j-2} and u_j should be directly connected after super-iteration $i+1$ and and it is thus safe to drop u_{j-1} and ensure P_{i+1} remains to be a path.

Finally, we use Property 2 to prove invariant 2: In any path P_i, every vertex of level j has at least $(1.25)^{i-j}$ coins. That is, we have *more* coins on the vertices that have *lower* levels. Note that this is sufficient to prove the round complexity. For, otherwise, if $|P_R| > 2$, due to the fact that the level of every vertex is at most $O(\log\log n)$, there should remain a vertex in P_R with at least

$$(1.25)^{R-j} \geq (1.25)^{\Theta(\log D + \log\log n) - O(\log\log n)} \geq (1.25)^{4\log D} \gg D+1$$

coins, while we had only $D+1$ coins in total. Property 2 is useful in the proof of invariant 2 in the following sense: If a low-level vertex $w \in P_i$ survives to P_{i+1} without increasing its level, its dropped 2-hop neighbor (which exists by invariant 1) cannot have a higher level than w by Property 2 (since their distance is at most two and a super-iteration includes four iterations), and thus passes enough coins to w. $\square$

5.2 Main Algorithm: Connectivity with $O(m) + \widetilde{O}(n)$ Total Space

In this section, we describe an $O(\log D + \log\log_{T/n} n)$ round connectivity algorithm assuming that the total available space is $T \geq m + n\log^{\alpha} n$ where α is some desirably large constant. We later show how to improve the total space to the optimal bound of $O(m)$ in Section 5.3. We start with description of the algorithm in Section 5.2.1 and proceed to its analysis in Sections 5.2.2 to 5.2.4.

5.2.1 The Algorithm

The algorithm consists of a number of *iterations*, each of which calls three subroutines named Connect2Hops, RelabelInterLevel, and RelabelIntraLevel.[2] Each iteration will be implemented in $O(1)$ rounds of MPC and we later show that $O(\log D + \log\log_{T/n} n)$ iterations are sufficient.

[2]The relabeling subroutines are close to the leader contraction operation we discussed in Section 5.1. However, we use a different terminology to emphasize the difference in handling *chains*. See Figure 5.1.

We first formalize the overall structure of the algorithm as Algorithm 1, then continue to describe the subroutines of each iteration one by one.

Algorithm 1. FindConnectedComponents$(G(V, E))$

```
1. To any vertex $v$, we assign a *level* $\ell(v) \leftarrow 0$, a *budget* $b(v) \leftarrow (\frac{T}{n})^{1/2}$, and a set $C(v) \leftarrow \{v\}$ which throughout the algorithm, denotes the set of vertices that $v$ *corresponds to*. Moreover, every vertex $v$ is initially marked as *active* and we set $\text{next}(v) \leftarrow v$.
2. Repeat the following steps until each remaining connected components becomes a clique:
    (a) Connect2Hops$(G, b, \ell)$
    (b) RelabelInterLevel$(G, b, \ell, C, \text{next})$
    (c) RelabelIntraLevel$(G, b, \ell, C)$
3. For every remaining connected component $\mathcal{C}$ corresponds to one of the connected components of the original graph whose vertex set is $\cup_{v \in \mathcal{C}} C(v)$.
```

Connect2Hops(G, b, ℓ)

```
For every active vertex $v$:
    1. Define $H(v) := \{u \mid \exists w \text{ s.t. } w \in N(u) \cap N(v), \ell(u) = \ell(w) = \ell(v)\}$.
    2. Let $d_v$ be the number of vertices currently connected to $v$ that have level at least $\ell(v)$. Pick $\min\{b(v) - d_v, |H(v)|\}$ arbitrary vertices in $H(v)$ and connect them to $v$.
```

RelabelInterLevel$(G, b, \ell, C, \text{next})$

```
1. For every (active or inactive) vertex $v$:
    (a) Let $h(v)$ be the neighbor of $v$ with the highest level with ties broken arbitrarily.
    (b) If $\ell(h(v)) > \ell(v)$, mark $v$ as *inactive* and set $\text{next}(v) \leftarrow h(v)$.
2. Replace every edge $\{u, v\}$ in the graph with $\{\text{next}(u), \text{next}(v)\}$.
3. Remove duplicate edges and self-loops.
4. For every vertex $v$, set $I(v) \leftarrow \cup_{u:\text{next}(u)=v} C(u)$.
5. For every vertex $v$, if $v$ is active, set $C(v) \leftarrow C(v) \cup I(v)$ and if $v$ is inactive set $C(v) \leftarrow I(v)$.
6. If an inactive vertex has become isolated, remove it from the graph.
```

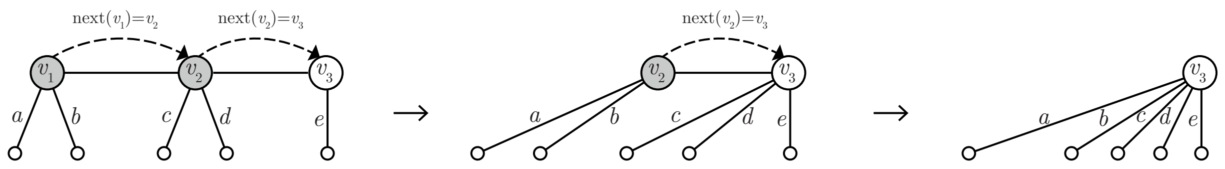


Figure 5.1: Algorithm 1 does not traverse "relabeling chains". In the first iteration, vertex v_1 is relabeled to v_2 and v_2 is relabeled to v_3. After two iterations, both v_1 and v_2 are contracted to v_3. Note that the edge $\{v_2, v_3\}$ of iteration 1 will become a self-loop after applying the relabeling $v_2 \to v_3$ and thus will be removed. However, the edge $\{v_2, v_3\}$ that still remains in the second iteration is the result of applying relabelings $v_1 \to v_2$ and $v_2 \to v_3$ on edge $\{v_1, v_2\}$.

```
RelabelIntraLevel$(G, b, \ell, C)$
1. Mark an active vertex $v$ as "saturated" if it has more than $b(v)$ active neighbors that
   have the same level as $v$.
2. If an active vertex $v$ has a neighbor $u$ with $\ell(u) = \ell(v)$ that is marked as saturated, $v$ is
   also marked as saturated.
3. Mark every saturated vertex $v$ as a "leader" independently with probability
   $\min\{\frac{3\log n}{b(v)}, 1\}$.
4. For every leader vertex $v$, set $\ell(v) \leftarrow \ell(v) + 1$ and $b(v) \leftarrow b(v)^{1.25}$.
5. Every non-leader saturated vertex $v$ that sees a leader vertex $u$ of the same level (i.e.,
   $\ell(u) = \ell(v)$) in its 2-hop (i.e., $\text{dist}(v, u) \le 2$), chooses one as its leader arbitrarily.
6. Every vertex is contracted to its leader. That is, for any vertex $v$ with leader $u$, every
   edge $\{v, w\}$ will be replaced with an edge $\{u, w\}$ and all vertices in set $C(v)$ will be
   removed and added to set $C(u)$. We then remove vertex $v$ from the graph.
7. Remove duplicate edges or self-loops and remove saturated/leader flags from the vertices.
```

Within the Connect2Hops subroutine, every active vertex v attempts to connect itself to a subset of the vertices in its 2-hop. If there are more candidates than the budget of v allows, we discard some of them arbitrarily. To formalize this, we use $N(u)$ to denote the neighbors of a vertex u.

Next, in the RelabelInterLevel subroutine, every vertex v that sees a vertex u of a higher level in its neighborhood, is "relabeled" to that vertex. That is, any occurrence of v in the edges is replaced with u. As a technical point, it might happen that we end up with a chain $v_1 \to v_2 \to v_3 \to \dots$ of relabelings where vertex v_1 has to be relabeled to v_2, v_2 has to be relabeled to v_3, and so on. In each iteration of the algorithm, we *only apply the direct relabeling of every vertex*, that is v_1 ends up with label v_2, v_2 ends up with label v_3, etc. An example of this is illustrated in Figure 5.1.

Finally, the last subroutine RelabelIntraLevel, is where we increase the budgets/levels.

5.2.2 Analysis of Algorithm 1 – Correctness

Correctness. We first show that the algorithm indeed computes the connected components of the given graph. The following lemma follows directly from the fact that Algorithm 1 does not split or merge connected components.

Lemma 5.3. *Let $\mathcal{S}_1, \ldots, \mathcal{S}_k$ be the connected components of G at the end of Algorithm 1. Then, the family of sets $\{\cup_{v \in \mathcal{S}_i} C(v) \mid i \in [k]\}$ is equal to the family of vertex sets of connected components of the original graph.*

Proof. We use induction to show that the claim is true at the end of each iteration r of Algorithm 1. Before we start the algorithm, i.e., when $r = 0$, for every vertex v we have $C(v) = \{v\}$. Therefore, clearly the base case holds. For the rest of the proof, it suffices to show that the three steps of Connect2Hops, RelabelInterLevel, and RelabelIntraLevel maintain this property.

Within the Connect2Hops subroutine, we only add edges to the graph. The only way that this operation may hurt our desired property, is if the added edges connect two different connected components of the previous iteration. However, every added edge is between two vertices of distance at most 2 (and thus in the same component) implying that this cannot happen.

For the RelabelInterLevel subroutine, we first have to argue that the relabelings do not change the connectivity structure of the graph. It is clear that two disconnected components cannot become connected since each vertex is relabeled to another vertex of the same connected component. Moreover, we have to argue that one connected component does not become disconnected. For this, consider a path between two vertices u and v of the same component. After relabeling vertices, there is still a walk between the corresponding vertices to u and v, thus they remain connected. Finally, observe that once a vertex v is relabeled to some vertex u, in Line 5 of the RelabelInterLevel subroutine, we add every vertex of $C(v)$ to $C(u)$. This ensures that for every component $\mathcal{S}$, the set $\cup_{v \in \mathcal{S}} C(v)$ does not lose any vertex and thus remains unchanged.

Similarly, in the RelabelIntraLevel step, the vertices only get contracted to the leaders in their 2-hop and once removed from the graph, a vertex v passes every element in $C(v)$ to $C(u)$ of another vertex u in its component, thus the property is maintained. □

5.2.3 Analysis of Algorithm 1 – Round Complexity

In order to pave the way for future discussions, we start with some definitions. We use $G_r = (V_r, E_r)$ to denote the resulting graph by the end of iteration r of Algorithm 1. Therefore, we have $V = V_0 \supseteq V_1 \supseteq V_2 \supseteq \dots$ as we do not add any vertices to the graph. Moreover, for any vertex $v \in V$ and any iteration $r \geq 0$, we define $\text{next}_r(v)$ to be the vertex $w \in V_r$ such that $v \in C(w)$ by the end of iteration r. That is, $\text{next}_r(v)$ is the vertex that *corresponds* to v by the end of iteration r.

Observation 5.4. *Let $v \in V_r$ be an active vertex. Then for any $r' \leq r$, we have $\text{next}_{r'}(v) = v$.*

For any iteration $r \geq 0$ and any vertex $v \in V$, we respectively use $\ell_r(v)$, $b_r(v)$ and $C_r(v)$ to denote the value of $\ell(\text{next}_r(v))$, $b(\text{next}_r(v))$ and $C(\text{next}_r(v))$ by the end of iteration r. Furthermore, for any two vertices $v, u \in V$, we use $\text{dist}_r(u, v)$ to denote the length of the shortest path between $\text{next}_r(u)$ and $\text{next}_r(v)$ in graph G_r.

The following claim implies that the corresponding level of a vertex is non-decreasing over time.

Claim 5.5. *For any vertex $v \in V$ and any $r \geq r'$, we have $\ell_r(v) \geq \ell_{r'}(v)$.*

Proof. We use induction on r. For the base case with $r = r'$, we clearly have $\ell_r(v) = \ell_{r'}(v)$. Suppose, by the induction hypothesis, that $\ell_{r-1}(v) \geq \ell_{r'}(v)$. If in iteration r of the algorithm, vertex $\text{next}_{r-1}(v)$ is not relabeled, i.e., if we have $\text{next}_r(v) = \text{next}_{r-1}(v)$, then $\ell_r(v) = \ell_{r-1}(v)$ by definition and the fact that the level of a particular vertex cannot decrease in one iteration. Therefore, by the induction hypothesis, we have $\ell_r(v) = \ell_{r-1}(v) \geq \ell_{r'}(v)$. On the other hand, if vertex v is relabeled in iteration r, i.e., if $\text{next}_r(v) \neq \text{next}_{r-1}(v)$, then it suffices to show that it is relabeled to a vertex whose level is higher. This is clear from description of the algorithm. A vertex that gets relabeled within the RelabelInterLevel subroutine, does so if and only if the new vertex has a higher level. Similarly, in within the RelabelIntraLevel subroutine of Algorithm 1, any vertex v that is contracted to another vertex does so if it is a marked saturated vertex of the same level, whose level increases by the end of the iteration. $\square$

The next claim shows, in a similar way, that the distance between the corresponding vertices of two vertices u and v is non-increasing over time.

Claim 5.6. *For any two vertices $v, u \in V$ and any $r \geq r'$,* $\text{dist}_r(u, v) \leq \text{dist}_{r'}(u, v)$.

Proof. Similar to the proof of Claim 5.5, we can show this by induction on r and, thus, the problem reduces to showing that in one iteration the corresponding distance between two vertices cannot increase. To show this, fix a shortest path p between two vertices v and u at any iteration. Within the next iteration, the Connect2Hops subroutine does not affect this path as it only adds some edges to the graph. Moreover, the only effect of the relabeling steps on this path is that it may shrink it as one vertex of the path can be relabeled to one of its neighbors in the path. However, relabeling can in no way destroy or increase the length of this path. Thus, the lemma follows. □

Our next observation follows directly from the description of the algorithm.

Observation 5.7. *For any $r \geq 0$ and any vertices $u, v \in V$ with $\text{dist}_r(u, v) \leq 1$, we have $\ell_{r+1}(u) \geq \ell_r(v)$ and $\ell_{r+1}(v) \geq \ell_r(u)$.*

Proof. This follows since any vertex who sees a neighbor of a higher level, is relabeled to its neighbor with the highest level in subroutine RelabelInterLevel of Algorithm 1. □

Claim 5.8. *With high probability for any iteration r and any vertex $v \in V_r$, if v becomes saturated in the next iteration $r + 1$, then there is at least one leader of the same level in its 2-hop, thus $\ell_{r+1}(v) \geq \ell_r(v) + 1$.*

Proof. If v is saturated, then by definition, it has at least $b(v)$ vertices in its inclusive 2-hop (i.e., the set $v \cup N(v) \cup N(N(v))$) that have the same level as that of v and are also saturated. To see this, note that if v is marked as saturated in Line 1 of RelabelIntraLevel, then it has at least $b(v)$ other active direct neighbors with level at least $b(v)$ all of which will be marked as saturated in Line 2. Furthermore, if a vertex v is marked as saturated in Line 2, then it has a saturated neighbor which has $b(v)$ direct saturated neighbors as just described. Thus v's 2-hop will include $b(v)$ saturated vertices as desired.

It suffices to show that one of these $b(v)$ saturated vertices will be marked as a leader with high probability. Recall that we mark each vertex independently with probability $\frac{3 \log n}{b(v)}$, thus

$$\mathbf{Pr}\left[\ell_{r+1}(v) = \ell_r(v)\right] \leq \left(1 - \frac{3 \log n}{b(v)}\right)^{b(v)} \leq \exp(-3 \log n) \leq 1/n^3.$$

By a union bound over all vertices, and over the total number of iterations of the algorithm which is clearly less than n^2, we get that with probability at least $1 - 1/n$ every vertex that gets saturated sees a marked vertex in its 2-hop and its corresponding level will thus be increased in the next iteration. □

The next lemma highlights a key property of the algorithm and will be our main tool in analyzing the round complexity. Intuitively, it shows that with high probability, after every 4 iterations of Algorithm 1, every vertex v is either connected to its 2-hop, or its corresponding level increases by at least 1.

Lemma 5.9. *Let $u, v \in V$ be two vertices with $\text{dist}_r(u,v) = 2$ for some iteration r. If $\text{dist}_{r+4}(u,v) \geq 2$, then $\ell_{r+4}(v) \geq \ell_r(v)+1$ and $\ell_{r+4}(u) \geq \ell_r(u)+1$. This holds for all vertices u and v and over all iterations of the algorithm with high probability.*

Proof. By Claim 5.6, we have $\text{dist}_{r+4}(u,v) \leq \text{dist}_r(u,v) = 2$. As such, to prove the lemma, it suffices to obtain a contradiction by assuming that $\text{dist}_{r+4}(u,v) = 2$ and (w.l.o.g.) $\ell_{r+4}(u) = \ell_r(u)$.

Recall that the lemma assumes that $\text{dist}_r(u,v) = 2$. Therefore, there must exist a vertex w with $\text{dist}_r(u,w) = \text{dist}_r(w,v) = 1$. By an application of Observation 5.7, we have

$$\ell_{r+4}(u) \geq \ell_{r+3}(w) \geq \ell_{r+2}(v) \geq \ell_{r+1}(w) \geq \ell_r(u).$$

Combining this with our assumption that $\ell_{r+4}(u) = \ell_r(u)$, we get

$$\ell_{r+4}(u) = \ell_{r+3}(w) = \ell_{r+2}(v) = \ell_{r+1}(w) = \ell_r(u). \tag{5.1}$$

Moreover, by Claim 5.5 which states the levels are non-decreasing over time, we have

$$\ell_{r+4}(u) \geq \ell_{r+2}(u) \geq \ell_r(u) \qquad \text{and} \qquad \ell_{r+3}(w) \geq \ell_{r+2}(w) \geq \ell_{r+1}(w). \tag{5.2}$$

Combination of (5.1) and (5.2) directly implies the following two useful inequalities.

Observation 5.10. $\ell_{r+2}(u) = \ell_{r+2}(w) = \ell_{r+2}(v)$.

Proof. Inequality $\ell_{r+4}(u) \geq \ell_{r+2}(u) \geq \ell_r(u)$ of (5.2) combined with equality $\ell_{r+4}(u) = \ell_r(u)$ of (5.1) implies $\ell_{r+2}(u) = \ell_r(u)$. This combined with $\ell_r(u) = \ell_{r+2}(v)$ of (5.1), gives $\ell_{r+2}(u) = \ell_{r+2}(v)$.

Inequality $\ell_{r+3}(w) \geq \ell_{r+2}(w) \geq \ell_{r+1}(w)$ of (5.2) combined with equality $\ell_{r+3}(w) = \ell_{r+1}(w)$ of (5.1) implies $\ell_{r+2}(w) = \ell_{r+1}(w)$. Combined with $\ell_{r+1}(w) = \ell_{r+2}(v)$ of (5.1), it gives $\ell_{r+2}(w) = \ell_{r+2}(v)$. □

Observation 5.11. $\ell_{r+4}(u) = \ell_{r+2}(u)$.

Proof. Inequality $\ell_{r+4}(u) = \ell_r(u)$ due to (5.1) combined with inequality $\ell_{r+4}(u) \geq \ell_{r+2}(u) \geq \ell_r(u)$ of (5.2) implies $\ell_{r+4}(u) = \ell_{r+2}(u) = \ell_r(u)$. □

Observation 5.10 implies that the corresponding levels of all three vertices u, w and v should be the same at the end of iteration $r+2$. Thus, within the Connect2Hops subroutine of iteration $r+3$, we have $\text{next}_{r+2}(v) \in H(\text{next}_{r+2}(u))$; now either we connect $\text{next}_{r+2}(u)$ and $\text{next}_{r+2}(v)$ which reduces their distance to 1 contradicting our assumption that $\text{dist}_{r+4}(u, v) = 2$, or otherwise vertex $\text{next}_{r+2}(u)$ spends its budget to get connected to at least $b_{r+2}(u)$ other vertices of level at least $\ell_{r+2}(u)$. Let N be the set of these neighbors of $\text{next}_{r+2}(u)$. There are three scenarios and each leads to a contradiction:

- If for a vertex $x \in N$, $\ell_{r+2}(x) > \ell_{r+2}(u)$, then by Observation 5.7, the level of the corresponding vertex of u increases in the next iteration and we have $\ell_{r+4}(u) \geq \ell_{r+3}(u) > \ell_{r+2}(u)$ which contradicts equality $\ell_{r+4}(u) = \ell_{r+2}(u)$ of Observation 5.11.

- If a vertex $x \in N$ is inactive, then x is in a chain by definition of inactive vertices. Every vertex in a chain has a vertex of higher level next to it, thus $\ell_{r+3}(x) > \ell_{r+2}(x)$ by Observation 5.7. Furthermore, since $x \in N$, we know $\ell_{r+2}(x) \geq \ell_{r+2}(u)$. This means that $\text{next}_{r+3}(u)$ has a neighbor of strictly higher level, thus by Observation 5.7, we have to have $\ell_{r+4}(u) > \ell_{r+2}(u)$ which contradicts equality $\ell_{r+4}(u) = \ell_{r+2}(u)$ of Observation 5.11.

- If the two cases above do not hold, then after applying Connect2Hops in iteration $r+2$, $\text{next}_{r+2}(u)$ has at least $b_{r+2}(u)$ active neighbors of level exactly $\ell_{r+2}(u)$. Furthermore, vertex $\text{next}_{r+2}(u)$ itself has to be active, or otherwise its corresponding level has to increase in the next iteration which is a contradiction. This means by definition that $\text{next}_{r+2}(u)$ is saturated during iteration 3. By Claim 5.8, with high probability the corresponding level of every saturated vertex increases by at least one in the next iteration, and thus we get $\ell_{r+3}(u) > \ell_{r+2}(u)$ which, again, would imply $\ell_{r+4}(u) \neq \ell_{r+2}(u)$ contradicting Observation 5.11.

To wrap up, we showed that if the distance between the corresponding vertices to u and v after the next 4 iterations is not decreased to at most 1, then the corresponding level of u and v has to go up by one with high probability. $\square$

As discussed before, Lemma 5.9 implies that after every $O(1)$ consecutive iterations of Algorithm 1, each vertex either is (roughly speaking) connected to the vertices in its 2-hop or sees a level increase. It is easy to show that if *every* vertex is connected to the vertices in its 2-hop, the diameter of the graph is reduced by a constant factor, and thus after $O(\log D)$ iterations every connected component becomes a clique. Notice, however, that Lemma 5.9

does not guarantee this, as for some vertices, we may only have a level increase instead of connecting them to their 2-hop. Let L be an upper bound on the level of the vertices throughout the algorithm. (We later show in Lemma 5.16 that $L = O(\log\log_{T/n} n)$.) Since the maximum possible level is L, each vertex does not connect 2-hops for at most L iterations. Therefore, if for instance, within each of the first L iterations of the algorithm, the corresponding level of *every* vertex increases, we cannot have any level-increases afterwards. Therefore within the next $O(\log D)$ iterations, each vertex connects 2-hops and every connected component becomes a clique. Overall, this takes $O(L + \log D)$ iterations. In reality, however, the level increases do not necessarily occur in bulk within the first L iterations of the algorithm. In fact, Lemma 5.9 alone is not enough to show a guarantee of $O(L + \log D)$. To get around this problem, we need to use another crucial property of the algorithm highlighted in Observation 5.7. A proof of sketch of how we combine these two properties to get our desired bound was already given in Section 5.1. The following lemma formalizes this.

Lemma 5.12. *Let L be an upper bound on the number of times that the corresponding level of a vertex may increase throughout the algorithm. Only $O(L + \log D)$ iterations of the for loop in Algorithm 1 suffices to make sure that with high probability, every remaining connected component becomes a clique.*

Proof. It will be convenient for the analysis to call every 4 consecutive iterations of the for-loop in Algorithm 1 a *super-iteration*. That is, for any $i \geq 1$, we define the ith super-iteration to be the combination of performing iterations $4i - 3, 4i - 2, \ldots, 4i$ of Algorithm 1.

Fix two arbitrary vertices u and v in a connected component of the original graph G. It suffices to show that after running the algorithm for $R := O(L + \log D)$ super-iterations, the corresponding vertices to u and v, are at distance at most 1. To show this, we maintain a path between u and v and update it over time. We use P_r to denote the maintained path by the end of super-iteration r, i.e., the path is updated every four iterations. The initial path, P_0, is any arbitrary shortest path between u and v in the original graph G; notice that P_0 has at most $D + 1$ vertices, as the diameter of G is D. As we move forward, u and v may be relabeled; nonetheless, the path P_r will be a path from vertex $\text{next}_{4r}(u)$ (which is the corresponding vertex to u by the end of iteration $4r$ or equivalently super-iteration r) to vertex $\text{next}_{4r}(v)$. Crucially, the path P_r is not necessarily the shortest path between $\text{next}_{4r}(v)$ and $\text{next}_{4r}(u)$ in G_r. The reason is that the naive shortest paths may "radically" change from one iteration to another. Instead, we carefully construct P_r to ensure that it passes only through the corresponding vertices of the vertices in P_{r-1}, which also inductively indicates that every vertex in P_r is in set $\{\text{next}_{4r}(w) \mid w \in P_0\}$.

To use these gradual updates, for every r, we define a potential function $\Phi_r : V(P_r) \to \mathbb{N}$ that maps every vertex of path P_r to a positive integer. The definition of function Φ_r and construction of path P_r are recursively based on Φ_{r-1} and P_{r-1}. As for the base case, we have $\Phi_0(v) = 1$ for every vertex $v \in P_0$. For the rest of the iterations, we follow the following steps.

To construct P_r from P_{r-1}, we first apply the relabelings of iterations $4r-3, \ldots, 4r$, on the vertices in P_{r-1}. That is, the sequence $P_{r-1} = (w_1, \ldots, w_s)$ becomes $Q = (q_1, \ldots, q_s)$ where $q_i = \text{next}_{4r}(w_i)$. Note that multiple vertices in P_{r-1} may have been relabeled to the same vertex throughout these four iterations, and thus the elements in Q are not necessarily unique. Next, we use an s-bit *mask* vector $K \in \{0,1\}^s$ to denote a subsequence[3] of Q that corresponds to the vertices in P_r. That is, P_r contains the ith element of Q if and only if $K_i = 1$. To guarantee that P_r is indeed a path and that it has some other useful properties, our mask vector K should satisfy the following properties:

(P1) $K_1 = K_s = 1$.

(P2) If for some $i, j \in [s]$ with $i \neq j$, we have $q_i = q_j$, then at most one of K_i and K_j is 1.

(P3) If for some $1 \leq i < j \leq s$ with $K_i = K_j = 1$, there is no k with $i < k < j$ for which $K_k = 1$, then q_i and q_j should have a direct edge in graph G_{4r}.

(P4) If for some $i \in [s]$, we have $\ell_{4r}(q_i) = \ell_{4(r-1)}(w_i)$ (i.e., the level of the corresponding vertex to w_i is not increased) and $K_i = 1$, then at least one of K_{i-2}, K_{i-1}, K_{i+1} or K_{i+1} should be 0.[4]

Property P1 guarantees that the path of the next iteration remains to be between the corresponding vertices to u and v. Property P2 ensures that we do not revisit any vertex in P_r which is necessary if we want P_r to be a path. Property P3 ensures that every two consecutive vertices in P_r are neighbors in G_{4r}, which again, is necessary if we want P_r to denote a path in G_{4r}. Finally, Property P4 guarantees that if the corresponding level of a vertex $w_i \in V_{4(r-1)}$ does not increase in iterations $4r-3, \ldots, 4r$, and that q_i (which is the corresponding vertex to w_i after these four iterations) is included in path P_r, there is a vertex in $\{w_{i-2}, w_{i-1}, w_{i+1}, w_{i+2}\}$ whose corresponding vertex at the next iteration is not

[3]A *subsequence* is a derived from another sequence by deleting some or no elements of it without changing the order of the remaining elements.

[4]K is an s-bit vector, but assume for preciseness of definition that $K_{-1} = K_0 = K_{s+1} = K_{s+2} = 1$.

included in P_r. Note that we have to be careful that by satisfying Property P4, we do not violate Property P3. In other words, we have to make sure that once we drop the (2-hop) neighboring vertices of w_i in P_{r-1} from P_r, P_r remains to be a connected path. However, this can be guaranteed by Lemma 5.9 which says if the corresponding level of a vertex does not increase in 4 iterations, its distance to the vertices in its 2-hop decreases to at most 1 (see also Section 5.1). Overall, we get the following result.

Claim 5.13. *If $q_1 \neq q_s$ and if $|P_{r-1}| > 3$, then with high probability there exists a mask vector K satisfying Properties P1-P4.*

Construction of function Φ_r is also based on the mask vector K that we construct P_r with. Recall that Φ_r is a function from the vertices in P_r to $\mathbb{N}$. Therefore, in order to describe Φ_r, it suffices to define the value of Φ_r on vertex q_i iff $K_i = 1$. Assuming that $K_i = 1$, define l_i to be the smallest number in $[1, i]$ such that $\sum_{j \in l_i}^{i-1} K_j \leq 1$. In a similar way, define r_i to be the largest number in $[i, s]$ where $\sum_{j=i+1}^{r_i} K_j \leq 1$. Having these, we define $\Phi_r(q_i)$ in the following way:

$$\Phi_r(q_i) := \Phi_{r-1}(w_i) + \frac{1}{4}\sum_{j=l_i}^{i-1} \Phi_{r-1}(w_j) + \frac{1}{4}\sum_{j=i+1}^{r_i} \Phi_{r-1}(w_j). \tag{5.3}$$

The next two claims are the main properties of function Φ_r that we use in proving Lemma 5.12.

Claim 5.14. *For any $r \geq 0$ and any vertex $w \in P_r$ with level $\ell = \ell_{4r}(w)$, $\Phi_r(w) \geq (1.25)^{r-\ell}$.*

Proof. We use induction on r. For the base case with $r = 0$, we have $\Phi_0(w) = 1$ and since it is before the first iteration, we have $\ell = 0$. Thus, we have $\Phi_0(w) \geq (1.25)^{0-0} = 1$. The induction hypothesis guarantees for every vertex w' of path P_{r-1} with level ℓ', that $\Phi_{r-1}(w') \geq (1.25)^{r-1-\ell'}$. We show that this carries over to the vertices of P_r as well.

We would like to prove that for every vertex $w \in P_r$, we have $\Phi_r(q) \geq (1.25)^{r-\ell_{4r}(q)}$. We know by construction of P_r from P_{r-1} that vertex w of P_r is the corresponding vertex of some vertex $w'_i \in P_{r-1}$ with $K_i = 1$ where K denotes the mask vector that we use to construct P_r from P_{r-1}, i.e., $w = \text{next}^4_{4(r-1)}(w'_i)$. By the induction hypothesis, we have

$$\Phi_{r-1}(w'_i) \geq (1.25)^{r-1-\ell_{4(r-1)}(w'_i)}. \tag{5.4}$$

Therefore, if during super-iteration r, the corresponding level of w_i increases, i.e., if we have $\ell_{4r}(w) \geq \ell_{4(r-1)}(w'_i) + 1$, then we have

$$\Phi_r(w) \geq \Phi_{r-1}(w'_i) \overset{\text{By (5.4)}}{\geq} (1.25)^{r-1-\ell_{4(r-1)}(w'_i)} \geq (1.25)^{r-\ell_{4r}(w)},$$

where the first inequality comes from the fact that $\Phi_r(w_i) > \Phi_{r-1}(w'_i)$ which itself is directly followed by (5.3). This means that if the corresponding level of w'_i remains unchanged within super-iteration r, we have our desired bound on $\Phi_r(w)$. The only scenario that is left is if the corresponding level of w'_i remains unchanged, i.e., $\ell_{4r}(w) = \ell_{4(r-1)}(w'_i)$.

If the corresponding level of w'_i remains unchanged during super-iteration r, then by Property P4 of the mask vector K, at least one of $\{K_{i-2}, K_{i-1}, K_{i+1}, K_{i+2}\}$ is 0. Suppose without loss of generality that $K_{i-2} = 0$. First, observe that we have to have

$$\ell_{4(r-1)}(w'_{i-2}) \leq \ell_{4(r-1)}(w'_i). \tag{5.5}$$

The reason is that if level of w'_{i-2}, which has distance at most 2 from w'_i in graph $G_{4(r-1)}$, has a higher level than w'_i, then by Observation 5.7, the corresponding level of w'_i after at most 2 iterations (still within a super-iteration) should increase to at least $\ell_{4(r-1)}(w'_{i-2})$ which would contradict the assumption that the corresponding level of w'_i remains unchanged for 4 iterations. This means that by the induction hypothesis, now on vertex w'_{i-2} of path P_{r-1}, we have

$$\Phi_{r-1}(w'_{i-2}) \geq (1.25)^{r-1-\ell_{4(r-1)}(w'_{i-2})} \overset{\text{By (5.5)}}{\geq} (1.25)^{r-1-\ell_{4(r-1)}(w'_i)}. \tag{5.6}$$

Now, recall that we assumed $K_{i-2} = 0$. This means, by construction of Φ_r using (5.3), that we have to have

$$\Phi_r(w) \geq \Phi_{r-1}(w'_i) + \frac{1}{4}\Phi_{r-1}(w'_{i-2}). \tag{5.7}$$

Therefore, we have

$$\begin{aligned}
\Phi_r(w) &\geq \Phi_{r-1}(w'_i) + \frac{1}{4}\Phi_{r-1}(w'_{i-2}) && \text{By (5.7).}\\
&\geq \Big((1.25)^{r-1-\ell_{4(r-1)}(w'_i)}\Big) + \frac{1}{4}\Big((1.25)^{r-1-\ell_{4(r-1)}(w'_i)}\Big) && \text{By induction hypothesis and (5.6).}\\
&= 1.25\Big((1.25)^{r-1-\ell_{4(r-1)}(w'_i)}\Big) = (1.25)^{r-\ell_{4(r-1)}(w'_i)}\\
&\geq (1.25)^{r-\ell_{4r}(w)}. && \text{Since } \ell_{4r}(w) \geq \ell_{4(r-1)}(w'_i).
\end{aligned}$$

Concluding the proof of Claim 5.14. □

Claim 5.15. *For any $r \geq 0$ with $|P_r| > 3$, we have $\sum_{w \in P_r} \Phi_r(w) \leq D + 1$.*

Proof. The inequality $\sum_{v \in P_0} \Phi_r(v) \leq D + 1$ is followed by the fact that P_0, which is a shortest path between u and v in the original graph has at most $D + 1$ vertices and that $\Phi_0(w) = 1$ for any vertex $w \in P_0$. Moreover, one can easily show that for any $r > 0$, we have $\sum_{w \in P_r} \Phi_r(w) \leq \sum_{w \in P_{r-1}} \Phi_{r-1}(w)$ directly by the definition of Φ_r from Φ_{r-1} and

Property P1 of the mask vectors used. Combining these two facts via a simple induction on r proves the claim. □

We are now ready to prove Lemma 5.12. Run the algorithm for $R := L + 4\log D$ super-iterations. If path P_R has at most 3 vertices, we are done since our goal is to show that the distance between the corresponding vertices of u and v in graph G_R is at most 2 — which itself would imply that every connected component in G_R has diameter at most 2. In fact, we show that this should always be the case.[5] Suppose for the sake of contradiction that we can continue to super-iteration R in constructing P_R and Φ_R and still have $|P_R| > 3$. Let $u_R := \text{next}_{4R}(u)$ be the corresponding vertex to vertex u by the end of super-iteration R. Property P1 of our mask vectors in constructing paths $P_1, \ldots, P_R$ ensures that path P_R should start with vertex u_R. By Claim 5.14, we have

$$\Phi_R(u_R) \geq (1.25)^{R-\ell_R(u_R)} \geq (1.25)^{R-L}, \tag{5.8}$$

where the latter inequality comes from the assumption that L is an upper bound on the level of every vertex. Now, since $R = L + 4\log D$, we have

$$R - L \geq 4\log D. \tag{5.9}$$

Combining (5.8) with (5.9) we get

$$\Phi_R(u_R) \geq (1.25)^{4\log D} > D + 1.$$

However, this contradicts with Claim 5.15 which guarantees $\Phi_R(u_R)$ should be less than $D+1$. Therefore, our initial assumption that R can be as large as $L + 2\log D$ cannot hold; meaning that in $O(L+\log D)$ iterations, the remaining graph will be a collection of connected components of diameter $O(1)$.

Once the diameter of every remaining connected component gets below $O(1)$, it is easy to confirm that in the next $O(L)$ iterations of the algorithm, the diameter reduces to 1 (i.e., every connected component becomes a clique). To see this, note that since the diameter is $O(1)$, the maximum level within each component propagates to all the vertices in $O(1)$ iterations. If this budget is not enough for a vertex to connect 2-hop, its level increases by Lemma 5.9. This level, again, propagates to all other vertices. Eventually, after the next $O(L)$ iterations, the vertices will reach the maximum possible level and thus have enough budget to get connected to every remaining vertex in the component.

[5]More precisely, the "always" here is conditional on the assumption that our high probability events hold. This is not a problem since otherwise we say the algorithm fails and this happens with probability $\leq 1/n$.

Overall, it takes $O(L+\log D)$ iterations until the diameter of every remaining connected component becomes $O(1)$ and after that, at most $O(L)$ other iterations for them to become cliques. □

To continue, we give the following upper bound on the levels.

Lemma 5.16. *For any vertex v, the value of $\ell(v)$ never exceeds $O(\log\log_{T/n} n)$.*

Proof. Observe that the only place throughout Algorithm 1 that we increase the level of a vertex is in Line 4 of the RelabelIntraLevel procedure. Within this line, the budget of the vertex is also increased from $b(v)$ to $b(v)^{1.25}$. Now, given that the initial budget of every vertex is $\beta_0 = (T/n)^{1/2}$, throughout the algorithm, we have $b(v) = \beta_0^{1.25^{\ell(v)}}$. On the other hand, observe that if a vertex reaches a budget of n, it will not be marked as saturated, and thus, we do not update its level/budget anymore. Therefore, we have $b(v) = \beta_0^{(1.25)^{\ell(v)}} \leq n$ which means $\ell(v) \leq \log_{1.25}\log_{\beta_0} n = O(\log\log_{T/n} n)$. □

Combining the two lemmas above, we can prove the following bound on the round complexity.

Lemma 5.17. *With high probability the number of rounds executed by Algorithm 1 is $O(\log D + \log\log_{T/n} n)$.*

Proof. By Lemma 5.12, it takes only $O(L+\log D)$ iterations for Algorithm 1 to halt where L is an upper bound on the level of the vertices. Lemma 5.16 shows that $L = O(\log\log_{T/n} n)$. Therefore, the round complexity of Algorithm 1 is $O(\log D + \log\log_{T/n} n)$. □

5.2.4 Analysis of Algorithm 1 – Implementation Details & Space

Lemma 5.18. *The total space used by Algorithm 1 is $O(T)$.*

Proof. To bound the total space used by Algorithm 1, we have to bound the number of edges that may exist in the graph. More specifically, we have to show that within the Connect2Hops subroutine, we do not add too many edges to the graph. Recall that we control this with the budgets. It is not hard to argue that sum of budgets of remaining vertices in each round of the algorithm does not exceed T. However, there is a subtle problem that prevents this property to be sufficient for bounding the number of edges in the graph. The reason is that throughout the algorithm, the degree of a vertex may be much larger than its budget. For instance in the first iteration, a vertex may have a degree of up to $\Omega(n)$ while the budgets are much smaller.

For the analysis, we require a few definitions. For every iteration r and any vertex $v \in V_r$, we define $d_r(v)$ to be the number of neighbors of v in G_r with level at least $\ell_r(v)$. Moreover, we define the *remaining budget* $s_r(v)$ of v to be $\max\{0, b_r(v) - d_r(v)\}$ if v is active and 0 otherwise. To clarify the definition, note that within the Connect2Hops subroutine, each vertex v connects to at most $s_r(v)$ new vertices. We further define

$$y_r := |E_r| + \sum_{v \in V_r} s_r$$

to be the *potential space* by the end of iteration r. It is clear by definition that y_r is an upper bound on the total number of edges in the graph after iteration r. Therefore it suffices to show that $y_r = O(T)$ for any r. The base case follows immediately:

Observation 5.19. $y_0 = O(T)$.

Proof. We have $y_0 = |E_0| + \sum_{v \in V_0} s_r \leq m + n \cdot (\frac{T}{n})^{1/2} < m + T \leq O(T)$ where the last inequality comes from the fact that $T = \Omega(m)$. □

In what follows, we argue that for any r, we have $y_r \leq y_0 + O(T) = O(T) + O(T) = O(T)$ as desired. To do this, we consider the effect of each of the three subroutines of Algorithm 1 in any iteration $r+1$ on the value of y_{r+1} compared to y_r. We first show that the two procedures Connect2Hops and RelabelInterLevel cannot increase the potential space. We then give an upper bound of $O(T)$ on the increase in the potential space due to procedure RelabelIntraLevel over the course of the algorithm (i.e., not just one round).

Connect2Hops **procedure.** In the Connect2Hops procedure, each vertex v connects itself to at most $s_r(v)$ other vertices of level at least $\ell(v)$ in its 2-hop as described above. These added edges, will then decrease the remaining budget of v by definition. Therefore, for any edge that is added to the graph, the remaining budget of at least one vertex is decreased by 1. Thus, the total potential space cannot increase.

RelabelInterLevel **procedure.** Next, within the RelabelInterLevel procedure, we do not add any edges to the graph. Therefore, the only way that we may increase the potential space is by increasing the remaining budget of the vertices. If a vertex gets relabeled to a higher level neighbor, the algorithm marks it as inactive; this by definition decreases its remaining budget to 0. As such, it only suffices to consider the remaining budget of the vertices that are not relabeled; take one such vertex v. Recall that the remaining budget of v depends on the level of the neighbors of v as well. The crucial property here is that whenever a vertex is relabeled to a neighbor, its corresponding level is increased. This implies that the change in the corresponding level of v's neighbors cannot increase the remaining budget of v.

There is still one way that v's remaining budget may increase: if an edge $\{v, u\}$ with $\ell_r(u) \geq \ell_r(v)$ is removed from the graph. Recall that an edge may be removed from the graph within Line 3 of RelabelInterLevel where we remove duplicate edges or self-loops. Note that if removal of an edge increases the remaining budget of one of its endpoints only, then the potential space y_{r+1} does not change as the increase in $\sum_v s_{r+1}(v)$ is canceled out by the decrease in $|E_{r+1}|$. However, we have to argue that removal of an edge cannot increase the remaining budget of its both end-points. To see this, observe that the graph, before the RelabelInterLevel procedure cannot have any duplicate edges or self-loops (as we must have removed them before) and all these edges have been created within this iteration. Take an edge $\{u, v\}$ and suppose that there are multiple duplicates of it. All, but at most one, of duplicates of $\{u, v\}$ are the result of the relabelings. Call these the *relabeled* edges and suppose due to symmetry that any removed edge is relabeled. Consider an edge e' that is relabeled to $\{u, v\}$ and is then removed. At least one of endpoints of e' must be some vertex w which is relabeled to either u or v, say u w.l.o.g. An equivalent procedure is to remove e' before w is relabeled to u and the outcome would be the same. Since $u \notin e'$, there is no way that removing e' would change the remaining budget of u. On the other hand, since w is relabeled and does not survive to V_{r+1} it does not have any effect on s_{r+1}. This means that removing any duplicate edge increases sum of remaining budgets by at most 1 thus the potential space cannot increase.

RelabelIntraLevel procedure. We showed that subroutines Connect2Hops and RelabelInterLevel cannot increase the potential space of the previous round. Here, we consider the effect of the last subroutine RelabelIntraLevel. Similar to RelabelInterLevel, we do not add any edges to the graph. Therefore, we only have to analyze the remaining budgets after this procedure.

First take a vertex v that is not marked as saturated. The remaining budget of v may increase if some of its edges are removed because of duplicates which are caused by contracting saturated vertices to their leaders. However, precisely for the same argument that we had for the RelabelInterLevel procedure, removal of an edge can only increase the remaining budget of at most one of its end-points thus this does not increase the potential space.

Next, if a vertex v is marked as saturated but is not marked as a leader, by Claim 5.8 it is, w.h.p., going to get contracted to a leader and removed from the graph. Therefore, the only case for which the remaining budget of a vertex may increase is for saturated vertices that are marked as leaders. We assume the worst case. That is, we assume that if a vertex v is saturated and is marked as a leader within iteration $r+1$, then the potential space is increased by its new budget $b_{r+1}(v)$ (note, by definition, that remaining budget can never be larger than budget). Instead of analyzing the effect of this increase within one iteration, we show that the total sum of such increases over all iterations of the algorithm is bounded by $O(T)$.

Let us use β_i to denote the budget of vertices with level i and use n_i to denote the number of vertices that have been selected as a leader over the course of algorithm for at least i times. In other words, n_i denotes the total number of vertices that reach a level of at least i throughout the algorithm. We can bound sum of increases in potential space due to the RelabelIntraLevel procedure over all iterations of the algorithm by:

$$\sum_{i=1}^{\infty} \beta_i \cdot n_i. \tag{5.10}$$

Thus it suffices to bound this quantity by $O(T)$.

Claim 5.20. *For any $i \geq 1$, we have $\beta_i = (\beta_{i-1})^{1.25}$ and have $\beta_0 = (T/n)^{1/2}$.*

Proof. We have $\beta_0 = (T/n)^{1/2}$ for Line 1 of Algorithm 1. Furthermore, we have $\beta_i = (\beta_{i-1})^{1.25}$ due to Line 4 of RelabelIntraLevel which is the only place we increase the level of a vertex and at the same time increase its budget from b to $b^{1.25}$. □

We also have the following bound on n_i:

Claim 5.21. *For any $i \geq 1$ we have $n_i < n_{i-1} \cdot (\beta_{i-1})^{-0.25}$.*

Before describing the proof of Claim 5.21, let us first see we can get an upper bound of $O(T)$ for the value of (5.10). For any $i \geq 1$, we have

$$\beta_i \cdot n_i \overset{\text{Claim 5.20}}{=} (\beta_{i-1})^{1.25} \cdot n_i \overset{\text{Claim 5.21}}{<} (\beta_{i-1})^{1.25} \cdot (\beta_{i-1})^{-0.25} \cdot n_{i-1} = \beta_{i-1} \cdot n_{i-1}. \tag{5.11}$$

On the other hand, recall by Lemma 5.16 that the maximum possible level for a vertex is $L = O(\log\log_{T/n} n)$, meaning that for any $i > L$ we have $n_i = 0$; thus:

$$\sum_{i=1}^{\infty} \beta_i \cdot n_i = \sum_{i=1}^{L} \beta_i \cdot n_i \overset{(5.11)}{<} L(\beta_0 \cdot n_0) \leq O(\log\log n) \cdot (T/n)^{1/2} \cdot n,$$

where the last inequality comes from the fact that $\beta_0 = (T/n)^{1/2}$ due to Claim 5.20 and $n_0 = n$ by definition. Moreover, recall that $T \geq m + n\log^{\alpha} n$ for some large enough constant α, therefore $(T/n)^{1/2} \geq \log^{\alpha/2} n \gg O(\log\log n)$. This means that

$$O(\log\log n) \cdot (T/n)^{1/2} \cdot n \ll (T/n)^{1/2} \cdot (T/n)^{1/2} \cdot n = T.$$

Therefore, the total increase over the potential space over the course of the algorithm is at most T, meaning that indeed for any r, $y_r = O(T)$ and thus in any iteration we have at most $O(T)$ edges. It is only left to prove Claim 5.21.

Proof of Claim 5.21. To prove the claim, we show that for every vertex of level $i-1$ that gets saturated and is marked as a leader, there are $(\beta_{i-1})^{0.5}$ other unique vertices of level $i-1$ that are not marked as a leader and are removed from the graph. This is clearly sufficient to show $n_i \leq n_{i-1} \cdot (\beta_{i-1})^{-0.5} \ll n_{i-1} \cdot (\beta_{i-1})^{-0.25}$.

Consider some arbitrary iteration of the algorithm, and denote the set of saturated vertices and leaders with budget β_{i-1} by S and L respectively. Since each saturated vertex of budget β_{i-1} is chosen to be a leader independently with probability $\Theta(\frac{\log n}{\beta_{i-1}})$, we have $\mathbf{E}[|L|] = \Theta(\frac{\log n}{\beta_{i-1}}|S|)$. On the other hand, note that if $S \neq \emptyset$, we have $|S| \geq \beta_{i-1}$ since a vertex of budget β_{i-1} is marked as saturated in Line 1 of RelabelIntraLevel if it has at least β_{i-1} active neighbors with budget β_{i-1}, all of which will also get marked as saturated in Line 2 and thus join S. Therefore, $|S| \geq \beta_{i-1}$, meaning that $\mathbf{E}[|L|] = \Theta(\frac{\log n}{\beta_{i-1}}|S|) = \Omega(\log n)$. Thus, by a standard Chernoff bound argument, we get $|L| = \Theta(\frac{\log n}{\beta_{i-1}}|S|)$ with high probability. On the other hand, recall that by Claim 5.8, every non-leader saturated vertex will be contracted to a leader in its 2-hop. That is, all vertices in $S \setminus L$ will be removed from the graph. This, averaged over the vertices in L, we get

$$\frac{|S \setminus L|}{|L|} \geq \frac{|S| - |L|}{|L|} \geq \frac{|S|}{|L|} - 1 \geq \frac{|S|}{\Theta(\frac{\log n}{\beta_{i-1}}|S|)} - 1 \geq \Theta\left(\frac{\beta_{i-1}}{\log n}\right)$$

unique vertices that are removed from the graph per leader. Thus, it suffices to show that $\frac{\beta_{i-1}}{\log n} \gg (\beta_{i-1})^{0.5}$. For this, observe from Claim 5.20 and $T \geq m + n\log^{\alpha} n$ that $\beta_{i-1} \geq (T/n)^{1/2} \geq \log^{\alpha/2} n$ where α is some sufficiently large constant. It suffices to set $\alpha > 4$, say $\alpha = 5$, to get $\beta_{i-1} \gg \log^2 n$ and thus $\log n \ll (\beta_{i-1})^{0.5}$. This indeed means $\Theta(\frac{\beta_{i-1}}{\log n}) \gg \frac{\beta_{i-1}}{(\beta_{i-1})^{0.5}} = (\beta_{i-1})^{0.5}$ as desired. □

We already showed how proving Claim 5.21 gives an upper bound of $O(T)$ on the potential space of all iterations, which by definition, is also an upper bound on the number of edges in the graph, concluding the proof of Lemma 5.18. □

The next lemma is important for implementing the algorithm.

Lemma 5.22. *For any r, we have $\sum_{v\in V_r}(b_r(v))^2 \le T$.*

Proof. We use induction on r. For the base case with $r=0$, we have

$$\sum_{v\in V_0}(b_0(v))^2 = \sum_{v\in V_0}((T/n)^{1/2})^2 = T.$$

Suppose by the induction hypothesis that $\sum_{v\in V_{r-1}}(b_{r-1}(v))^2 \le T$, we prove $\sum_{v\in V_r}(b_r(v))^2 \le T$. For this, it suffices to show that $\sum_{v\in V_r}(b_r(v))^2 \le \sum_{v\in V_{r-1}}(b_{r-1}(v))^2$. Recall that we only increase the budgets in the RelabelIntraLevel procedure, thus we only have to consider the effect of this procedure. Take a vertex $v \in V_r$ and with $b_r(v) > b_{r-1}(v)$ (otherwise the sum remains unchanged clearly). Note that v must have been marked as a leader in iteration r and thus $b_r(v) = b_{r-1}(v)^{1.25}$. Recall from the proof of Claim 5.21 above that there are at least $b_{r-1}(v)^{0.5}$ unique vertices for v with budget $b_{r-1}(v)$ that get removed from the graph in iteration r. Denote the set of these vertices by U. Removing these vertices decreases sum of budgets' square by

$$\sum_{u\in U} b_{r-1}(u)^2 = |U|(b_{r-1}(v))^2 \ge b_{r-1}(v)^{2.5}. \tag{5.12}$$

On the other hand, increasing the budget of v from $b_{r-1}(v)$ to $b_{r-1}(v)^{1.25}$ increases the sum of budgets' square by $(b_{r-1}(v)^{1.25})^2 = b_{r-1}(v)^{2.5}$ which is not more than the decrease due to (5.12). Thus, we have $\sum_{v\in V_r}(b_r(v))^2 \le \sum_{v\in V_{r-1}}(b_{r-1}(v))^2$ as desired. □

It only remains to argue that each iteration of Algorithm 1 can be implemented in $O(1)$ rounds of MPC using $O(n^\delta)$ space per machine and with $O(T)$ total space. Since the proof is straightforward by known primitives, we omit the details. See [41] for details.

5.3 Improving Total Space to $O(m)$

In the previous section, we showed how it is possible to find connected components of an input graph in $O(\log D + \log\log_{T/n} n)$ rounds so long as $T \ge m + n\log^\alpha n$. In this section, we improve the total space to $O(m)$. The key to the prove is an algorithm that shrinks the number of vertices by a constant factor with high probability. More formally:

Lemma 5.23. *There exists an MPC algorithm using $O(n^\delta)$ space per machine and $O(m)$ total space that with high probability, converts any graph $G(V,E)$ with n vertices and m edges to a graph $G'(V',E')$ and outputs a function $f: V \to V'$ such that:*

1. $|V'| \leq \gamma n$ for some absolute constant $\gamma < 1$.

2. $|E'| \leq |E|$.

3. For any two vertices u and v in V, vertices $f(u)$ and $f(v)$ in V' are in the same component of G' if and only if u and v are in the same component of G.

We emphasize that Lemma 5.23 shrinks the number of vertices by a constant factor *with high probability*. This is crucial for our analysis. An algorithm that shrinks the number of vertices by a constant factor in expectation was already known [109] but cannot be used for our purpose.

Let us first show how Lemma 5.23 can be used to improve total space to $O(m)$ proving Theorem 5.2.

Proof of Theorem 5.2. First, observe that if $m \geq n \log^{\alpha} n$ or if $T \geq m + n \log^{\alpha} n$, then the algorithm of Section 5.2 already satisfiees the requirements of Theorem 5.2. Assuming that this is not the case, we first run the algorithm of Lemma 5.23 for $(\alpha \log_{1/\gamma} \log n)$ iterations. Let $G'(V', E')$ be the final graph and f be the function mapping the vertices of the original graph to those of G'. We have

$$|V'| \leq n \cdot \gamma^{\alpha \log_{1/\gamma} \log n} = n \cdot \log^{-\alpha} n.$$

Now, we can run the algorithm of Section 5.2 on graph G' to identify its connected components. The total space required for this is

$$O(|E'| + |V'| \cdot \log^{\alpha} |V'|) = O\Big(m + (n \cdot \log^{-\alpha} n) \cdot \log^{\alpha} n\Big) = O(m + n) = O(m).$$

We can then use function f to identify connected components of the original graph in $O(1)$ rounds.

Also, observe that the running time required is $O(\log \log n) + O(\log D + \log \log_{T/n} n)$. Given that $m \leq n \log^{\alpha} n$ and $T \leq m + n \log^{\alpha} n$ (as discussed above), we have $T/n = O(\text{poly} \log n)$, thus $\log \log_{T/n} n = \Omega(\log \frac{\log n}{\log \log n}) = \Omega(\log \log n)$; meaning that $O(\log \log n) + O(\log D + \log \log_{T/n} n) = O(\log D + \log \log_{T/n} n)$ and thus the running time also remains asymptotically unchanged. $\square$

We now turn to prove Lemma 5.23.

Proof of Lemma 5.23. In order to prove this lemma, we show that the following procedure reduces the number of vertices of the graph by a constant factor, with high probability.

This procedure only merges some neighboring vertices and hence maintains the connected components. In this procedure, without loss of generality, we assume that there is no isolated vertices. One can simply label and remove all isolated vertices at the beginning. It is easy to implement this procedure in $\frac{1}{\delta}$ rounds using $O(n^\delta)$ space per machine and a total space of $O(m)$ (see [41] for details).

Algorithm 2. Shrinks the number of vertices by a constant factor in $O(1)$ rounds w.h.p.

1. For each vertex v, draw a directed edge from v to its neighbor with the minimum id.
2. If for two vertices u and v, we drew two directed edges (u, v) and (v, u), we remove one arbitrary.
3. If a vertex has more than one incoming edge, we remove its outgoing edge.
4. If a vertex v has more than one incoming edge, we merge it with all its neighbors pointing to v and remove the incoming directed edges of the neighbors of v.
5. We remove each edge with probability $2/3$.
6. We merge each directed isolated edge.

Next we show that this procedure reduces the number of vertices by a constant factor. For readability, we do not optimize this constant. Note that in Line 1 we are adding n edges. It is easy to see that there is no cycle of length larger than 2 in the directed graph constructed in Line 1. Line 2 removes at most half of the edges. Moreover, it removes all cycles of length 2. Thus by the end of Line 2 we have a rooted forest with at least $n/2$ edges.

After Line 3 every vertex with indegree more than 1 has no outgoing edges. Recall that each vertex has at most one outgoing edge. Thus, after Line 3 we have a collection of rooted trees where only the root may have degree more than 2. We call such trees *long tail stars*. Note that if we remove the outgoing edge of a vertex v there are two incoming edges pointing to v (which uniquely correspond to v). Although the process of Line 3 may cascade and remove the incoming edges of v, the following simple double counting argument bounds the number of removed edges. Note that this argument is just to bound the number of the edges and we do not require to run it, in order to execute our algorithm.

We put a token on each directed edge of the forest (before running Line 3). Next we are going to move the tokens such that (a) we never have more than two tokens on each edge, and (b) at the end we move all tokens to the edges that survive after Line 3. This says that at least half of the edges (i.e., at least $n/4$ edges) survive Line 3.

We traverse over each rooted tree from the root to the leaves. At each step, if the outgoing edge of a vertex v is removed, by induction hypothesis there are at most two tokens on the edge. Also, v has at least two incoming edges. We move each of the tokens on the outgoing edge of v to one of its incoming edges. Note that this is the only time we move a token to the incoming edges of v and hence we do not have more than two tokens on each edge as desired.

If we merge a vertex v with r incoming edges in Line 4, we remove at most $2r$ directed edges (r incoming edges of v and at most one incoming edge per each neighbor of v. On the other hand, we decrease the number of vertices by r. Thus, if this stage removes more than $n/8$ edges the number of vertices drops to at most $n - \frac{n}{16} = \frac{15}{16}n$, as desired. To complete the proof, we assume that at most $n/8$ edges are removed in Line 4 and show that in this case Lines 5 and 6 decrease the number of vertices by a constant factor.

Note that Line 4 removes the root of all long tailed stars. Thus after Line 4 we have a collection of directed edges. The probability that an edge passed to Line 5 becomes an isolated edge after sampling is at least $\frac{2}{3} \cdot \frac{1}{3} \cdot \frac{2}{3} = \frac{4}{27}$. If we mark every third edge (starting from an end of each path), the chance that each marked edge becomes an isolated edge after sampling is independent of other marked edges. There are $\frac{1}{3} \cdot \frac{n}{8} = \frac{n}{24}$ marked edges. Let X be a random variable that indicates the number of marked edges that are isolated after sampling. Note that $\mathbf{E}[X] \geq \frac{4}{27}\frac{n}{24} = \frac{n}{162}$. By applying a simple Chernoff bound we have

$$\mathbf{Pr}\left[X \leq 0.5\frac{n}{162}\right] \leq \exp\left(-\frac{0.5^2\frac{n}{162}}{2}\right) = \exp\left(-\frac{n}{1296}\right).$$

Therefore, with high probability we merge at least $\frac{n}{324}$ edges in Line 6 as desired. $\square$

Part II

Sublinear-Time Algorithms

Chapter 6

Sublinear Algorithms for Matching & Vertex Cover

In this chapter, we study algorithms that run in time sublinear in the input size. That is, the algorithm is not even able to read the whole data. To achieve such algorithms, it is important to specify how the input is presented. For graph problems—the focus of this thesis—two models have been commonly considered in the literature:

- **The adjacency list model:** In this model, for any vertex v of its choice, the algorithm may query the degree of v in the graph and, for any $1 \leq i \leq \deg(v)$, may query the i-th neighbor of v stored in an arbitrarily ordered list.
- **The adjacency matrix model:** In this model, the algorithm may query, for any vertex-pair (u, v) of its choice, whether or not u and v are adjacent in the graph.

We consider both models in this chapter.

Our focus is particularly on algorithms that estimate the size of maximum matching or minimum vertex cover in general graphs. To be consistent with the conventions of the literature of sublinear time algorithms, for $\alpha \geq 1$ and $0 \leq \varepsilon \leq 1$, we say in this chapter that an estimate $\widetilde{\mu}(G)$ for the maximum matching size $\mu(G)$ and an estimate $\widetilde{\nu}(G)$ for the minimum vertex cover size $\nu(G)$ provide "multiplicative-additive" $(\alpha, \varepsilon n)$-approximations if

$$\frac{\mu(G)}{\alpha} - \varepsilon n \leq \widetilde{\mu}(G) \leq \mu(G) \qquad \text{and} \qquad \nu(G) \leq \widetilde{\nu}(G) \leq \alpha\nu(G) + \varepsilon n.$$

A standard multiplicative α-approximation is also essentially a $(\alpha, 0)$-approximation.

It is well-known that if M is a *maximal* matching, then the number of edges of M 2-approximates $\mu(G)$ and the number of vertices of M 2-approximates $\nu(G)$. But the greedy algorithm for maximal matching takes linear time in the input size to find. Can we achieve the same in sublinear time?

Although at the first glance it may seem impossible to do much without reading the whole input, numerous sublinear-time algorithms have been designed over the years for various optimization problems. In addition to matching and vertex cover, which have been studied extensively in the area [139, 133, 154, 136, 107, 69], the list includes estimating the weight/size of minimum spanning tree (MST) [67, 75], traveling salesman problem (TSP) [69], k-nearest neighbor graph [77], graph's average degree [87, 97], as well as problems such as vertex coloring [16], metric linear sampling [83], and many others. (This is by no means a comprehensive list of all the prior works.) For some of the classic results of the area, see the excellent survey of Czumaj and Sohler [76].

The randomized greedy maximal matching (RGMM) algorithm of Chapter 3, and in particular its local simulation oracle discussed in Section 3.2 is the basis of one of the most successful approaches for estimating the size of matching and vertex cover in sublinear time. To utilize this local simulation, the most common approach is to plug it into the framework of Parnas and Ron [139]: Pick a number of random vertices in the graph, simulate the local RGMM on them, and report the fraction of them that are matched as an estimate for the fraction of vertices matched in the whole graph. The final time-complexity of the algorithm, therefore, depends on the size of the local neighborhood that one has to explore for each randomly chosen vertex, which is also known as the "average query-complexity" of RGMM [133, 154, 136].

Indeed, our main contribution in this chapter is to prove the near-tight bound of Theorem 3.2 for the average query-complexity of RGMM. Recall that Theorem 3.2 asserts that for a vertex chosen uniformly at random, the expected query complexity is $O(\bar{d} \cdot \log n)$.

6.1 Applications of Theorem 3.2

Theorem 3.2 leads to a number of sublinear-time algorithms for matching and vertex cover that are provably time-optimal up to logarithmic factors. We elaborate on these applications here and also mention some other applications of Theorem 3.2.

As before, in all the statements below we use n to denote the number of vertices, Δ to denote the maximum degree, and $\bar{d}$ to denote average degree.

Application 1 – multiplicative approximation in the adjacency list model:

Theorem 6.1. *For any $\varepsilon > 0$, there is an algorithm that with probability $1 - 1/\operatorname{poly}(n)$ reports a $(2+\varepsilon)$-approximation to the size of maximum matching and that of minimum vertex cover using $O(n) + \widetilde{O}(\Delta/\varepsilon^2)$ time and queries in the adjacency list model.*

Observe that $\Omega(n)$ queries are necessary even to distinguish an empty graph from one with just one edge. Thus, $\Omega(n)$ time and queries are necessary for any multiplicative approximation in this model, implying that Theorem 6.1 is nearly time-optimal.

Theorem 6.1, notably, gives the *first* multiplicative estimator in the literature that runs in $\widetilde{O}(n)$ time for all graphs. For a $(2+\varepsilon)$-approximation, in particular, no $o(n^2)$ time algorithm was known for general graphs prior to our work. Allowing a larger $O(1)$-approximation, a recent result of Kapralov, Mitrovic, Norouzi-Fard, and Tardos [108] with some work leads to a $O(n + \Delta^2/\bar{d})$ time algorithm which can take $\widetilde{\Omega}(n\sqrt{n})$ time.

Interestingly, under the rather mild assumptions that $\bar{d} = \Omega(1)$ (which, e.g., holds if there are no singleton vertices) and that (estimates of) $\bar{d}$ and Δ are given, the $\Omega(n)$ lower bound is avoidable and the algorithm of Theorem 6.1 actually runs in $\widetilde{O}(\Delta/\varepsilon^2)$ time.

Application 2 – multiplicative-additive approximation in the adjacency list model:

Theorem 6.2. *For any $\varepsilon > 0$, there is an algorithm that with probability $1 - 1/\operatorname{poly}(n)$ reports a $(2, \varepsilon n)$-approximation to the size of maximum matching and that of minimum vertex cover using $\widetilde{O}((\bar{d}+1)/\varepsilon^2)$ time and queries in the adjacency list query model.*

Theorem 6.2 (nearly) matches an $\Omega(\bar{d}+1)$ lower bound of Parnas and Ron [139] that holds for any $(O(1), \varepsilon n)$-approximation of maximum matching or minimum vertex cover in this model. This culminates a long line of work [139, 133, 154, 136, 108] on this problem that we overview next.

Parnas and Ron [139] were the first to give a multiplicative-additive approximation for matching and vertex cover. They showed how to obtain a $(2, \varepsilon n)$-approximation in $\Delta^{O(\log(\Delta/\varepsilon))}$ time by simulating a distributed local algorithm for each sampled vertex. A quasi-polynomial dependence on Δ, however, is unavoidable with this approach due to existing distributed lower bounds [117].

The next wave of results [133, 154, 136] were based on the RGMM algorithm. Yoshida *et al.* [154] built on the work of Nguyen and Onak [133] and, notably, gave the first algorithm with a polynomial-in-Δ time-complexity of $O(\Delta^4/\varepsilon^2)$ for a $(2, \varepsilon n)$-approximation. Onak *et al.* [136] later shaved off some of the Δ factors from the bound of [154]. Although a bound of $\widetilde{O}_\varepsilon(\Delta)$ was first claimed in [136], Chen, Kannan, and Khanna [69] discovered a subtlety with a claim of [136] that happens to be crucial for their final result. Nonetheless, as also observed by Chen *et al.* [69], the techniques developed in [136] combined with the average query-complexity analysis of [154] discussed above, still implies an improved bound of $\widetilde{O}((\bar{d} \cdot \Delta + 1)/\varepsilon^2) = \widetilde{O}(\Delta^2/\varepsilon^2)$ for a $(2, \varepsilon n)$-approximation.[1]

Observe that all the algorithms of the literature discussed above require some upper bound on the degrees to run in sublinear-time and can take up to $\Omega(n^2)$ time for general graphs. A more recent algorithm by Kapralov *et al.* [108] has a more desirable time-complexity of $O(\Delta/\varepsilon^2)$. However, it only obtains an $(O(1), \varepsilon)$-approximation and as pointed out by Chen, Kannan, and Khanna [69]:

> *"Unfortunately, the constant hidden in the $O(1)$ notation [of [108]] is very large, and efficiently obtaining a $(2, \varepsilon n)$-approximation to matching size remains an important open problem"* [69].

Theorem 6.2, in light of the lower bound of [139], strongly resolves this open problem.

Application 3 – multiplicative-additive approximation in the adjacency matrix model:

Theorem 6.3. *For any $\varepsilon > 0$, there is an algorithm that with probability $1 - 1/\operatorname{poly}(n)$ reports a $(2, \varepsilon n)$-approximation to the size of maximum matching and that of minimum vertex cover using $\widetilde{O}(n/\varepsilon^3)$ time and queries in the adjacency matrix query model.*

A similar bound to Theorem 6.3 was claimed in [136]. However, as discussed above, the proof of [136] had a subtlety. Chen *et al.* [69] proposed a fix, but their algorithm runs in $\widetilde{O}_\varepsilon(n\sqrt{n})$ time. Theorem 6.3 improves this latter bound by a factor of $\sqrt{n}$.

On the lower bound side, suppose that the graph is either a random perfect matching or is empty. It is not hard to see that distinguishing the two cases requires $\Omega(n)$ queries to the adjacency matrix. As such, $\Omega(n)$ queries are necessary for any multiplicative-additive approximation in the adjacency-matrix model, implying that Theorem 6.3 is nearly time-optimal. Note that distinguishing an empty graph from one that includes only one edge requires $\Omega(n^2)$ queries in the adjacency-matrix model. This implies that, unlike Theorem 6.1 for the adjacency-list model, no non-trivial multiplicative approximation can be obtained in the adjacency-matrix model.

[1]We note that while Theorem 6.2 can be seen as a fix to [136], our techniques are very different from the approach of [136]. Particularly, [136] did not claim the tight upper bound of Theorem 3.2 that we prove here, but rather claimed a weaker bound of $O(\bar{d} \cdot \rho)$ where ρ is the ratio of the maximum degree over the min degree. As a result, even if one manages to fix the proof of [136], one does not get the strong multiplicative approximation of Theorem 6.1.

To prove Theorem 6.3, in addition to Theorem 3.2, we give a new reduction from the adjacency matrix model to the adjacency list model that, unlike the reduction of [136], does not lead to parallel edges or self-loops. This is crucial for our result; see Section 6.5.

Other Applications – TSP: Chen *et al.* [69] gave sublinear time algorithms for estimating the size of graphic TSP and $(1, 2)$-TSP in the adjacency matrix model. Their algorithms utilize a matching size estimator as a subroutine. Plugging Theorem 6.3 into their framework implies that:

Corollary 6.4 (of using Theorem 6.3 in the algorithms of [69]). *There is an $\widetilde{O}(n)$-time randomized algorithm that estimates the cost of graphic TSP to within a factor of (27/14). There is also an $\widetilde{O}(n)$-time randomized algorithm that estimates the cost of $(1, 2)$-TSP to within a factor of* 1.625*.*

Instead of Theorem 6.3, Chen *et al.* [69] used their $\widetilde{O}(n\sqrt{n})$-time matching size estimator in their algorithms. As a result, their algorithms were slower by a factor of $\sqrt{n}$ than those in Corollary 6.4.

Other Applications – AMPC: The *adaptive massively parallel computations* (AMPC) model was first introduced by [42] and has been further studied by [47, 66]. While the precise definition of the AMPC model is out of the scope of this chapter, it augments the standard MPC model that we covered extensively at Part I with a "distributed hash table". Combined with the techniques developed for the maximal independent set problem in [42], Theorem 3.2 implies an $O(1)$-round AMPC algorithm for maximal matching using a strongly sublinear space of $O(n^\delta)$ per machine (for any constant $\delta > 0$) and an optimal total space of $\widetilde{O}(m)$ in m-edge graphs. This has to be compared with an $O(\log \log n)$-round algorithm presented in [47] that has the same space complexity.

The essence of the improved AMPC algorithm mentioned above is the surprising fact, implicit in Theorem 3.2, that if one starts the local simulation of RGMM from *every* vertex in the graph all in parallel, then the expected sum of the query-complexities of all these parallel calls, or equivalently the "total work" of the algorithm, is $n \cdot O(\bar{d} \log n) = O(m \log n)$, i.e., near-linear in the input size!

6.2 Our Techniques & Background on the Query-Complexity of RGMM

As discussed above, our main technical contribution is the upper bound of Theorem 3.2 on the average query-complexity of the randomized greedy maximal matching algorithm. In this section, we provide more context about this result, further compare it to the previous bounds, and give an overview of our techniques and new insights for proving it.

The first problem is that analyzing the expected query-complexity for a *given* edge or vertex seems to be challenging. In fact, obtaining a $\text{poly}(\Delta, \log n)$ bound for this problem remains a major open question in the study of local computation algorithms (LCA's) [5, 92]. In a beautiful paper, Yoshida, Yamamoto, and Ito [154] got around this challenge and proved a $\text{poly}(\Delta)$ upper bound on the *average* query-complexity with a global analysis. Namely, instead of bounding the expected number of queries that an edge e generates, they showed that for any edge $e = (u, v)$ the expected number of edges in the graph whose query process reaches e is at most $O(\deg(u) + \deg(v))$.[2] This way, the expected sum of all queries starting from all the edges in the graph, can be upper bounded by $\sum_{(u,v)\in E} O(\deg(u)+\deg(v)) = O(L)$, where L is the number of edges in the line-graph. This implies that for an *edge* chosen uniformly at random, the expected query-complexity can be upper bounded by $O(L/m)$. Although it is not immediate, we note that essentially the same proof also implies that for a *vertex* chosen uniformly at random, the expected query-complexity is $O(L/n)$. As we discussed in Section 3.2, L/n is $O(\bar{d} \cdot \Delta)$ and can also be as large as $\Omega(\bar{d} \cdot \Delta)$.

Our new analysis builds on some of the ideas introduced by Yoshida *et al.* [154]. Similar to their work and, crucially, instead of directly analyzing the number of recursive calls *out* of an edge or a vertex, we analyze the recursive calls made *to* an edge or a vertex. There are, however, several crucial differences between the two approaches that leads to our near-tight analysis in Theorem 3.2.

We prove that for any edge $e = (u, v)$ in the graph, the expected number of starting queries that reach e can be upper bounded by $O(\log n)$. This improves over the previous upper bound of $O(\deg(u) + \deg(v))$ by Yoshida *et al.* [154] and is the key component of our analysis.

Let us fix edge e and overview how we prove the bound of $O(\log n)$ on the expected number of starting points that query e. To prove this upper bound, if in a permutation π we happen to query e from q different starting points, we charge q other permutations $\pi_1, \ldots, \pi_q$.

[2]The analysis of [154] proceeds on the line-graph and $\deg(u) + \deg(v)$ is the degree of e in the line-graph.

Observe that by a simple double counting argument, the expected number of times that a random permutation $\pi \in \Pi$ is charged by all other permutations combined, is exactly equal to the expected number of starting points that query e in a random permutation π. As such, if we prove an upper bound of T on the number of times that each permutation is charged, then we get that e is queried by at most T starting points in expectation. Unfortunately, however, we remark that there will be permutations that get charged up to even $\Omega(n)$ times with our charging method.

To get past this hurdle, we draw a novel connection to an orthogonal line of work on bounding the *parallel depth* of RGMM (or more generally randomized greedy maximal independent set) pioneered by Blelloch, Fineman, and Shun [59] whose bounds were tightened by Fischer and Noever [89]. In particular, by carefully analyzing the structure of RGMM queries, we show that if a permutation π is charged b times with our charging method, then the parallel depth of GMM for this permutation must be at least $\Omega(b)$. Plugging the high probability upper bound of [89] that bounds the parallel depth of a random permutation by $O(\log n)$, guarantees that $1 - 1/\operatorname{poly}(n)$ fraction of the permutations are charged at most $O(\log n)$ times, which also suffices for proving Theorem 3.2.

6.3 Average Query-Complexity of RGMM

Recall from Section 3.2 that we use $T(v, \pi)$ to denote the number of recursive calls to the edge oracle $\mathsf{EO}(\cdot, \pi)$ over the course of answering $\mathsf{VO}(v, \pi)$. We note here again that for some edge e, $\mathsf{EO}(e, \pi)$ may be called multiple times during the execution of $\mathsf{VO}(v, \pi)$ and we count all of these in $T(v, \pi)$, though only the first call to $\mathsf{EO}(e, \pi)$ may generate new recursive calls as the rest of the calls use the cached value for $\mathsf{EO}(e, \pi)$.

In this section, we prove Theorem 3.2 discussed in Section 3.2 that is restated below:

Theorem 3.2 (restated). *For a vertex $v \sim V$ chosen uniformly at random and for a permutation π chosen independently and uniformly at random,*

$$\mathop{\mathbf{E}}_{v \sim V, \pi}[T(v, \pi)] = O(\bar{d} \cdot \log n),$$

where recall that $n := |V|$ and $\bar{d}$ is the average degree of the graph.

For any edge $e \in E$ and a vertex $v \in V$, we use $Q(e, v, \pi)$ to denote the number of times that $\mathsf{EO}(e, \pi)$ is called during the execution of $\mathsf{VO}(v, \pi)$, and we define $Q(e, \pi) := \sum_{v \in V} Q(e, v, \pi)$. The following immediately follows from the definition:

Observation 6.5. *For any $\pi \in \Pi$ and any $v \in V$, $T(v,\pi) = \sum_{e \in E} Q(e,v,\pi)$.*

The next bound holds because we cache query results as discussed in Section 3.2.

Observation 6.6. *For every edge $e = \{a,b\}$ and every $\pi \in \Pi$, $Q(e,\pi) \le O(n^2)$.*

Proof. Take an arbitrary vertex $v \in V$. Over the course of answering $\mathsf{VO}(v,\pi)$, we either call $\mathsf{EO}(e,\pi)$ directly by the vertex oracle (at most once) or during the execution of $\mathsf{EO}(f,\pi)$ for some edge f incident to e. The key observation is that $\mathsf{EO}(f,\pi)$ generates new recursive calls only the first time that it is called (due to caching). Hence, in total $\mathsf{EO}(e,\pi)$ is called at most $(\deg_G(a) - 1) + (\deg_G(b) - 1) + 1 \le 2n - 1$ times while answering $\mathsf{VO}(v,\pi)$. This means $Q(e,v,\pi) \le 2n-1$. Since $Q(e,\pi) = \sum_v Q(e,v,\pi)$, we get $Q(e,\pi) \le \sum_v (2n-1) = n(2n-1) = O(n^2)$. □

The main technical part is the proof of Lemma 6.7 below. Intuitively, the lemma states that if we run the vertex oracle on every vertex $v \in V$ all in parallel (in a way that the cashed values stored for one parallel call are not used for another), then in expectation over π, the total number of times that edge oracle $\mathsf{EO}(e,\pi)$ is called can be bounded by $O(\log n)$.

Lemma 6.7. *For any edge e, $\mathbf{E}_\pi[Q(e,\pi)] = O(\log n)$.*

Lemma 6.7 easily implies Theorem 3.2 as described next.

Proof of Theorem 3.2 via Lemma 6.7. It holds that

$$\mathop{\mathbf{E}}_{\pi}\left[\sum_{v \in V} T(v,\pi)\right] \overset{\text{Obs 6.5}}{=} \mathop{\mathbf{E}}_{\pi}\left[\sum_{v \in V}\sum_{e \in E} Q(e,v,\pi)\right] = \mathop{\mathbf{E}}_{\pi}\left[\sum_{e \in E}\sum_{v \in V} Q(e,v,\pi)\right] = \mathop{\mathbf{E}}_{\pi}\left[\sum_{e \in E} Q(e,\pi)\right]$$
$$= \sum_{e \in E} \mathop{\mathbf{E}}_{\pi}[Q(e,\pi)] \overset{\text{Lemma 6.7}}{=} \sum_{e \in E} O(\log n) = O(m \log n). \tag{6.1}$$

Therefore, for a vertex $v \in V$ that is chosen uniformly at random from V, we get

$$\mathop{\mathbf{E}}_{v,\pi}[T(v,\pi)] = \frac{1}{n} \cdot \mathop{\mathbf{E}}_{\pi}\left[\sum_{v \in V} T(v,\pi)\right] \overset{(6.1)}{=} \frac{1}{n} \cdot O(m \log n) = O(\bar{d} \cdot \log n). \quad \square$$

The rest of this section is devoted to proving Lemma 6.7.

Query paths: Let $S_{v,\pi}$ be the stack of recursive calls to the edge oracle during the execution of $\mathsf{VO}(v,\pi)$. Namely, whenever the edge oracle $\mathsf{EO}(e,\pi)$ is called on some edge e we push e to the stack, and pop e when the value of $\mathsf{EO}(e,\pi)$ is determined. Let $P = (e_1, \dots, e_k)$ be the ordered set of edges inside $S_{v,\pi}$ at some point during the execution of $\mathsf{VO}(v,\pi)$ with e_k

being the last edge pushed into the stack. Because our vertex and edge oracles both process the neighboring edges greedily in the increasing order of their ranks, it is easy to verify that P must be a path in graph G with v being one of its endpoints (particularly $v \in e_1$). Now if we make the edges in P directed such that $\vec{P} = (\vec{e_1}, \dots, \vec{e_k})$ is a directed path starting from v, we call $\vec{P}$ a (v, π)-query-path. With this definition, $T(v, \pi)$ is essentially the total number of (v, π)-query-paths.

Let us take an edge $e = \{a, b\} \in E$ and let $\vec{e} = (a, b)$ be the same edge made directed from a to b. We define $\mathcal{Q}(\vec{e}, v, \pi)$ to be the set of all (v, π)-query-paths that end at $\vec{e}$ (in this precise direction) and define $\mathcal{Q}(\vec{e}, \pi) := \bigcup_{v \in V} \mathcal{Q}(\vec{e}, v, \pi)$. Furthermore, we define $Q(\vec{e}, v, \pi) := |\mathcal{Q}(\vec{e}, v, \pi)|$ and define $Q(\vec{e}, \pi) := \sum_{v \in V} Q(\vec{e}, v, \pi) = |\mathcal{Q}(\vec{e}, \pi)|$ (the latter equality follows since $\mathcal{Q}(\vec{e}, v, \pi)$ and $\mathcal{Q}(\vec{e}, u, \pi)$ are disjoint for $u \neq v$).

Observation 6.8. *Let $e = \{a, b\}$, $\vec{e} = (a, b)$, and $\overleftarrow{e} = (b, a)$. We have $Q(e, \pi) = Q(\vec{e}, \pi) + Q(\overleftarrow{e}, \pi)$.*

Proof. Recall from the definition that $Q(e, v, \pi)$ is the number of times that edge oracle $\mathsf{EO}(e, \pi)$ is called during the execution of $\mathsf{VO}(v, \pi)$. Every time that we call $\mathsf{EO}(e, \pi)$, we add e to the stack $S_{v,\pi}$ and thus get exactly one (v, π)-query-path that ends at e either in the direction of $\vec{e}$ or $\overleftarrow{e}$. This implies that $Q(e, v, \pi) = Q(\vec{e}, v, \pi) + Q(\overleftarrow{e}, v, \pi)$. The observation follows since by definition $Q(\vec{e}, \pi) = \sum_v Q(\vec{e}, v, \pi)$, $Q(e, \pi) = \sum_v Q(e, v, \pi)$, and $Q(\overleftarrow{e}, \pi) = \sum_v Q(\overleftarrow{e}, v, \pi)$. $\square$

In what follows, we prove the following lemma which easily implies Lemma 6.7.

Lemma 6.9. *For any arbitrarily directed edge $\vec{e} = (a, b)$, $\mathbf{E}_\pi[Q(\vec{e}, \pi)] = O(\log n)$.*

Proof of Lemma 6.7 via Lemma 6.9. By Observation 6.8, Lemma 6.9 implies Lemma 6.7 since for any edge $e = \{u, v\}$, $\mathbf{E}_\pi[Q(e, \pi)] = \mathbf{E}_\pi[Q(\vec{e}, \pi)] + \mathbf{E}_\pi[Q(\overleftarrow{e}, \pi)] = O(\log n) + O(\log n) = O(\log n)$. $\square$

We now turn to prove Lemma 6.9 for edge $\vec{e}$ that is fixed for the rest of the proof.

First, given a permutation $\pi \in \Pi$ and a directed path $\vec{P} = (\vec{e_1}, \dots, \vec{e_k})$, we define $\phi(\pi, \vec{P}) \in \Pi$ to be another permutation $\sigma \in \Pi$ constructed as:

$$(\sigma(e_1), \dots, \sigma(e_{k-1}), \sigma(e_k)) := (\pi(e_2), \dots, \pi(e_k), \pi(e_1)), \text{ and}$$
$$\sigma(e') := \pi(e') \qquad \forall e' \notin \vec{P}.$$

That is, $\phi(\pi, \vec{P})$ is obtained by rotating the ranks of π along $\vec{P}$ in the reverse direction.

Now we construct a bipartite graph $H = H(\vec{e})$ with two parts A and B such that $|A| = |B| = |\Pi|$. Each permutation $\pi \in \Pi$ over the edge-set of G has one corresponding vertex π_A in A and one corresponding vertex π_B in B. Furthermore, for any permutation $\pi \in \Pi$ and any query-path $\vec{P} = (\vec{e_1}, \dots, \vec{e_k} = \vec{e}) \in \mathcal{Q}(\vec{e}, \pi)$, we connect vertex $\pi_A \in A$ to vertex $\sigma_B \in B$ corresponding to permutation $\sigma := \phi(\pi, \vec{P})$.

Graph H is particularly constructed such that for each permutation $\pi \in \Pi$ the degree of vertex $\pi_A \in A$ equals $Q(\vec{e}, \pi)$. Namely:

Observation 6.10. *For any permutation $\pi \in \Pi$, $\deg_H(\pi_A) = Q(\vec{e}, \pi)$.*

Proof. By construction there is a one-to-one mapping between the edges of π_A and query paths $\vec{P} \in \mathcal{Q}(\vec{e}, \pi)$. Thus $\deg_H(\pi_A) = |\mathcal{Q}(\vec{e}, \pi)| = Q(\vec{e}, \pi)$. □

Therefore to prove Lemma 6.9 that $\mathbf{E}_\pi[Q(\vec{e}, \pi)] = O(\log n)$, it suffices to bound the average degree of graph H by $O(\log n)$. In other words, it suffices to show that for a vertex $\pi_A \in A$ chosen uniformly at random, $\mathbf{E}_{\pi_A \sim A}[\deg_H(\pi_A)] = O(\log n)$. This is our plan for the rest of the proof.

For some large enough constant $c \geq 1$ and parameter $\beta = c \log n$, we partition Π into two subsets of *likely* permutations L and *unlikely* permutations U as follows:

$$L := \left\{ \pi \in \Pi \;\middle|\; \max_{P \in \mathcal{Q}(\vec{e}, \pi)} |P| \leq \beta \right\}, \qquad U := \Pi \setminus L. \tag{6.2}$$

In words, if all query-paths in $\mathcal{Q}(\vec{e}, \pi)$ have length $\leq \beta$ then $\pi \in L$, and otherwise $\pi \in U$. We use A_L (resp. A_U) to denote the set of vertices in part A of graph H that correspond to permutations in L (resp. U).

We use the next two lemmas to bound the number of edges connected to A_U and A_L respectively.

Lemma 6.11. *Any vertex $\sigma_B \in B$ has at most β neighbors in A_L.*

Lemma 6.12. *If constant c is large enough, $|A_U| \leq m!/n^2$.*

Let us first see how Lemmas 6.11 and 6.12 prove Lemma 6.9 (and thus Theorem 3.2 as discussed before). We then turn to prove these two claims.

Proof of Lemma 6.9 via Lemmas 6.11 and 6.12. We first upper bound the size of the edge-set $E(H)$ of H by $O(m! \log n)$. Since each vertex $\pi_A \in A$ has degree $O(n^2)$ by Observation 6.6, and that by Lemma 6.12, $|A_U| \le m!/n^2$, the number of edges connected to A_U is $\frac{m!}{n^2} \cdot O(n^2) = O(m!)$. Moreover, by Lemma 6.11, each vertex $\sigma_B \in B$ has at most β neighbors in A_L. Since H is bipartite and every edge of A_L goes to B, the total number of edges connected to A_L can be upper bounded by $|B| \cdot \beta = m! \cdot c \log n = O(m! \log n)$. Since A_L and A_U partition A and that every edge of H has one endpoint in A, we get $|E(H)| = O(m! \log n) + O(m!) = O(m! \log n)$.

Now if we pick a vertex π_A from A, it has expected degree $\frac{|E(H)|}{|A|} = \frac{O(m! \log n)}{m!} = O(\log n)$. By Observation 6.10, this implies $\mathbf{E}_{\pi \in \Pi}[Q(\vec{e}, \pi)] = \mathbf{E}_{\pi_A \sim A}[\deg_H(\pi_A)] = O(\log n)$. □

6.3.1 Proof of Lemma 6.11

We prove the following statement which we show suffices to prove Lemma 6.11.

Claim 6.13. *Let $\pi, \pi' \in \Pi$ with $\pi \neq \pi'$, let $\vec{P}$ and $\vec{P}'$ respectively be (v, π)- and (v', π')-query-paths both ending at $\vec{e}$ for some $v, v' \in V$, and suppose that $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P}') = \sigma$. Then $|\vec{P}| \neq |\vec{P}'|$.*

Let us first see how Claim 6.13 suffices to prove Lemma 6.11:

Proof of Lemma 6.11 via Claim 6.13. Let us take an arbitrary vertex σ_B in part B of graph H. For any edge $\{\pi_A, \sigma_B\}$ in H, by construction of H there must be a unique (v, π)-query-path $\vec{P}$ such that $\phi(\pi, \vec{P}) = \sigma$ where here $\pi, \sigma \in \Pi$ are the permutations corresponding to π_A and σ_B respectively. Let us define the label $\chi(\pi_A, \sigma_B)$ of edge $\{\pi_A, \sigma_B\}$ to be the length $|\vec{P}|$ of this query-path $\vec{P}$. Claim 6.13 essentially implies that all the edges of σ_B receive different labels. On the other hand, if π_A is a neighbor of σ_B and $\pi_A \in A_L$, then by definition (6.2) of set L, $\chi(\pi_A, \sigma_B) \le \beta$. The uniqueness of the integer labels and the upper bound of β imply together that σ_B can have at most β neighbors in A_L. The proof of Lemma 6.11 is thus complete. □

We now turn to prove Claim 6.13.

Let $\pi, \pi', \vec{P}, \vec{P}', v, v', \sigma$ be as defined in Claim 6.13 and assume for contradiction that $|\vec{P}| = |\vec{P}'|$. First observe that if in addition to their lengths, the edges traversed by the two paths $\vec{P}, \vec{P}'$ are also the same i.e., $\vec{P} = \vec{P}'$, then by definition of the mapping ϕ, we have $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P}')$ iff $\pi = \pi'$ which contradicts the assumption $\pi \neq \pi'$ of Claim 6.13. So let us assume that $\vec{P} \neq \vec{P}'$ but $|\vec{P}| = |\vec{P}'|$. This implies that the two paths must "branch" at least once.

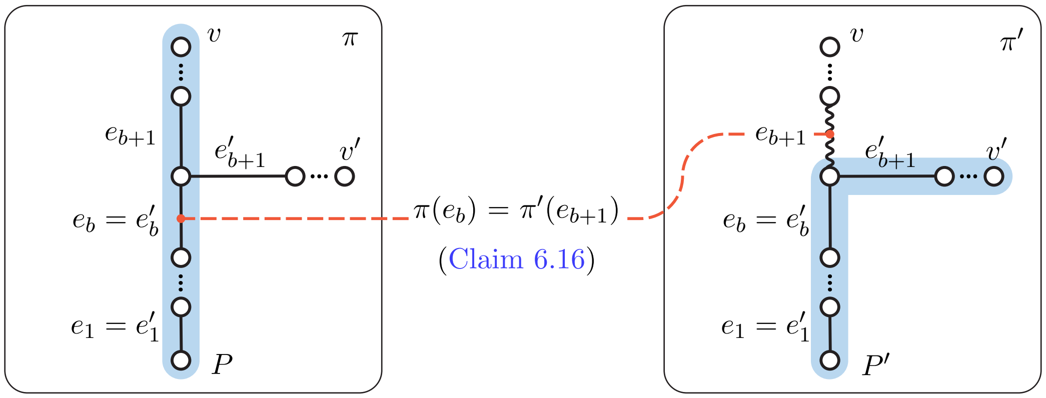


Figure 6.1: On the right hand side, we have permutation π' and query-path P' is highlighted. On the left hand side, we have permutation π and path P is highlighted. Our arguments essentially show that e_{b+1} must belong to matching $\mathsf{GMM}(G, \pi')$. We also show that $\pi'(e_{b+1}) < \pi'(e'_b)$. Therefore, $\mathsf{EO}(e'_{b+1}, \pi')$ should terminate (returning FALSE) before calling $\mathsf{EO}(e'_b, \pi')$. This means P' cannot be a valid query-path in π' and this is our desired contradiction which proves Claim 6.13.

It would be convenient to define $\overleftarrow{P} = (\overleftarrow{e}_1, \overleftarrow{e}_2, \dots, \overleftarrow{e}_k)$ and $\overleftarrow{P}' = (\overleftarrow{e}'_1, \overleftarrow{e}'_2, \dots, \overleftarrow{e}'_k)$ to be respectively the same as $\vec{P} = (\vec{e_k}, \dots, \vec{e_1})$ and $\vec{P}' = (\vec{e_k}', \dots, \vec{e_1}')$ except that their edges are traversed in the reverse direction. Since both $\vec{P}$ and $\vec{P}'$ end at $\vec{e}$ (by definition of query-paths), both $\overleftarrow{P}$ and $\overleftarrow{P}'$ must start from $\overleftarrow{e} = \overleftarrow{e}_1 = \overleftarrow{e}'_1$. Let $(b+1)$ be the smallest index where $\overleftarrow{e}_{b+1} \neq \overleftarrow{e}'_{b+1}$. That is, we have $e_1 = e'_1, \dots, e_b = e'_b$ and $e_{b+1} \neq e'_{b+1}$ (see Figure 6.1). Observe that $b + 1 \geq 2$ since $\overleftarrow{e}_1 = \overleftarrow{e}'_1 = \overleftarrow{e}$.

Let us make the following assumption that comes without loss of generality as we have not distinguished π and π' in any other way up to this point of the analysis.

Assumption 6.14. $\pi(e_b) \leq \pi'(e'_b)$.

Observation 6.15. *It holds that* $\pi(e_1) < \pi(e_2) < \dots < \pi(e_k)$ *and* $\pi'(e'_1) < \pi'(e'_2) < \dots < \pi'(e'_k)$.

Proof. This holds because $\vec{P}$ is a query-path in π and $\vec{P}'$ is a query-path in π'. Recall that query-paths correspond to recursions in the stack and Algorithm 1 only recurses on edges with lower rank than the current edge. Hence the edges along a query path must be decreasing in rank. □

Claim 6.16. *For any* $i \in \{b+1, \dots, k\}$, $\pi'(e_i) = \pi(e_{i-1})$ *and* $\pi(e'_i) = \pi'(e'_{i-1})$.

Proof. First, since $i \geq b+1 \geq 2$, we have $e_i \neq e_1$. Therefore by definition of $\phi(\pi, \vec{P})$, which rotates the ranks of π in the reverse direction of $\vec{P}$ (i.e., in direction of $\overleftarrow{P}$), $\phi(\pi, \vec{P})(e_i) = \pi(e_{i-1})$. On the other hand, note that $e_i \notin P'$ since $i \geq b+1$ and P and P' branch after edge $e_b = e'_b$ — this implies that $\phi(\pi', \vec{P'})(e_i) = \pi'(e_i)$. Combined with our assumption that $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P'}) = \sigma$, this implies the desired equality $\pi'(e_i) = \pi(e_{i-1})$.

The second equality follows from essentially the same argument; we still state it for completeness. We have $\phi(\pi', \vec{P'})(e'_i) = \pi'(e'_{i-1})$ since $e_i \neq e_1$. Also, since $e'_i \notin P$ for $i \geq b+1$, we have $\phi(\pi, \vec{P})(e'_i) = \pi(e'_i)$. Thus to have $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P'}) = \sigma$ it is necessary that $\pi'(e'_{i-1}) = \pi(e'_i)$. □

Claim 6.17. $\pi'(e_{b+1}) < \pi'(e'_b)$.

Proof. From Claim 6.16 we know $\pi'(e_{b+1}) = \pi(e_b)$. Combined with Assumption 6.14 that $\pi(e_b) \leq \pi'(e'_b)$, this implies $\pi'(e_{b+1}) \leq \pi'(e'_b)$. The equality can be ruled out since π' is a permutation and distinct edges e_{b+1} and e'_b cannot be assigned the same rank. As such, $\pi'(e_{b+1}) < \pi'(e'_b)$. □

Claim 6.18. *If $\pi(f) \neq \pi'(f)$ for some edge f, then $\pi(f) \geq \pi(e_b)$ and $\pi'(f) \geq \pi(e_b)$. In other words, the two permutations π and π' are identical on all ranks smaller than $\pi(e_b)$.*

Proof. If $f \notin P \cup P'$, then clearly $\pi'(f) = \pi(f) = \sigma(f)$ since $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P'}) = \sigma$ and ϕ only changes the ranks of edges along the query-path that it is given. So we can assume $f \in P \cup P'$; there are therefore four possible scenarios:

Case (1) $f \in \{e_1, \ldots, e_{b-1}\}$: For any $i \in \{2, \ldots, b\}$, to have $\phi(\pi, \vec{P})(e_i) = \phi(\pi', \vec{P'})(e_i)$, it is necessary that $\pi(e_{i-1}) = \pi'(e_{i-1})$. Thus, in this case $\pi(f) = \pi'(f)$.

Case (2) $f = e_b$: We have $\pi(f) = \pi(e_b)$ since $f = e_b$ and we have $\pi'(f) = \pi'(e_b) \geq \pi(e_b)$ by Assumption 6.14. So the claim holds in this case.

Case (3) $f \in \{e_{b+1}, \ldots, e_k\}$: We have $\pi(f) > \pi(e_b)$ in this case by Observation 6.15. Furthermore, by Claim 6.16, $\pi'(e_i) = \pi(e_{i-1})$ for $i \geq b+1$; thus $\pi'(f) \in \{\pi(e_b), \ldots, \pi(e_{k-1})\}$ in this case. By Observation 6.15, $\min\{\pi(e_b), \ldots, \pi(e_{k-1})\} = \pi(e_b)$ and thus $\pi'(f) \geq \pi(e_b)$. So the claim holds in this case.

Case (4) $f \in \{e'_{b+1}, \ldots, e'_k\}$: The proof of this case is similar to case (3). We have $\pi'(f) > \pi'(e_b) \geq \pi(e_b)$ by Observation 6.15 and Assumption 6.14. On the other hand, from Claim 6.16 we get $\pi(e'_i) = \pi'(e'_{i-1})$ for any $i \geq b+1$, which implies $\pi(f) \in \{\pi'(e'_b), \ldots, \pi'(e'_{k-1})\}$. By Observation 6.15 $\min\{\pi'(e'_b), \ldots, \pi'(e'_{k-1})\} = \pi'(e'_b)$ and thus

$\pi(f) \geq \pi'(e'_b)$. With $\pi'(e'_b) \geq \pi(e_b)$ of Assumption 6.14, we get $\pi(f) \geq \pi(e_b)$. So the claim holds in this case too and the proof is complete. □

Claim 6.19. $e_{b+1} \in \mathsf{GMM}(G, \pi')$.

Proof. Assume for the sake of contradiction that $e_{b+1} \notin \mathsf{GMM}(G, \pi')$. This means that e_{b+1} must be incident to an edge $f \in \mathsf{GMM}(G, \pi')$ with $\pi'(f) < \pi'(e_{b+1}) = \pi(e_b)$ (where recall the last equality follows from Claim 6.16). Since π and π' are identical for ranks smaller than $\pi(e_b)$ by Claim 6.18, $f \in \mathsf{GMM}(G, \pi')$ implies $f \in \mathsf{GMM}(G, \pi)$ as well. Using this and combined with $\pi(f) = \pi'(f) < \pi(e_b)$, we show that $\vec{P}$ is not a valid query-path in permutation π which is a contradiction. To see this, note that while running the edge oracle $\mathsf{EO}(e_{b+1}, \pi)$, we should call $\mathsf{EO}(f, \pi)$ before $\mathsf{EO}(e_b, \pi)$ because $\pi(f) < \pi(e_b)$; moreover, since $f \in \mathsf{GMM}(G, \pi)$, $\mathsf{EO}(e_{b+1}, \pi)$ will immediately terminate and return FALSE without calling $\mathsf{EO}(e_b, \pi)$. □

We are now ready to finalize the proof of Claim 6.13 by showing that the assumptions above lead to a contradiction. The contradiction that we prove is that $\vec{P'}$ cannot be a valid query-path in permutation π'. To see this, recall that during the execution of $\mathsf{EO}(e'_{b+1}, \pi')$, we have to call $\mathsf{EO}(e_{b+1}, \pi')$ before $\mathsf{EO}(e'_b, \pi')$ since by Claim 6.17 $\pi'(e_{b+1}) < \pi'(e'_b)$. On the other hand, by Claim 6.19 the answer to $\mathsf{EO}(e_{b+1}, \pi')$ is TRUE and so $\mathsf{EO}(e'_{b+1}, \pi')$ should terminate immediately and return FALSE without calling $\mathsf{EO}(e'_b, \pi')$. Therefore, $\vec{P'}$ is not a valid query-path in π' contradicting our assumption that $|\vec{P}| = |\vec{P'}|$ is possible. The proof of Claim 6.13 is thus complete. As discussed at the start of Section 6.3.1, this also completes the proof of Lemma 6.11.

6.3.2 Proof of Lemma 6.12

To prove Lemma 6.12 we first recall a parallel implementation of the randomized greedy maximal matching algorithm and the bounds known for its *round-complexity*. For more details see [59, 89].

Parallel Randomized Greedy MM: Given a graph G and a permutation π over its edge-set, we repeat the following until G becomes empty: in parallel add any "local minimum" edge e to the matching and remove its endpoints from the graph. An edge is local minimum if its rank is smaller than that of all of its neighboring edges that remain in the graph.

It can be easily confirmed that the output of the algorithm above is exactly $\mathsf{GMM}(G, \pi)$. We use $\rho(G, \pi)$ to denote the round-complexity of the algorithm, i.e., the number of iterations that it takes until the graph becomes empty.

It was shown in [59] that $\rho(G, \pi)$ can be bounded for a random permutation by $O(\log^2 n)$ with high probability. This was improved to $O(\log n)$ in [89]:

Lemma 6.20 ([89])**.** *Let π be a permutation chosen uniformly at random over the edge-set of an n-vertex graph G. With probability at least $1 - n^{-2}$, $\rho(G, \pi) = O(\log n)$.*

We prove the following:

Claim 6.21. *Let P be any query-path in G for permutation π, then $\rho(G, \pi) \geq \lfloor \frac{|P|}{2} \rfloor$.*

Claim 6.21 suffices to prove Lemma 6.12 as proved next.

Proof of Lemma 6.12 via Claim 6.21. For any permutation $\pi \in U$, by definition (6.2) there is at least one query-path of length at least $\beta+1$ in permutation π. This implies by Claim 6.21 that $\rho(G, \pi) \geq \lfloor \frac{\beta+1}{2} \rfloor$ for any $\pi \in U$. If we set the constant c in $\beta = c \log n$ to be sufficiently large, we can use Lemma 6.20 to bound the probability of this event by $\leq 1/n^2$. Thus, $|U|/|\Pi| \leq 1/n^2$ which implies $|U| \leq |\Pi|/n^2 = m!/n^2$. The lemma follows noting that $|A_U| = |U|$ due to the one-to-one correspondence between vertices A_U and permutations in U that we discussed in Section 6.3.1. □

Proof of Claim 6.21. Let $P = (e_k, \ldots, e_1)$ be our query-path and recall from Observation 6.15 that $\pi(e_k) > \ldots > \pi(e_1)$. For any edge e, we use $\rho(e)$ to denote the round of the parallel implementation in which edge e gets removed from the graph according to permutation π. We show that for any $i \in \{2, \ldots, k-1\}$, $\rho(e_i) \geq \rho(e_{i-2}) + 1$. By a simple induction, this implies $\rho(e_{k-1}) \geq \lfloor k/2 \rfloor$ and so $\rho(G, \pi) \geq \lfloor k/2 \rfloor$.

Suppose for the sake of contradiction that $\rho(e_i) < \rho(e_{i-2}) + 1$ (or equivalently $\rho(e_i) \leq \rho(e_{i-2})$) for some $2 \leq i \leq k-1$. This means that at the start of round $\rho(e_i)$ both e_i and e_{i-2} are still in the graph. Observe that if $\rho(e_{i-1}) < \rho(e_i)$, then during round $\rho(e_{i-1})$ at least one of the endpoints of e_{i-1} must be matched and so either e_i or e_{i-2} (or both) should also be removed from the graph which contradicts $\rho(e_{i-1}) < \rho(e_i) \leq \rho(e_{i-2})$. Therefore $\rho(e_{i-1}) \geq \rho(e_i)$ and so e_{i-1} is also still in the graph along with e_i and e_{i-2} at the start of round $\rho(e_i)$. This means that e_i is not a local minimum during round $\rho(e_i)$ and so it is removed in this round because some other edge $f \neq e_i$ joins the matching. Note also that $f \neq e_{i-1}$ since e_{i-1} is incident to e_{i-2} and is not a local minimum.

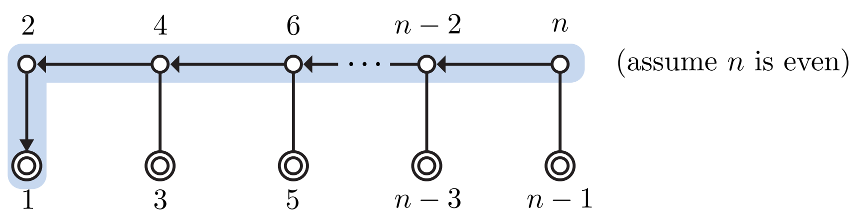


Figure 6.2: In this permutation, all the vertices with odd ranks join the MIS in round one and the whole graph becomes empty immediately, hence the parallel depth is 1. However, the query process for the vertex with rank n first goes through all even nodes, thus has length $\geq n/2$.

Let x be the vertex incident to both e_{i-1} and e_i and let y be the other endpoint of e_i incident to e_{i+1} (note that since $i \leq k-1$ edge e_{i+1} should exist). Since f is a neighbor of e_i, it is either connected to x or y. We show that both cases lead to contradictions.

Case (1) f connected to y: Noting that $\pi(f) < \pi(e_i)$ since $\pi(f)$ is a local minimum when e_i still is in the graph, we get that $\mathsf{EO}(e_{i+1}, \pi)$ calls $\mathsf{EO}(f, \pi)$ before $\mathsf{EO}(e_i, \pi)$. But because $f \in \mathsf{GMM}(G, \pi)$, $\mathsf{EO}(e_{i+1}, \pi)$ would not continue to call $\mathsf{EO}(e_i, \pi)$ and P cannot be a valid query-path.

Case (2) f connected to x: Since f is a local minimum when e_{i-1} still exists in the graph, $\pi(f) < \pi(e_{i-1})$. This implies that $\mathsf{EO}(e_i, \pi)$ should call $\mathsf{EO}(f, \pi)$ before $\mathsf{EO}(e_{i-1}, \pi)$. But because $f \in \mathsf{GMM}(G, \pi)$, $\mathsf{EO}(e_i, \pi)$ would not continue to call $\mathsf{EO}(e_{i-1}, \pi)$ and so, again, P cannot be a valid query-path.

The contradictions above rule out the possibility of our assumption that $\rho(e_i) < \rho(e_{i-2}) + 1$ and so we indeed get $\rho(e_i) \geq \rho(e_{i-2}) + 1$ for all $2 \leq i \leq k-1$. As discussed, this implies $\rho(G, \pi) \geq \lfloor k/2 \rfloor$ completing the proof. □

Remark 6.22. *Claim 6.21 shows that for any permutation π, the maximum query length in the random greedy maximal matching algorithm is asymptotically upper bounded by the parallel round-complexity of this algorithm for the same permutation. One may wonder if this also holds for the randomized greedy maximal independent set (MIS) algorithm which processes the* vertices *in a random order and adds each encountered feasible vertex to the independent set greedily (see [154, 59]). Interestingly, the answer turns out to be negative. See Figure 6.2.*

6.4 The Final Algorithms for the Adjacency List Query Model

In this section, we show how the oracle analysis of Section 6.3 can lead to our claimed bounds of Theorems 6.1 and 6.2 in the adjacency list query model.

As before, let $G = (V, E)$ be an arbitrary graph with n vertices, m edges, maximum degree Δ, and average degree $\bar{d}$. Having defined and analyzed the oracle calls of the GMM algorithm in Section 6.3, we now employ the standard recipe of the literature in estimating the size of MCM or MVC. We sample a number of random vertices and simulate the vertex oracle of Algorithm 2 on each. If our sample size is sufficiently large, the fraction of matched sampled vertices is a good estimate of the fraction of vertices in the graph that are matched by GMM. To formalize this, we first show how to simulate the vertex and edge oracles of Section 6.3 using adjacency list queries.

First, to generate the random permutation π, one can for each edge e sample an independent rank $\sigma(e)$ which is a real in $[0, 1]$ chosen uniformly at random, and then obtain π by sorting the edges in the increasing order of their ranks. This way we can expose the random permutation "on the fly" only where it is needed, avoiding the $\Omega(m)$ time needed for generating it for all the edges. Another challenge remains though. A trivial simulation of the edge oracle $\mathsf{EO}(e, \pi)$ (Algorithm 1) is to generate the rank $\sigma(e')$ of all edges e' incident to e upon calling $\mathsf{EO}(e, \pi)$. The problem with this approach is that the total number of queries in answering $\mathsf{VO}(v, \pi)$ can be as large as $O(T(v, \pi)\Delta)$ whereas we need a bound of $\widetilde{O}(T(v, \pi))$ for our final results. Here $T(v, \pi)$ as defined in Section 6.3 is the number of edges on which the edge oracle is recursively called during the execution of $\mathsf{VO}(v, \pi)$ and the $O(\Delta)$ factor comes from querying up to $O(\Delta)$ neighbors of each such edge. To get rid of this Δ factor, the idea is to expose the neighbors of each edge in "batches," only when they are needed. This leads to the following bound, a variant of which was first proved by [136]:

Lemma 6.23 ([136])**.** *Let v be an arbitrary vertex in a graph $G = (V, E)$. There is an algorithm that draws a* <u>*random*</u> *permutation π over E, and determines whether v is matched in $\mathsf{GMM}(G, \pi)$ in time $\widetilde{O}(T(v, \pi)+1)$ having query access to the adjacency lists. The algorithm succeeds w.h.p.*

We note that Lemma 6.23 is slightly stronger than its variant proved in [136] where the produced answers were only approximately close, in total variation distance, to the actual distribution. We observe that this can be turned to an exact guarantee by using the exact sublinear time binomial samplers of [84, 63]. For a proof of Lemma 6.23 see [26].

6.4.1 Proof of Theorem 6.1: Multiplicative Approximation

We assume, w.l.o.g., in this section that the graph has no singleton vertices, and that the average degree $\bar{d}$ and maximum degree Δ are given. Note that we can simply query the degree of every vertex in the graph, discard all the singleton vertices, and compute $\bar{d}$ and Δ for the rest of the vertices in $O(n)$ time and queries as allowed by Theorem 6.1. Hence, the assumption comes w.l.o.g. Having this assumption, the rest of the algorithm of this section runs in $\widetilde{O}(\Delta/\varepsilon^2)$ time.

As discussed, the general idea is to take k random vertices and run the greedy oracle of Lemma 6.23 on them to see what fraction of them get matched. For the guarantee of Theorem 6.1, it turns out that setting $k = \widetilde{\Theta}(\Delta/\bar{d}\varepsilon^2)$ suffices to see sufficiently many matched vertices. The following simple claim plays a crucial role in arguing that these many samples suffice for our purpose:

Claim 6.24. *For any n-vertex graph G of max degree Δ and average degree $\bar{d}$, $\mu(G) \geq \frac{n\bar{d}}{4\Delta}$.*

Proof. By Vizing's theorem, any graph G of maximum degree Δ has a proper $(\Delta+1)$-edge-coloring. Since the m edges are colored only via $(\Delta+1)$ colors, there must be a color that is assigned to $\geq m/(\Delta+1)$ edges. Since the edges of any color form a matching, the graph must have a matching of size $\geq m/(\Delta+1)$. Noting that $\bar{d} = 2m/n$, we get:

$$\mu(G) \geq \frac{m}{\Delta+1} = \frac{n\bar{d}/2}{\Delta+1} \geq \frac{n\bar{d}}{4\Delta}. \qquad \square$$

Our starting point is the following Algorithm 8:

Algorithm 8: An algorithm used for Theorem 6.1, given parameter $\varepsilon > 0$.

```
1 $k \leftarrow 128 \cdot 24(\Delta \ln n)/(\varepsilon^2 \bar{d})$. ;        // Note that $\bar{d}$ and $\Delta$ are known to the algorithm.
2 Sample $k$ vertices $v_1, \ldots, v_k$ (with replacement) independently and uniformly from $V$.
3 For each $i \in [k]$ run the algorithm of Lemma 6.23 on vertex $v_i$. For each $i \in [k]$ let $X_i$ be
    the indicator of the event that $v_i$ is matched once we run Lemma 6.23 for $v_i$.
4 Let $X \leftarrow \sum_{i=1}^{k} X_i$ and let $f \leftarrow X/k$ be the fraction of vertices $v_1, \ldots, v_k$ that get
    matched.
5 Let $\widetilde{\mu} \leftarrow (1 - \frac{\varepsilon}{2}) fn/2$ and let $\widetilde{\nu} \leftarrow (1 + \frac{\varepsilon}{2}) fn$.
6 **return** $\widetilde{\mu}$ as the estimate for $\mu(G)$ and **return** $\widetilde{\nu}$ as the estimate for $\nu(G)$.
```

We start by analyzing the approximation ratio of Algorithm 8:

Lemma 6.25. *Let $\widetilde{\mu}$ and $\widetilde{\nu}$ be the outputs of Algorithm 8. With probability $1 - 2n^{-4}$,*

$$(1-\varepsilon)\frac{1}{2}\mu(G) \leq \widetilde{\mu} \leq \mu(G) \qquad \& \qquad \nu(G) \leq \widetilde{\nu} \leq (1+\varepsilon)2\nu(G).$$

Proof. Let us now measure the expected value of our estimates. From our definition, $X_i = 1$ if and only if a random vertex v_i for a random permutation π is matched in $\mathsf{GMM}(G, \pi)$. Since the number of vertices matched in a matching is twice the size of the matching, this implies that

$$\mathbf{E}[X_i] = \mathop{\mathbf{Pr}}_{v_i,\pi}[X_i = 1] = \frac{2\,\mathbf{E}_\pi\,|\mathsf{GMM}(G,\pi)|}{n}.$$

As such,

$$\mathbf{E}[X] = \mathbf{E}[X_1 + \ldots + X_k] = \frac{2k\,\mathbf{E}_\pi\,|\mathsf{GMM}(G,\pi)|}{n}. \tag{6.3}$$

Since X is sum of independent Bernoulli random variables, by the Chernoff bound (Proposition 2.1):

$$\mathbf{Pr}\left[|X - \mathbf{E}[X]| \geq \sqrt{12\,\mathbf{E}[X]\ln n}\right] \leq 2\exp\left(-\frac{12\,\mathbf{E}[X]\ln n}{3\,\mathbf{E}[X]}\right) = 2/n^4. \tag{6.4}$$

Noting from Algorithm 8 that $f \cdot n = Xn/k$, inequality (6.4) implies that with probability $1 - 2/n^4$,

$$\begin{aligned}
f \cdot n &\in \frac{(\mathbf{E}[X] \pm \sqrt{12\,\mathbf{E}[X]\ln n})n}{k} \\
&= \frac{\mathbf{E}[X]n}{k} \pm \sqrt{12\,\mathbf{E}[X]n^2k^{-2}\ln n} \\
&= 2\mathop{\mathbf{E}}_{\pi}|\mathsf{GMM}(G,\pi)| \pm \sqrt{24\mathop{\mathbf{E}}_{\pi}|\mathsf{GMM}(G,\pi)|nk^{-1}\ln n} && \text{(By (6.3).)} \\
&= 2\mathop{\mathbf{E}}_{\pi}|\mathsf{GMM}(G,\pi)| \pm \sqrt{\frac{\mathbf{E}_\pi\,|\mathsf{GMM}(G,\pi)|\varepsilon^2 n\bar{d}}{128\Delta}}. && \text{(Since } k = 128 \cdot 24\tfrac{\Delta \ln n}{\varepsilon^2 \bar{d}}.\text{)}
\end{aligned}$$

Note that $\mu(G) \leq 2\,\mathbf{E}_\pi\,|\mathsf{GMM}(G,\pi)|$ and also recall from Claim 6.24 that $\mu(G) \geq \frac{n\bar{d}}{4\Delta}$. As such, we have $\frac{n\bar{d}}{4\Delta} \leq 2\,\mathbf{E}_\pi\,|\mathsf{GMM}(G,\pi)|$. Combined with the range above, with probability $1 - 2/n^4$ we have

$$f \cdot n \in 2\mathop{\mathbf{E}}_{\pi}|\mathsf{GMM}(G,\pi)| \pm \sqrt{\frac{(\mathbf{E}_\pi\,|\mathsf{GMM}(G,\pi)|)^2\varepsilon^2}{16}} = \left(2 \pm \frac{\varepsilon}{4}\right)\mathop{\mathbf{E}}_{\pi}|\mathsf{GMM}(G,\pi)|.$$

Since $\widetilde{\mu} = (1 - \frac{\varepsilon}{2})f \cdot n/2$ and $\widetilde{\nu} = (1 + \frac{\varepsilon}{2})f \cdot n$, this means

$$(1 - \varepsilon)\mathop{\mathbf{E}}_{\pi}|\mathsf{GMM}(G,\pi)| \leq \widetilde{\mu} \leq \mathop{\mathbf{E}}_{\pi}|\mathsf{GMM}(G,\pi)|, \tag{6.5}$$

$$2\mathop{\mathbf{E}}_{\pi}|\mathsf{GMM}(G,\pi)| \leq \widetilde{\nu} \leq (1 + \varepsilon)2\mathop{\mathbf{E}}_{\pi}|\mathsf{GMM}(G,\pi)|. \tag{6.6}$$

Next, observe that $\frac{1}{2}\mu(G) \leq \mathbf{E}_\pi\,|\mathsf{GMM}(G,\pi)| \leq \mu(G)$ since a maximal matching is a 2-approximate maximum matching, and $\nu(G) \leq 2\,\mathbf{E}_\pi\,|\mathsf{GMM}(G,\pi)| \leq 2\nu(G)$ since the set of vertices of a maximal matching is a 2-approximate minimum vertex cover. These, plugged into (6.5) and (6.6) give the desired inequalities of the lemma, completing the proof. □

We are now ready to prove Theorem 6.1.

Proof of Theorem 6.1. By Theorem 3.2, for a random vertex $v \in V$, $\mathbf{E}_{v,\pi}[T(v,\pi)] = O(\bar{d} \cdot \log n)$. As such, for each vertex v_i of Algorithm 8, the algorithm of Lemma 6.23 takes $\widetilde{O}(\bar{d}+1)$ expected time to determine whether it is matched in a random permutation. Since we run this algorithm for $k = \widetilde{O}(\Delta/\varepsilon^2\bar{d})$ vertices, the expected running time of Algorithm 8 is $\widetilde{O}(\Delta/\varepsilon^2\bar{d}) \cdot \widetilde{O}(\bar{d}+1) = \widetilde{O}(\Delta/\varepsilon^2)$ where the latter equality crucially uses our assumption of the start of the section that there are no singleton vertices in the graph, which implies $1/\bar{d} = O(1)$.

To achieve the high probability bound on the time-complexity, we run $\Theta(\log n)$ instances of Algorithm 8 in parallel and return the output of the instance that terminates first. By Markov's inequality, each instance terminates in time $\widetilde{O}(\Delta/\varepsilon^2)$ with a constant probability. As such, at least one of the instances terminates in $\widetilde{O}(\Delta/\varepsilon^2)$ time with probability $1 - 1/\operatorname{poly}(n)$. Recall also that we spent $O(n)$ time at the start of the section to throw away singleton vertices and compute $\bar{d}$ and Δ. As such, the total time complexity is, w.h.p., $O(n) + \widetilde{O}(\Delta/\varepsilon^2)$. On the other hand, since the approximation ratio guarantee of Lemma 6.25 holds with probability $1 - 1/\operatorname{poly}(n)$ for each instance, *all* $O(\log n)$ instances (including the one that terminates first) achieve the claimed approximation with a high probability of $1 - 1/\operatorname{poly}(n)$. □

6.4.2 Proof of Theorem 6.2: Multiplicative-Additive Approximation

The algorithm we use for Theorem 6.2 is similar to Algorithm 8 for Theorem 6.1 except that the additive εn error of Theorem 6.2 allows us to take $\widetilde{O}(1/\varepsilon^2)$ sample vertices instead of $\widetilde{O}(\Delta/\varepsilon^2\bar{d})$ as in Algorithm 8. Formally, we use the following Algorithm 9:

Algorithm 9: An algorithm used for Theorem 6.2, given parameter $\varepsilon > 0$.

```
1 $k \leftarrow 16 \cdot 24 \ln n/\varepsilon^2$.
2 Sample $k$ vertices $v_1, \ldots, v_k$ (with replacement) independently and uniformly from $V$.
3 For each $i \in [k]$ run the algorithm of Lemma 6.23 on vertex $v_i$. For each $i \in [k]$ let $X_i$ be
    the indicator of the event that $v_i$ is matched once we run Lemma 6.23 for $v_i$.
4 Let $X \leftarrow \sum_{i=1}^{k} X_i$ and let $f \leftarrow X/k$ be the fraction of vertices $v_1, \ldots, v_k$ that get
    matched.
5 Let $\widetilde{\mu} \leftarrow \frac{fn}{2} - \frac{\varepsilon}{2}n$ and let $\widetilde{\nu} \leftarrow fn + \frac{\varepsilon}{4}n$.
6 return $\widetilde{\mu}$ as the estimate for $\mu(G)$ and return $\widetilde{\nu}$ as the estimate for $\nu(G)$.
```

Lemma 6.26. *Let $\widetilde{\mu}$ and $\widetilde{\nu}$ be the outputs of Algorithm 9. With probability* $1 - 2n^{-4}$,

$$\frac{1}{2}\mu(G) - \varepsilon n \leq \widetilde{\mu} \leq \mu(G) \qquad \& \qquad \nu(G) \leq \widetilde{\nu} \leq 2\nu(G) + \varepsilon n.$$

Proof. Observe that inequalities (6.3) and (6.4) that we proved for Algorithm 8 hold for Algorithm 9 too for exactly the same reasons. Particularly, inequality (6.4) implies that with probability $1 - 2/n^4$,

$$\begin{aligned}
f \cdot n &\in \frac{(\mathbf{E}[X] \pm \sqrt{12\,\mathbf{E}[X] \ln n})n}{k} \\
&= \frac{\mathbf{E}[X]n}{k} \pm \sqrt{12\,\mathbf{E}[X] n^2 k^{-2} \ln n} \\
&= 2 \mathop{\mathbf{E}}_{\pi} |\mathsf{GMM}(G,\pi)| \pm \sqrt{24 \mathop{\mathbf{E}}_{\pi} |\mathsf{GMM}(G,\pi)| n k^{-1} \ln n} && \text{(By (6.3).)} \\
&= 2 \mathop{\mathbf{E}}_{\pi} |\mathsf{GMM}(G,\pi)| \pm \sqrt{\mathop{\mathbf{E}}_{\pi} |\mathsf{GMM}(G,\pi)| \varepsilon^2 n/16}. && \text{(Since } k = 16 \cdot 24 \tfrac{\ln n}{\varepsilon^2}.\text{)} \\
&\in 2 \mathop{\mathbf{E}}_{\pi} |\mathsf{GMM}(G,\pi)| \pm \varepsilon n/4. && \text{(Since } \mathbf{E}_\pi |\mathsf{GMM}(G,\pi)| \leq n.\text{)}
\end{aligned}$$

Combined with $\widetilde{\mu} = \frac{fn}{2} - \frac{\varepsilon}{2}n$ and $\widetilde{\nu} = fn + \frac{\varepsilon}{4}n$, this implies that, w.h.p.,

$$\mathop{\mathbf{E}}_{\pi} |\mathsf{GMM}(G,\pi)| - \varepsilon n \leq \widetilde{\mu} \leq \mathop{\mathbf{E}}_{\pi} |\mathsf{GMM}(G,\pi)|, \tag{6.7}$$

$$2 \mathop{\mathbf{E}}_{\pi} |\mathsf{GMM}(G,\pi)| \leq \widetilde{\nu} \leq 2 \mathop{\mathbf{E}}_{\pi} |\mathsf{GMM}(G,\pi)| + \varepsilon n. \tag{6.8}$$

Plugging $\frac{1}{2}\mu(G) \leq \mathbf{E}_\pi |\mathsf{GMM}(G,\pi)| \leq \mu(G)$ and $\nu(G) \leq 2\,\mathbf{E}_\pi |\mathsf{GMM}(G,\pi)| \leq 2\nu(G)$ into (6.7) and (6.8) completes the proof. □

Proof of Theorem 6.2. As discussed in the proof of Theorem 6.1, each call to Lemma 6.23 for a randomly chosen vertex takes $\widetilde{O}(\bar{d}+1)$ expected time. Since $k = \widetilde{O}(1/\varepsilon^2)$, Algorithm 9 takes $\widetilde{O}((\bar{d}+1)/\varepsilon^2)$ expected time in total.

To achieve the high probability bound on the time-complexity, as in the proof of Theorem 6.1, we run $\Theta(\log n)$ instances of Algorithm 9 in parallel and return the output of the instance that terminates first. By Markov's inequality, each instance terminates in time $\widetilde{O}((\bar{d}+1)/\varepsilon^2)$ with a constant probability. As such, at least one of the instances terminates in $\widetilde{O}((\bar{d}+1)/\varepsilon^2)$ time with probability $1 - 1/\operatorname{poly}(n)$. On the other hand, since the approximation guarantee of Lemma 6.26 holds with probability $1 - 1/\operatorname{poly}(n)$ for each instance, *all* $O(\log n)$ instances (including the one that terminates first) achieve a $(2, \varepsilon n)$-approximation with probability $1 - 1/\operatorname{poly}(n)$. □

6.5 The Final Algorithm for the Adjacency Matrix Query Model

In this section, we prove Theorem 6.3. The first challenge is that Lemma 6.23 is based on adjacency list queries. As such, we need a way of implementing these adjacency list queries in the adjacency matrix model. To do this, we transform the input graph G to another graph

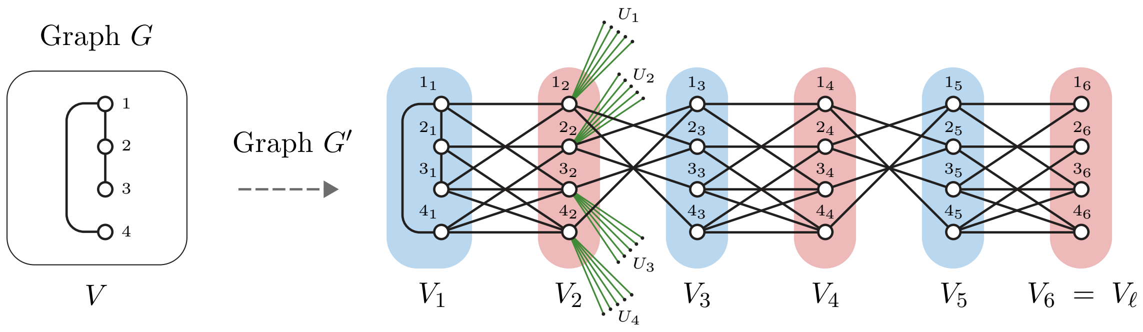


Figure 6.3: An example of graph G' constructed from G. Here the graph G has $n = 4$ vertices. The illustration is drawn for $\ell = 6$ and (for simplicity) for $s = |U_i| = 5$. Note that all the vertices in V_1 and $V_3 \cup \ldots \cup V_{\ell-1}$ have degree exactly $n = 4$, the vertices in V_2 have degree exactly $n + s = 9$, the vertices in $U_1, \ldots, U_n$ have degree exactly one, and the vertices in V_ℓ have varying degrees.

G' where (most) adjacency list queries to G' can be implemented efficiently via adjacency matrix queries to G and at the same time the oracle calls on G' suffice to approximate the size of MCM and MVC for G.

We note that another such reduction was given before by [136]. However, the reduction of [136] is not applicable in our case since it, crucially, adds parallel edges and self-loops to G' while Theorem 3.2 is proved for simple graphs. Our reduction is, in fact, completely disjoint from that of [136] and uses the properties of RGMM more aggressively.

We first make a mild assumption that ε in Theorem 6.3 is at least $1/n$, noting that otherwise $\widetilde{O}(n/\varepsilon^3)$ is large enough to query the whole graph, making Theorem 6.3 trivial.

Let us now formalize the construction of graph $G' = (V', E')$ from the input graph $G = (V, E)$ (see Figure 6.3 for an illustration). Throughout this section, we continue to use n to denote the number of vertices in the original graph G. Define $\ell := C \log n$ for a sufficiently large constant C and let $s := 10n/\varepsilon$. The vertex-set V' of G' is defined as follows:

- For any $i \in [\ell]$ define set $V_i = \{1_i, 2_i, \ldots, n_i\}$ of size n.
- For any $i \in [n]$ define set $U_i = \{1'_i, 2'_i, \ldots, s'_i\}$ of size s.
- The vertex set V' of G' is set $(V_1 \cup \ldots \cup V_\ell) \cup (U_1 \cup \ldots \cup U_n)$. Note, in particular, that G' has $n' := |V'| = \ell n + ns = \Theta(n^2/\varepsilon)$ vertices.

We formalize the edge-set E' of G' by describing the adjacency-list of each vertex $v \in V'$ (while being careful that for each edge $(u, v) \in E'$, both u and v appear in each other's adjacency lists). Suppose that the vertices of graph G are labeled as $V = \{1, \dots, n\}$. The edge-set of G' is as follows:

1. For any vertex $v_1 \in V_1$ and any $i \in [\deg_{G'}(v)] = [n]$: If $(v, i) \in E$ then the i-th neighbor of v_1 is vertex $i_1 \in V_1$, otherwise it is vertex $i_2 \in V_2$. Note, in particular, that with this construction $G'[V_1]$ is isomorphic to G.
2. For any vertex $v_j \in V_j$ with even $j \in \{4, \dots, \ell - 1\}$ and any $i \in [\deg_{G'}(v_j)] = [n]$: If $(v, i) \in E$ then the i-th neighbor of v_j is $i_{j+1} \in V_{j+1}$, otherwise it is $i_{j-1} \in V_{j-1}$.
3. For any vertex $v_j \in V_j$ with odd $j \in \{3, \dots, \ell - 1\}$ and any $i \in [\deg_{G'}(v_j)] = [n]$: If $(v, i) \in E$ then the i-th neighbor of v_j is $i_{j-1} \in V_{j-1}$, otherwise it is $i_{j+1} \in V_{j+1}$.
4. For any vertex $v_2 \in V_2$ and any $i \in [\deg_{G'}(v)] = [n + s]$: If $i \leq n$ and $(v, i) \in E$, the i-th neighbor of v_2 is $i_3 \in V_3$; If $i \leq n$ and $(v, i) \notin E$, then the i-th neighbor of v_2 is $i_1 \in V_1$; finally if $i > n$ then the i-th neighbor of v_2 is $v'_{i-n} \in U_v$.
5. For vertices in V_ℓ, we assume that their adjacency lists are sorted in an arbitrary way, say in the increasing order of vertex ID's. Note that all the edges of V_ℓ have their other endpoint in $V_{\ell-1}$ and are already defined above.
6. Any vertex in U_j has exactly one neighbor, and it is vertex $j_2 \in V_2$.

The following observation follows immediately from the construction above and the way neighbors of each vertex are ordered in their adjacency lists:

Observation 6.27. *For any vertex $v \in V' \setminus V_\ell$, $\deg_{G'}(v)$ does not depend on the edges in G and is, therefore, known a priori with zero queries to G. Furthermore, for any $v \in V' \setminus V_\ell$ and any $i \in [\deg_{G'}(v)]$ one can determine the i-th edge of v in its adjacency list of G' with at most one adjacency matrix query to G.*

Proof. The construction already fixes the degree of any vertex $v \in V' \setminus V_\ell$ and so the first part of the observation is trivial. For the second part, note that for any vertex $v_j \in V_j$ with $j \in [\ell - 1]$ and any $i \in [\deg_{G'}(v_j)]$, querying whether $(v, i) \in E$ suffices to determine the i-th edge of v_j by the construction. The vertices in U_j, on the other hand, have only one neighbor j_2 and this can be answered with zero queries to G. □

By Observation 6.27, adjacency list queries to all vertices of G', except for those in V_ℓ, can be answered very efficiently, each with just one query to the adjacency matrix of G. However, adjacency list queries to V_ℓ are still costly. In fact, if the algorithm queries the adjacency list or the degree of a vertex $v_\ell \in V_\ell$, we immediately terminate the process and output FAIL. Fortunately, if we start the vertex oracle of Lemma 6.23 from a vertex in V_1, it can be shown that it is highly unlikely for the process to require such queries:

Claim 6.28. *Let $v \in V_1$ and suppose that we run the algorithm of Lemma 6.23 on v. With probability $1 - 1/\operatorname{poly}(n)$, the algorithm does not query the adjacency list nor the degree of any vertex in V_ℓ.*

Proof. The crucial observation is that graph G' is constructed such that the distance between a vertex in V_1 and a vertex in V_ℓ is at least $\ell - 1$. On the other hand, recall from Claim 6.21 that every query path is asymptotically bounded by the algorithm's parallel round-complexity, which by Lemma 6.20, is $O(\log|V'|)$ with probability $1-1/\operatorname{poly}(|V'|)$. Since $|V'| = n\ell+ns = \Theta(n\log n + n^2/\varepsilon) = \Theta(n^2/\varepsilon)$ and $\varepsilon > \frac{1}{n}$ (assumed at the start of the section), the maximum query path has size $O(\log(n^2/\varepsilon)) = O(\log n)$ with probability $1 - 1/\operatorname{poly}(n)$.

This means that if the constant C in $\ell = C\log n$ is large enough, a query process starting from V_1, with probability $1 - 1/\operatorname{poly}(n)$ does not reach V_ℓ. Hence, Lemma 6.23, which implements the oracle for a random permutation, will not query the adjacency list nor the degree of any vertex in V_ℓ if the starting vertex is in V_1, with probability $1-1/\operatorname{poly}(n)$. □

Observe that for Claim 6.28 to be applicable, we have to start our queries from the vertices in V_1 and not any vertex of G'. Claim 6.29 below shows that if we pick a random vertex from V_1 instead of the whole vertex-set V', the expected query complexity is still small.

We use $T_{G'}(v,\pi)$ instead of $T(v,\pi)$ (defined in Section 6.3 and used in Lemma 6.23) to emphasize that we run RGMM on graph G' and not G in this section.

Claim 6.29. *Let π be a random permutation over the edge-set E' of G'. For a vertex v chosen uniformly at random from V_1 and independently from π,*

$$\mathop{\mathbf{E}}_{v\sim V_1,\pi}[T_{G'}(v,\pi)] = \widetilde{O}(n/\varepsilon).$$

Proof. By the construction, graph G' has $|E'| = O(n^2\ell + ns) = \widetilde{\Theta}(n^2/\varepsilon)$ edges and $|V'| = n\ell + ns = \Theta(n^2/\varepsilon)$ vertices. Applying Theoroem 3.2, for a vertex $v \in V'$ chosen uniformly at random, we get

$$\mathop{\mathbf{E}}_{v\sim V',\pi}[T_{G'}(v,\pi)] = O\left(\frac{|E'|}{|V'|}\log|V'|\right).$$

Instead of querying a random vertex $v \in V'$, suppose that we sum over all of them. This gives:

$$\sum_{v \in V'} \mathop{\mathbf{E}}_{\pi}[T_{G'}(v,\pi)] = |V'| \cdot \mathop{\mathbf{E}}_{v \sim V',\pi}[T_{G'}(v,\pi)] = O\left(|E'| \log |V'|\right) = \widetilde{O}(n^2/\varepsilon). \tag{6.9}$$

Finally, since $|V_1| = n$, for a vertex v chosen uniformly at random from V_1, we have

$$\mathop{\mathbf{E}}_{v \sim V_1,\pi}[T_{G'}(v,\pi)] \leq \left(\sum_{v \in V'} \mathop{\mathbf{E}}_{\pi}[T_{G'}(v,\pi)]\right) / |V_1| \overset{(6.9)}{=} \widetilde{O}(n^2/\varepsilon)/n = \widetilde{O}(n/\varepsilon). \quad \square$$

Now that we know queries starting from a random vertex in V_1 can be implemented efficiently in $\widetilde{O}(n/\varepsilon)$ expected time, the question is how can we use these queries to get our estimators for the size of maximum matching and minimum vertex cover of G.

For any permutation π over E', let us define

$$M_1(\pi) := \mathsf{GMM}(G',\pi) \cap (V_1 \times V_1)$$

to be the set of edges in matching $\mathsf{GMM}(G',\pi)$ with both endpoints in V_1. The following Claim 6.30 shows that the expected value of $M_1(\pi)$ for a random permutation π can be used for approximating both maximum matching and the minimum vertex cover of G.

The idea behind Claim 6.30 is as follows. Note that by construction, $G'[V_1]$ is isomorphic to G and so for any π, there is a matching in G that corresponds to $M_1(\pi)$. However, the vertices in V_1 can also be matched to the vertices in V_2 in graph G', and so $M_1(\pi)$ is not necessarily a maximal matching of $G'[V_1]$. The key insight, however, is that the vast majority of vertices $v_2 \in V_2$ will actually be matched to their neighbors in U_v in a RGMM of G'. Hence, $M_1(\pi)$ for a random permutation π is indeed close to a maximal matching of $G'[V_1]$ and its size can be used to approximate both the size of MCM and that of the MVC of G. Formally:

Claim 6.30. *It holds that*

$$\frac{1}{2}\mu(G) - \frac{\varepsilon}{20}n \leq \mathop{\mathbf{E}}_{\pi}|M_1(\pi)| \leq \mu(G) \qquad \& \qquad \nu(G) - \frac{\varepsilon}{10}n \leq 2\mathop{\mathbf{E}}_{\pi}|M_1(\pi)| \leq 2\nu(G).$$

Proof. Let us use $B_1(\pi)$ to denote the set of vertices in V_1 that are matched to V_2 in $\mathsf{GMM}(G',\pi)$. We first show that for every permutation π over E', it holds that

$$\frac{1}{2}(\mu(G) - |B_1(\pi)|) \leq |M_1(\pi)| \leq \mu(G) \qquad \& \qquad \nu(G) - |B_1(\pi)| \leq 2|M_1(\pi)| \leq 2\nu(G). \tag{6.10}$$

For the first inequality, observe that $M_1(\pi)$ is a matching of $G'[V_1]$ and $G'[V_1]$ is isomorphic to G by the construction; thus, clearly $|M_1(\pi)| \leq \mu(G)$.

For the second inequality, observe that $M_1(\pi)$ is a maximal matching of $G'[V_1 \setminus B_1(\pi)]$ which is isomorphic to $G[V \setminus B_1(\pi)]$. Since a maximal matching is at least half the size of a maximum matching, we get $|M_1(\pi)| \geq \frac{1}{2}\mu(G[V \setminus B_1(\pi)]) \geq \frac{1}{2}(\mu(G) - |B_1(\pi)|)$.

For the third inequality, note that since $M_1(\pi)$ is a maximal matching of $G'[V_1 \setminus B_1(\pi)]$, the set of vertices matched by it covers all edges in $G'[V_1 \setminus B_1(\pi)]$. Adding the vertices in $B_1(\pi)$ to this set, we cover all edges in $G'[V_1]$. Hence $\nu(G'[V_1]) \leq 2|M_1(\pi)| + |B_1(\pi)|$. The inequality then follows from $G'[V_1]$ being isomorphic to G.

For the fourth inequality, note that since $M_1(\pi)$ is a matching in $G'[V_1]$ and each vertex can cover at most one edge of it, we have $|M_1(\pi)| \leq \nu(G'[V_1])$. Given that $G'[V_1]$ is isomorphic to G, we thus get $|M_1(\pi)| \leq \nu(G)$. Multiplying through by a factor of 2 and adding $|B_1(\pi)|$ to both sides, we get $2|M_1(\pi)| + |B_1(\pi)| \leq 2\nu(G) + |B_1(\pi)|$.

Now, observe that in any permutation π, if the lowest rank neighbor of a vertex $v_2 \in V_2$ is connected to U_v, then this edge is in matching $\mathsf{GMM}(\pi)$ since the vertices in U_v have degree exactly one. Moreover, since each vertex $v_2 \in V_2$ has degree exactly $n + s$ and s of these neighbors are to U_v, the minimum rank edge of v_2 is indeed connected to U_v with probability $\frac{s}{n+s} = \frac{10n/\varepsilon}{n+10n/\varepsilon} \geq 1 - \varepsilon/10$. As such the number of vertices in V_2 that are matched to V_1 is at most $\frac{\varepsilon}{10}|V_2| = \frac{\varepsilon}{10}n$ in expectation taken over π. This, in turn, implies that $\mathbf{E}_\pi |B_1(\pi)| \leq \frac{\varepsilon}{10}n$. Taking expectation over a random π in (6.10) and plugging this upper bound for $\mathbf{E}_\pi |B_1(\pi)|$ proves the claim. □

Finally, the algorithm we use for Theorem 6.3 is formalized below as Algorithm 10.

Algorithm 10: An algorithm used for Theorem 6.3, given parameter $\varepsilon > 0$.

```
1 Let $G' = (V', E')$ be the graph described above (we do not explicitly construct $G'$ here).
2 $k \leftarrow 16 \cdot 24 \ln n/\varepsilon^2$.
3 Sample $k$ vertices $v^1, \ldots, v^k$ (with replacement) independently and uniformly from $V_1$.
4 For each $i \in [k]$ run the algorithm of Lemma 6.23 on vertex $v^i$, for graph $G'$. Let $X_i$ be
    the indicator of the event that $v^i$ is matched to another vertex in $V_1$.
5 Let $X \leftarrow \sum_{i=1}^k X_i$ and let $f \leftarrow X/k$ be the fraction of $v_1, \ldots, v_k$ that get matched to $V_1$.
6 Let $\widetilde{\mu} \leftarrow \frac{fn}{2} - \frac{\varepsilon}{2}n$ and $\widetilde{\nu} \leftarrow fn + \frac{\varepsilon}{2}n$.
7 return $\widetilde{\mu}$ as the estimate for $\mu(G)$ and return $\widetilde{\nu}$ as the estimate for $\nu(G)$.
```

Let us analyze the approximation ratio of Algorithm 10.

Lemma 6.31. *Let $\widetilde{\mu}$ and $\widetilde{\nu}$ be the outputs of Algorithm 10. With probability $1 - 2n^{-4}$,*

$$\frac{1}{2}\mu(G) - \varepsilon n \leq \widetilde{\mu} \leq \mu(G) \qquad \& \qquad \nu(G) \leq \widetilde{\nu} \leq 2\nu(G) + \varepsilon n.$$

Proof. Let us now measure the expected value of our estimates. From our definition, $X_i = 1$ if and only if a random vertex $v^i \in V_1$ for a random permutation π is matched to another vertex of V_1 in $\mathsf{GMM}(G', \pi)$. Recall that we defined $M_1(\pi)$ to be the set of edges in $\mathsf{GMM}(G', \pi) \cap (V_1 \times V_1)$. Combined with the fact that the number of vertices matched in a matching is twice the size of the matching, this implies that

$$\mathbf{E}[X_i] = \mathop{\mathbf{Pr}}_{v^i,\pi}[X_i = 1] = \frac{2\,\mathbf{E}_\pi |M_1(\pi)|}{|V_1|} = \frac{2\,\mathbf{E}_\pi |M_1(\pi)|}{n}.$$

As such,

$$\mathbf{E}[X] = \mathbf{E}[X_1 + \ldots + X_k] = \frac{2k\,\mathbf{E}_\pi |M_1(\pi)|}{n}. \tag{6.11}$$

Since X is sum of independent Bernoulli random variables, by the Chernoff bound (Proposition 2.1):

$$\mathbf{Pr}\left[|X - \mathbf{E}[X]| \geq \sqrt{12\,\mathbf{E}[X] \ln n}\right] \leq 2 \exp\left(-\frac{12\,\mathbf{E}[X] \ln n}{3\,\mathbf{E}[X]}\right) = 2/n^4. \tag{6.12}$$

Noting from Algorithm 10 that $f \cdot n = Xn/k$, inequality (6.12) implies that with probability $1 - 2/n^4$,

$$\begin{aligned}
f \cdot n &\in \frac{(\mathbf{E}[X] \pm \sqrt{12\,\mathbf{E}[X] \ln n})n}{k} \\
&= \frac{\mathbf{E}[X]n}{k} \pm \sqrt{12\,\mathbf{E}[X] n^2 k^{-2} \ln n} \\
&= 2 \mathop{\mathbf{E}}_{\pi} |M_1(\pi)| \pm \sqrt{24 \mathop{\mathbf{E}}_{\pi} |M_1(\pi)| n k^{-1} \ln n} && \text{(By (6.11).)} \\
&= 2 \mathop{\mathbf{E}}_{\pi} |M_1(\pi)| \pm \sqrt{\mathop{\mathbf{E}}_{\pi} |M_1(\pi)| \varepsilon^2 n / 16} && (\text{Since } k = 16 \cdot 24 \tfrac{\ln n}{\varepsilon^2}.) \\
&\in 2 \mathop{\mathbf{E}}_{\pi} |M_1(\pi)| \pm \frac{\varepsilon n}{4}. && (\text{Since } \mathbf{E}_\pi |M_1(\pi)| \leq |V_1| = n.)
\end{aligned}$$

Combined with $\frac{1}{2}\mu(G) - \frac{\varepsilon}{20} n \leq \mathbf{E}_\pi |M_1(\pi)| \leq \mu(G)$ from Claim 6.30, and $\widetilde{\mu} = \frac{fn}{2} - \frac{\varepsilon}{2} n$, the range above for fn implies $\frac{1}{2}\mu(G) - \varepsilon n < \widetilde{\mu} < \mu(G)$.

Combined with $\nu(G) - \frac{\varepsilon}{10} n \leq 2\,\mathbf{E}_\pi |M_1(\pi)| \leq 2\nu(G)$ from Claim 6.30 and $\widetilde{\nu} = fn + \frac{\varepsilon}{2} n$, the range above for fn implies $\nu(G) < \widetilde{\nu} < 2\nu(G) + \varepsilon n$. □

Proof of Theorem 6.3. By Claim 6.29 each call to Lemma 6.23 for a random vertex from V_1 leads to $\widetilde{O}(n/\varepsilon)$ adjacency list queries to G'. By Claim 6.28, with probability $1 - 1/\operatorname{poly}(n)$, none of these calls leads to querying the adjacency list or degree of a vertex in V_k. Hence, by Observation 6.27, each of these adjacency list queries to G', can be implement with one adjacency matrix query to G in $O(1)$ time. The total expected number of adjacency matrix queries to G and the time-complexity of the algorithm, is therefore, $k \cdot \widetilde{O}(n/\varepsilon) = \widetilde{O}(n/\varepsilon^3)$.

To achieve the high probability bound on the time-complexity, as in the proofs of Theorems 6.1 and 6.2, we run $\Theta(\log n)$ instances of Algorithm 10 in parallel and return the output of the instance that terminates first. By Markov's inequality, each instance terminates in time $\widetilde{O}(n/\varepsilon^3)$ with a constant probability. As such, at least one of the instances terminates in $\widetilde{O}(n/\varepsilon^3)$ time with probability $1 - 1/\operatorname{poly}(n)$. On the other hand, since the approximation guarantee of Lemma 6.31 holds with probability $1 - 1/\operatorname{poly}(n)$ for each instance, *all* $O(\log n)$ instances (including the one that terminates first) achieve a $(2, \varepsilon n)$-approximation with probability $1 - 1/\operatorname{poly}(n)$. □

Part III

Dynamic Algorithms

Chapter 7

Fully Dynamic Maximal Independent Set

A maximal independent set (MIS) is another fundamental graph property with several theoretical and practical applications. It is one of the most well-studied problems in distributed and parallel settings following the seminal works of [126, 6]. MIS has also been studied in a variety of other models and has diverse applications such as approximating matching and vertex cover [133, 154], graph coloring [126, 122], clustering [3], leader-election [79], and many others.

In this chapter, we consider MIS in *fully-dynamic* graphs. The graph is updated via both edge insertions and deletions and the goal is to maintain an MIS by the end of each update. Dynamic graphs constitute an active area of research and have seen a plethora of results over the past two decades. The MIS problem in dynamic graphs has also attracted a significant attention, especially recently [64, 14, 101, 81, 137, 17]. We first overview these works below and then describe our contribution.

Related Work: In static graphs with m edges, a simple greedy algorithm can find an MIS in $O(m)$ time. As such, one can trivially maintain MIS by recomputing it from scratch after each update, in $O(m)$ time. In a pioneering work, Censor-Hillel, Haramaty, and Karnin [64] presented a round-efficient randomized algorithm for MIS in dynamic *distributed* networks. Implementing the algorithm of [64] in the sequential setting—the focus of this chapter—requires $\Omega(\Delta)$ update-time (see [64, Section 6]) where Δ is the maximum-degree in the graph which can be as large as $\Omega(n)$ or even $\Omega(m)$ for sparse graphs. Improving this bound was one of the major problems the authors left open. Later, in a breakthrough, Assadi, Onak, Schieber, and Solomon [14] presented a deterministic algorithm with $O(m^{3/4})$ update-time; thereby improving the $O(m)$ bound for all graphs. This result was further improved in a series of subsequent papers [101, 81, 137, 17]. The current state-of-the-art is a randomized algorithm due to Assadi *et al.* [17], which requires $\widetilde{O}(\min\{\sqrt{n}, m^{1/3}\})$ amortized update-time in n-vertex graphs.

Our Contribution: In this chapter, we show that it is possible to maintain an MIS of fully-dynamic graphs in polylogarithmic time.[1] This exponentially improves over the prior algorithms, which all have polynomial update-time on general graphs. Our algorithm is randomized and requires the standard *oblivious adversary*[2] assumption (as do all previous randomized algorithms).

Theorem 7.1 (main result)**.** *There is a data structure to maintain an MIS against an oblivious adversary in a fully-dynamic graph that, per update, takes* $O(\log^2 \Delta \cdot \log^2 n)$ *expected time. Furthermore, the number of adjustments to the MIS per update is* $O(1)$ *in expectation.*

Since our algorithm bounds the expected time *per update* without amortization, we can use it as a black-box in a framework of Bernstein *et al.* [53, Theorem 1.1] to also get a worst-case guarantee w.h.p. (We note that this comes at the cost of losing the guarantee on the adjustment-complexity.)

Corollary 7.2. *There is a data structure to maintain an MIS against an oblivious adversary in a fully-dynamic graph that w.h.p. has* $O(\log^2 \Delta \cdot \log^4 n)$ *worst-case update-time.*

To prove Theorem 7.1, we give an algorithm that carefully simulates RGMIS (see Section 3.2 for its definition and properties). We fix the random order over the vertices at the start of the algorithm. Once this order is fixed, the GMIS of becomes unique for any given edge-set. This is particularly useful for dynamic graphs as it makes the output *history-independent*. That is, the order of edge insertions and deletions by the adversary cannot affect the reported MIS. See [64, Section 5] for more discussion on this property.

We note that maintaining RGMIS has been done before by Censor-Hillel *et al.* [64] and also partially by Assadi *et al.* [17] who combined it with another deterministic algorithm. However, as discussed above, both these algorithms require a polynomial update-time. The

[1]Independently and currently with our work, Chechik and Zhang [68] also obtained an algorithm with essentially the same guarantee.

[2]In the standard *oblivious adversarial* model, the adversary can feed in any sequence of edge updates and is aware of the algorithm to be used, but is unaware of the random-bits used by the algorithm. Equivalently, one can assume that the sequence of edge updates is picked adversarially *before* the dynamic algorithm starts to operate.

novelty of our approach is in (1) *the algorithm and data structures* with which we maintain this MIS, and (2) the *analysis* of why polylogarithmic time is sufficient. The high-level intuitions behind both the algorithm and the analysis are presented in Section 7.1.

7.1 Technical Overview

As pointed out earlier, our main contribution is to show that it is actually possible to maintain RGMIS at an expected polylogarithmic cost per update. In this section we attempt to explain some of the barriers and how our work overcomes them.

The first hurdle behind maintaining the greedy MIS is that it may change a lot under updates. But it is also well-known [64, Theorem 1] that for a random ordering, the expected alteration to GMIS after the insertion or deletion of a single edge is $O(1)$. This already shows that maintaining RGMIS is sufficient to get an algorithm with $O(1)$ expected adjustments per update. However, it is not clear how to *detect* these changes and maintain RGMIS efficiently: The natural algorithm to do so would do a breadth-first-search (BFS) from the endpoints of the edge being updated, but even exploring the neighborhood of a single vertex of degree Δ might require $\Omega(\Delta)$ time which is prohibitively expensive for general graphs where Δ can be as large as $\Omega(n)$.[3]

Our first idea is to maintain not just the RGMIS, but also the "eliminator" of every vertex v in the graph. Briefly, given a ranking $\pi : V \to [0, 1]$, the eliminator of a vertex v in a graph G under π is its neighbor in G of the lowest rank that belongs to the RGMIS. (If v is in the RGMIS, then its eliminator is defined to be itself.) Maintaining the eliminators only seems to complicate our task further: (1) Even if the MIS changes by a little, it is conceivable that the eliminators of many more vertices might change. (2) It is still unclear how to find the set of vertices whose eliminators have changed in $o(\Delta)$ time.

For problem (1) we extend the classical analysis [64, 154], which showed that the MIS changes only by a little after each update, to show that the eliminators are also extremely robust under updates. We stress that this extension is not simple and requires many new ideas. Overall, we get the following guarantee which may be of independent interest. (It is crucial for our analysis that we prove this bound on *vertex* updates—we will discuss this towards the end of this section.)

Theorem 7.10 (informal—see page 115 for the formal statement)**.** *For any arbitrary vertex*

[3]This is precisely the $\Omega(\Delta)$ barrier mentioned in Section 6 of [64].

addition or deletion, the expected number of vertices whose eliminator changes is $O(\log n)$.

We now turn to problem (2), i.e., the challenge of maintaining information such as membership in the MIS and eliminators of vertices. Consider an edge update (a, b) with $\pi(a) < \pi(b)$ and suppose that this changes b's MIS-status. A priori, this seems to require exploring every neighbor of b (at the very least) and checking to see if their status or eliminator changes. But a quick examination reveals we only need to explore those neighbors u of b whose eliminators have rank larger than $\pi(a)$. (Vertices with rank less than $\pi(a)$ don't change their membership in the MIS, and so vertices with eliminators of rank less than $\pi(a)$ don't change their eliminator.) To help this prune our exploration space, it would make sense to store all neighbors of b (and of every vertex for that matter) in a search tree indexed by the rank of their eliminator and indeed this is an idea we pursue. However maintaining every neighbor of b indexed by its eliminator-rank leads to new maintenance problems: Up to Δ trees may need to be updated when b changes its eliminator-rank! We overcome this barrier with the following solution (which is essentially our final solution): We only maintain the neighbors of *low-rank* in a search tree indexed by eliminator-ranks and maintain the neighbors of high-rank in a more static tree indexed by just their ID (i.e., their name).

Specifically, for each vertex v, we partition its neighborhood (dynamically) in two parts, $N^-(v)$ and $N^+(v)$ as described next. The set $N^-(v)$ includes neighbors of v whose eliminators have smaller rank than the eliminator of v. Each vertex $u \in N^-(v)$ is indexed by the rank of its (dynamically changing) eliminator. The set $N^+(v)$ includes the rest of neighbors of v and every vertex $u \in N^+(v)$ is indexed by its (static) ID. Armed with these data structures it turns out one can implement updates in expected time $\mathrm{polylog}\, n$ per *affected vertex*, i.e., those whose eliminator has changed. (See Lemma 7.4). A key insight behind this analysis is that vertices whose eliminators have small rank are not likely to change their eliminators under many updates, allowing us to keep the cost of reindexing $N^-(v)$ small. Another insight is that the maximum degree in the graph induced on vertices whose eliminators have high ranks is small. Therefore, set $N^+(v)$ will be typically small and the fact that it is not indexed by the rank of its members' eliminators is not troublesome.

Theorem 7.10 and Lemma 7.4 almost settle our analysis, with the former asserting that the expected number of affected nodes is small, and the latter asserting that the expected time to maintain the data structures, per affected node, is small. One final analytic hurdle emerges at this stage though: These two events are not a priori independent and so the product of the expectations is not an upper bound on the expected running time of an update! To overcome this, we introduce another twist in our analysis. Recall that Theorem 7.10 holds even if an entire node is updated (say deleted along with all its edges). When applied to an edge update (a, b), this gives an upper bound of $O(\log n)$ on the expected number of affected vertices even if we condition on any value of $\pi(a)$. (See Lemma 7.11.) The reason, roughly speaking, is that once we condition on $\pi(a)$, the edge update (a, b) can now be regarded as insertion or deletion of vertex b.

Overall, we use the randomization in $\pi(a)$ to bound the expected time per affected vertex by $\operatorname{polylog} n$ and, conditioned on this, still get an $O(\log n)$ upper bound on the expected number of affected vertices due to Lemma 7.11. This allows us to prove an expected $\operatorname{polylog} n$ upper bound on the total running time (see Section 7.5), thus concluding our analysis.

7.2 Some Notation and Basic Tools

We will follow the generic definitions and notation of Chapters 2 and 3. The following is the additional notation we will use throughout this chapter.

As discussed, our approach is based on maintaining the output of RGMIS. In addition to the definitions of Chapter 3, for each vertex v, we define the *eliminator* of v, denoted by $\operatorname{elim}_{G,\pi}(v)$, as the (unique) vertex in $\Gamma_G(v)$ that belongs to $\mathsf{GMIS}(G, \pi)$ and has the lowest rank. This is the first vertex in the greedy construction of $\mathsf{GMIS}(G, \pi)$ adding which to the independent set prevents v from joining it, hence the name eliminator. Note that if v is in the MIS, we have $\operatorname{elim}_{G,\pi}(v) = v$; otherwise, $\operatorname{elim}_{G,\pi}(v) \neq v$ and $\pi(\operatorname{elim}_{G,\pi}(v)) < \pi(v)$. When no confusion is possible, we may write $\operatorname{elim}(v)$ instead of $\operatorname{elim}_{G,\pi}(v)$ for brevity.

As discussed in Chapter 3, instead of a random permutation we draw random ranks on the vertices. It is not hard to see that choosing $\Theta(\log n)$ bit reals is enough to guarantee no two entries assume the same rank w.h.p. From now on, when we use the term "random ranking" π, we indeed assume that each entry of π has $\Theta(\log n)$ bits.

In this chapter, we will use a slightly modified variant of the sparsification property of RGMIS discussed in Section 3.4 stated below.

Lemma 7.3. *Consider a graph $G = (V, E)$, let $\pi : V \to [0, 1]$ be a random ranking, and for any real $p \in [0, 1]$, define V_p as the subset of V including any vertex v with $\pi(\operatorname{elim}_{G,\pi}(v)) > p$. W.h.p., for all $O(\log n)$ bit values of $p \in [0, 1]$, the maximum degree in graph $G[V_p]$ is $O(p^{-1} \cdot \log n)$.*

Proof sketch. The proof is similar to that of Lemma 3.6 except that we union bound over all poly(n) choices of the ranks. □

7.3 Data Structures & The Algorithm

In this section, we present the data structures and the algorithm required for maintaining RGMIS after each update.

We fix a random ranking π in the pre-processing step and maintain $\mathsf{GMIS}(G, \pi)$ after each update. Throughout the rest of this section, we focus on the data structures required for maintaining $\mathsf{GMIS}(G, \pi)$ and the algorithm we use to update them. Fix an arbitrary t and suppose that we have to address edge update number t. We use "time t" to refer to the moment *after* the first t edge updates. Moreover, we use $G_t = (V, E_t)$ to denote the resulting graph at time t. The following definitions are crucial both for the algorithm's description and its analysis.

- $\mathcal{A} := \{v \mid \text{elim}_{G_{t-1},\pi}(v) \neq \text{elim}_{G_t,\pi}(v)\}$: The set of vertices whose eliminator changes after the update; we call these the *affected* vertices.
- $\mathcal{F}$: The set of vertices w that belong to exactly one of $\mathsf{GMIS}(G_t, \pi)$ or $\mathsf{GMIS}(G_{t-1}, \pi)$. We call these the *flipped* vertices. Note that $\mathcal{F} \subseteq \mathcal{A}$.

Our main result in this section is the following algorithm.

Lemma 7.4. *There is an algorithm to update $\mathsf{GMIS}(G, \pi)$ and the data structures required for it after insertion or deletion of any edge $e = (a, b)$ in*

$$O\left(|\mathcal{A}| \min\{\Delta, \frac{\log n}{\min\{\pi(a), \pi(b)\}}\} \log \Delta\right)$$

time w.h.p.

Note that the bound on the update-time in the statement above is parametrized by two random variables $|\mathcal{A}|$ and $\min\{\pi(a), \pi(b)\}$ of the ranking π. To provide a concrete bound on the update-time, we need to analyze how these two random variables are related. We prove the necessary tools for this analysis in Section 7.4 and finally prove that this quantity is in fact polylog n in Section 7.5.

In the rest of this section, we only focus on proving Lemma 7.4. We describe the data structures in Section 7.3.1, describe the algorithm in Section 7.3.2, and prove the correctness and running time of the algorithm in Section 7.3.3.

7.3.1 Data Structures

As described before, our algorithm starts with a pre-processing step where we choose a random ranking π over the n fixed vertices in V, i.e., as discussed, we pick a $\Theta(\log n)$ bit real $\pi(v) \in [0,1]$ for each vertex v. The ranking π will then be used to maintain $\mathsf{GMIS}(G, \pi)$ after each update to graph G. To update this MIS efficiently, we maintain the following data structures for each vertex $v \in V$.

- $m(v)$: A binary variable that is 1 if $v \in \mathsf{GMIS}(G, \pi)$ and 0 otherwise.
- $k(v)$: The rank of v's eliminator, i.e., $k(v) = \pi(\text{elim}_{G,\pi}(v))$. Note that $m(v) = 1$ iff $k(v) = \pi(v)$.
- $N^-(v)$: The set of neighbors u of v where $k(u) \leq k(v)$. The set $N^-(v)$ is stored as a self-balancing binary search tree (BST) and each vertex u in it is indexed by $k(u)$.
- $N^+(v)$: The set of neighbors u of v where $k(u) \geq k(v)$. The set $N^+(v)$ is also stored as a BST, but unlike $N^-(v)$, each member u in $N^+(v)$ is indexed by its ID.

It has to be noted that each vertex $u \in N^-(v)$ is indexed by $k(u)$, a property that may change after an edge update and thus we may need to re-order the vertices in $N^-(v)$. However, the vertices in $N^+(v)$ are simply indexed by their IDs which are static. Also, observe that:

Observation 7.5. *For any two neighbors u and v, $u \in N^+(v)$ if and only if $v \in N^-(u)$.*

Proof. If $u \in N^+(v)$, then $k(u) \geq k(v)$; since $N^-(u)$ includes every neighbor w of u with $k(w) \leq k(u)$, and $k(v) \leq k(u)$, we have $v \in N^-(u)$. Similarly, if $v \in N^-(u)$, then $k(v) \leq k(u)$; since $N^+(v)$ includes every neighbor w of v with $k(w) \geq k(v)$ and $k(u) \geq k(v)$, we have $u \in N^+(v)$. □

From now on, we use $m_t(v)$, $k_t(v)$, $N_t^-(v)$ and $N_t^+(v)$ to respectively refer to data structures $m(v)$, $k(v)$, $N^-(v)$ and $N^+(v)$ by time t. Before describing the algorithm, we describe the pre-processing step in more details.

Pre-processing Step. Apart from choosing random ranking π, we initialize an array $\mathsf{P}v \leftarrow \emptyset$ for every vertex v in the pre-processing step. This array will later be used in the update algorithm in Section 7.3.2. Moreover, we construct greedy MIS over the original graph $G_0 = (V, E_0)$ via the trivial approach: We iterate over the vertices according to π to construct

$\mathsf{GMIS}(G_0, \pi)$ and set $m(v)$ for each vertex v. Then for each vertex v, we iterate over all of its neighbors to fill in $k(v)$, $N^+(v)$, and $N^-(v)$. We initially spend $O(n \log n)$ time for sorting the vertices, then for each vertex v, we spend $O(\deg(v))$ time to fill in its data structures. This process, overall, takes $O((|V|+|E_0|)\log n)$ time which is clearly optimal (up to a logarithmic factor) as it is required to read the input.

7.3.2 The Algorithm

We now turn to describe how we maintain the data structures defined in the previous section after each edge update. Consider update number t, and suppose that an edge $e = (a, b)$ is either inserted or deleted. Moreover, assume w.l.o.g. that $\pi(a) < \pi(b)$. We show how our data structures can be adjusted accordingly in the time specified by Lemma 7.4.

Since we are maintaining random greedy MIS—and not just any MIS—of a dynamically changing graph, a single edge update can potentially affect vertices that are multiple-hops away. To detect these vertices efficiently, we use an iterative approach with which, intuitively, we do not "look" at too many unaffected vertices. Before formalizing this, we start with an observation. The proof is a simple consequence of the structure of greedy MIS and thus we defer it to Section 7.5.2.

Observation 7.6. *For any vertex $v \in \mathcal{A}$, the following properties hold:*

1. *$k_{t-1}(v) \geq \pi(a)$ and $k_t(v) \geq \pi(a)$.*
2. *if $v \neq b$, then v has a neighbor u such that $\pi(u) < \pi(v)$ and $u \in \mathcal{F}$.*

We start with an intuitive and informal description of the algorithm. The algorithm's formal description and the proofs are given afterwards.

Algorithm Outline. Observation 7.6 part 2 implies that if a vertex u is in set $\mathcal{A}$, then there should be a path from vertex b to u where all the vertices in the path (except u) belong to $\mathcal{F}$ and the ranks in the path are monotonically increasing. This motivates us to use an iterative approach. We start by a set $\mathcal{S}$ which originally only includes vertex b. Then we iteratively take the minimum rank vertex v from $\mathcal{S}$, detect whether $v \in \mathcal{F}$ and if so, we add all the "relevant neighbors" of v that may continue these monotone paths to set $\mathcal{S}$. Clearly, we cannot add all neighbors of v to $\mathcal{S}$ since there could be as many as $\Omega(\Delta)$ such nodes. Rather, we only consider neighbors u of v where $k_{t-1}(u) \geq \pi(a)$. Observation 7.6 part 1 guarantees that every vertex $u \in \mathcal{A}$ has $k_{t-1}(u) \geq \pi(a)$ and thus this set of relevant neighbors is sufficient to ensure any vertex in $\mathcal{A}$ will be added to $\mathcal{S}$ at some point. Note that by definition, for any vertex $u \in V \setminus \mathcal{A}$, both $k(u)$ and $m(u)$ will remain unchanged after the update. Therefore, once we handle all vertices in set $\mathcal{S}$, for every vertex u in the graph, $k(u)$ and $m(u)$ should be updated. However, note that the adjacency lists of vertices outside $\mathcal{A}$ may require to be updated if they have a neighbor in $\mathcal{A}$. We do this at the end of the algorithm. Algorithm 11 below formalizes the structure of this algorithm and the subroutines used are formalized afterwards.

We use $k_{t-1}(v)$ and $k_t(v)$ to refer to the value $k(v)$ should hold before and after the update respectively. In the process of updating $k(v)$ from $k_{t-1}(v)$ to $k_t(v)$, whenever we use $k(v)$ without any subscript in the algorithm, we refer to the value of this data structure at that specific time. In particular, since we update the vertices iteratively, it could happen that in a specific time during the algorithm, for some vertex u, $k(u) = k_t(u)$ and for another vertex w, $k(w) = k_{t-1}(w)$. The same notation extends to $m(v), N^+(v)$, and $N^-(v)$ in the natural way.

Algorithm 11: Maintaining data structures after insertion/deletion of $e = (a, b)$.

```
1  $\mathcal{S} \leftarrow \{b\}$
2  For each vertex $v$, we have an array $\mathsf{P}v = \emptyset$.      // Initialized in the pre-processing step.
3  while $\mathcal{S}$ is not empty do
4      Let $v \leftarrow \arg\min_{u \in \mathcal{S}} \pi(u)$ be the minimum rank vertex in $\mathcal{S}$.
5      if IsAffected($v$) then      // Checks whether $v \in \mathcal{A}$ in time $O(|\mathsf{P}v|)$.
6          $\mathcal{H}_v \leftarrow$ FindRelevantNeighbors($v, \pi(a)$)
           // $\mathcal{H}_v$ includes neighbors $u$ of $v$ with $k_{t-1}(u) \geq \pi(a)$ and has size $O(\frac{\log n}{\pi(a)})$ w.h.p.
7          UpdateEliminator($v, \mathcal{H}_v$)      // Updates $k(v)$ and $m(v)$ by iterating over $\mathcal{H}_v$.
8          if $v \in \mathcal{F}$ then
9              for any vertex $u \in \mathcal{H}_v$ do
10                 if $\pi(u) > \pi(v)$ then insert $u$ to $\mathcal{S}$ and insert $v$ to $\mathsf{P}u$.
11     Remove $v$ from $\mathcal{S}$ and set $\mathsf{P}v \leftarrow \emptyset$.
12 UpdateAdjacencyLists()      // Updates adjacency lists $N^+$ and $N^-$ where necessary.
```

We use *iteration* to refer to iterations of the while loop in Algorithm 11. The following invariants hold at the beginning of the algorithm when $\mathcal{S} = \{b\}$ and, as we will show in Claim 7.27 via an induction, will continue to hold throughout.

Invariant 7.7. *Consider the start of any iteration and let v be the lowest-rank vertex in $\mathcal{S}$. It holds true that $k(u) = k_t(u)$ and $m(u) = m_t(u)$ for every vertex u with $\pi(u) < \pi(v)$, i.e., $k(u)$ and $m(u)$ already hold the correct values. Moreover, $k(u) = k_{t-1}(u)$ and $m(u) = m_{t-1}(u)$ for every other vertex u with $\pi(u) \geq \pi(v)$.*

Invariant 7.8. *Consider the start of any iteration and let v be the lowest-rank vertex in $\mathcal{S}$. The set $\mathsf{P}v$ includes a vertex u iff: (1) $\pi(u) < \pi(v)$, and (2) $u \in \mathcal{F}$, and (3) u and v are adjacent.*

Invariant 7.9. *For any vertex u, before reaching Line 12 of Algorithm 11, adjacency lists $N^+(u)$ and $N^-(u)$ respectively hold values $N^+_{t-1}(u)$ and $N^-_{t-1}(u)$.*

We continue by formalizing all the subroutines used in Algorithm 11.

Subroutine $\textsc{IsAffected}(v)$**.** This function returns true if $v \in \mathcal{A}$ and returns false otherwise. We consider two cases where $v = b$ and $v \neq b$ individually. For the former case, we show that $b \in \mathcal{A}$ if and only if $m(a) = 1$ and $k(b) \geq \pi(a)$. For the latter case, we first scan the set $\mathsf{P}v$ to see if there exists a vertex $u \in \mathsf{P}v$ with $\pi(u) = k(v)$. If such vertex u exists, then $v \in \mathcal{A}$. Otherwise, let u be the lowest-rank vertex in $\mathsf{P}v$ such that $m(u) = 1$. If $\pi(u) < k(v)$, then $v \in \mathcal{A}$ and otherwise $v \notin \mathcal{A}$. This subroutine clearly takes $O(|\mathsf{P}v|)$ time. We also prove its correctness in Claim 7.21.

Subroutine $\textsc{FindRelevantNeighbors}(v, \pi(a))$**.** The goal in this subroutine is to find

$$\mathcal{H}_v := \{u \in N(v) \mid k_{t-1}(u) \geq \pi(a)\}. \tag{7.1}$$

By definition of $N^+(v)$ and $N^-(v)$, each neighbor $u \in N(v)$ is at least in one of these two sets. Therefore, to construct set $\mathcal{H}_v$, we have to find neighbors u of v with $k_{t-1}(u) \geq \pi(a)$ in both $N^+(v)$ and $N^-(v)$. For the former, we simply iterate over all neighbors u of v in set $N^+(v)$ and if $k_{t-1}(u) \geq \pi(a)$, we add u to $\mathcal{H}_v$. For the latter, recall from Invariant 7.9 that $N^-(v) = N^-_{t-1}(v)$; thus, the vertices u in $N^-(v)$ are indexed by $k_{t-1}(u)$. To find only those in $N^-(v)$ with $k_{t-1}(u) \geq \pi(a)$, it suffices to search for index $\pi(a)$ and traverse over all vertices whose index is at least $\pi(a)$. The correctness and an analysis of the running time of this algorithm is provided in Claim 7.22.

Subroutine $\textsc{UpdateEliminator}(v, \mathcal{H}_v)$**.** Given that the minimum-rank vertex $v \in \mathcal{S}$ is in set $\mathcal{A}$, this subroutine updates $k(v)$ assuming that the set $\mathcal{H}_v$ is already computed and given. To do this, let u be the lowest-rank vertex in $\mathcal{H}_v$ with $m(u) = 1$. If no such vertex exists, or if $\pi(u) > \pi(v)$, v has to join the MIS and thus we set $k(v) \leftarrow \pi(v)$ and $m(v) \leftarrow 1$. Otherwise, u has to be the new eliminator of v and we set $k(v) \leftarrow \pi(u)$ and $m(v) \leftarrow 0$. This subroutine clearly takes $O(|\mathcal{H}_v|)$ time. We also prove its correctness in Claim 7.23.

Subroutine UPDATEADJACENCYLISTS(). If e is deleted, we remove a from $N^+(b)$ and $N^-(b)$, and remove b from $N^+(a)$ and $N^-(a)$ (note that some of these sets may not include the removing vertex). If e is inserted, we insert a and b into each other's "appropriate" adjacency list according to the current values of $k(a)$ and $k(b)$; namely:

- If $k(a) < k(b)$, insert a into $N^-(b)$, and insert b into $N^+(a)$.
- If $k(a) > k(b)$, insert a into $N^+(b)$, and insert b into $N^-(a)$.
- If $k(a) = k(b)$, insert a into $N^-(b)$ and $N^+(b)$, and insert b into $N^-(a)$ and $N^+(a)$.

We also need to update the adjacency lists of any affected vertex v, since after changing $k(v)$, some neighbors of v may have to move from $N^+(v)$ to $N^-(v)$ or vice versa. Moreover, if an affected vertex v is in $N^-(u)$ of some vertex u, we also need to recompute the position of v in $N^-(u)$, since recall that v should be indexed by $k(v)$ in $N^-(u)$ which has now changed.

To address the changes above, the crucial property is that for any vertex $v \in \mathcal{A}$, any vertex u that has to move between $N^+(v)$ and $N^-(v)$ or has v in its set $N^-(u)$, has to belong to $\mathcal{H}_v$ (see Claim 7.24 for the proof). Therefore in the algorithm, for any vertex $v \in \mathcal{A}$, we only iterate over the vertices $u \in \mathcal{H}_v$ and based on $k(u)$ and $k(v)$, which at this point in the algorithm are correctly updated, determine the membership of vertex v in adjacency lists of vertex u and vice versa. We then update $N^-(v)$, $N^+(v)$, $N^-(u)$ and $N^+(u)$ accordingly.

7.3.3 Overview of Correctness & The (Parametrized) Running Time

The correctness of Algorithm 11 follows mainly from the greedy structure of RGMIS and does not require a sophisticated analysis. As such, we defer it to Section 7.5.2. Here, we focus on the main ideas required for bounding the running time of the algorithm stated in Lemma 7.4. A complete proof of this lemma is also presented in Section 7.5.2.

One particularly important property is that, w.h.p., the size of set $\mathcal{H}_v$ for every vertex $v \in \mathcal{A}$ is $O\big(\min\{\Delta, \frac{\log n}{\pi(a)}\}\big)$. This is formally proved in Claim 7.22 of Section 7.5.2; but the main intuition is as follows. From definition of $\mathcal{H}_v$, every vertex $u \in \mathcal{H}_v$ has $k_{t-1}(u) \geq \pi(a)$. Moreover, since $v \in \mathcal{A}$, by Observation 7.6 part 1, we also have $k_{t-1}(v) \geq \pi(a)$. This means that if we construct GMIS in graph G_{t-1} on the prefix of vertices with rank in $[0, \pi(a))$, then vertex v will survive and will have a remaining degree of at least $|\mathcal{H}_v|$. Since the adversary is oblivious and the ranking π and graph G_{t-1} are independently chosen, we can use Lemma 7.3 to argue that in this remaining graph, maximum degree is, w.h.p., at most $O\big(\min\{\Delta, \frac{\log n}{\pi(a)}\}\big)$ implying the same upper bound on $|\mathcal{H}_v|$.

Observe that in the algorithm, only for vertices $v \in \mathcal{F}$ we insert (a subset of) their relevant neighbors $\mathcal{H}_v$ to $\mathcal{S}$. Therefore, the total number of vertices inserted to $\mathcal{S}$ is at most $O\big(|\mathcal{F}| \min\{\Delta, \frac{\log n}{\pi(a)}\}\big)$, w.h.p. However, this is not an upper bound on the algorithm's running time since each vertex in $\mathcal{S}$ is not simply processed in constant time. We summarize these procedures below.

Subroutine $\textsc{IsAffected}(v)$**.** This subroutine is called for every vertex $v \in \mathcal{S}$. It is clear from description that $\textsc{IsAffected}(v)$ takes $O(|\mathsf{P}v|)$ time. Therefore, the aggregated running time of this function for all vertices in $\mathcal{S}$ is $\sum_{v \in S} |\mathsf{P}v|$. Observe that each vertex $u \in \mathsf{P}v$ is in $\mathcal{F}$. Furthermore, each vertex $u \in \mathcal{F}$ belongs to $\mathsf{P}v$ of at most $|\mathcal{H}_u|$ vertices due to Line 10. Therefore, a simple double-counting argument shows that w.h.p., $\sum_{v \in S} |\mathsf{P}v| \leq O\big(|\mathcal{F}| \min\{\Delta, \frac{\log n}{\pi(a)}\}\big)$.

Subroutine $\textsc{FindRelevantNeighbors}(v, \pi(a))$**.** This is called for every vertex $v \in \mathcal{A}$. Thanks to the fact that $N^-(v)$ is indexed by $k_{t-1}(.)$ and that $N^+(v)$ has size at most $O(|\mathcal{H}_v|)$ (we show this in the proof of Claim 7.22) this subroutine takes $O(|\mathcal{H}_v| \log \Delta)$ time where the extra $\log \Delta$ factor is for iterating over BST $N^-(v)$. Thus, the aggregated running time is $O\big(|\mathcal{A}| \min\{\Delta, \frac{\log n}{\pi(a)}\} \log \Delta\big)$.

Subroutine $\textsc{UpdateEliminator}(v, \mathcal{H}_v)$**.** This subroutine is only called for vertices $v \in \mathcal{A}$ and takes $O(|\mathcal{H}_v|)$ time. Clearly, the total running time is $O\big(|\mathcal{A}| \min\{\Delta, \frac{\log n}{\pi(a)}\}\big)$, w.h.p.

Subroutine $\textsc{UpdateAdjacencyLists}()$**.** As described in the subroutine, for any vertex $v \in \mathcal{A}$, v has to be re-indexed or moved in adjacency lists of at most $|\mathcal{H}_v|$ of its neighbors. Each such operation requires $O(\log \Delta)$ time. Therefore, the aggregated running time is w.h.p. $O\big(|\mathcal{A}| \min\{\Delta, \frac{\log n}{\pi(a)}\} \log \Delta\big)$.

The total running time of the algorithm is the sum of the aggregated running time of each of the procedures above which is $O\big(|\mathcal{A}| \min\{\Delta, \frac{\log n}{\pi(a)}\} \log \Delta\big)$ as required by Lemma 7.4.

7.4 An Analysis of Affected Vertices: Proof of Theorem 7.10

In this section, we prove Theorem 7.10 which we briefly highlighted in Section 7.1. In this regard, for any two graphs $G = (V, E)$ and $G' = (V', E')$ with $V' \subseteq V$ and a ranking π over V, we define $\mathcal{A}_\pi(G, G') := \{v \in V \mid \mathrm{elim}_{G,\pi}(v) \neq \mathrm{elim}_{G',\pi}(v)\}$ to be the set of vertices with different eliminators in the two graphs. Note that this is analogous to the definition of "affected vertices" in the previous section and hence the choice of notation. A more formal statement of Theorem 7.10 reads as follows:

Theorem 7.10. *Fix an arbitrary graph $G = (V, E)$ and let $G' = G[V \setminus \{v\}]$ be obtained by removing an arbitrary vertex v from G. If π is a random ranking over V, $\mathbf{E}_{\pi}[|\mathcal{A}_{\pi}(G, G')|] = O(\log n)$.*

The fact that Theorem 7.10 bounds the number of affected vertices as a result of a vertex update can be used to bound the affected vertices by $O(\log n)$ as a result of an edge update $e = (a, b)$, *even when we condition on any value for* $\min\{\pi(a), \pi(b)\}$.

Lemma 7.11. *Fix an arbitrary graph $G = (V, E)$ and let $G' = (V, E')$ be the graph obtained by adding or removing an arbitrary edge $e = (a, b)$ to G. If π is a random ranking over V, then for any value of $\lambda \in [0, 1]$, it holds that $\mathbf{E}_{\pi}[|\mathcal{A}_{\pi}(G, G')| \mid \min\{\pi(a), \pi(b)\} = \lambda] \leq O(\log n)$.*

Lemma 7.11 is crucial for our analysis as it implies that the two random variables $|\mathcal{A}_{\pi}(G, G')|$ and $\min\{\pi(a), \pi(b)\}$, which recall are used in the statement of Lemma 7.4, can be regarded as "almost" independent. We elaborate more on this in Section 7.5.

We first prove Lemma 7.11 given the correctness of Theorem 7.10. The bulk of analysis is then concentrated around proving Theorem 7.10.

Proof of Lemma 7.11. Suppose w.l.o.g. that $\pi(a) < \pi(b)$, i.e., $\pi(a) = \lambda$ by the conditional event. Let U be the subset of V containing vertices w with $\pi(w) \leq \pi(a)$. We prove the lemma even when the set U, and the rank of the vertices in it are chosen adversarially.

We have $G'[U] = G[U]$ since $u \notin U$ and the only difference of the two graphs G' and G which is in edge (a, b) does not belong to either of the two induced graphs. Therefore, we have $\mathsf{GMIS}(G'[U], \pi) = \mathsf{GMIS}(G[U], \pi)$; let I_U be this MIS. Furthermore, let $H'(V'_H, E'_H)$ and $H(V_H, E_H)$ be the residual graphs after we remove vertices in I_U and their neighbors from G' and G respectively. It is not hard to see that either $H = H'$ (if $a \notin I_U$ or if b has another neighbor w with $\pi(w) < \pi(a)$ in I_U) or H' has exactly one extra vertex than H which has to be b, i.e., $H = H'[V'_H \setminus \{b\}]$. In the former case, since the two graphs are equal, no matter how π is chosen, the eliminators of all vertices will be the same. In the latter case, the two graphs H and H' differ in only one vertex and no information about the relative order of the vertices in V_H or V'_H in π is revealed. Thus, by Theorem 7.10, the expected number of vertices whose eliminators are different in H and H' is $O(\log n)$. $\square$

We now, turn to prove Theorem 7.10 and start with some notation. Throughout the rest of this section, vertex v should be regarded as fixed. We use I and I' to respectively denote independent sets $\mathsf{GMIS}(G, \pi)$ and $\mathsf{GMIS}(G', \pi)$. Also, for brevity, we use $\mathcal{A}_\pi$ instead of $\mathcal{A}_\pi(G, G')$. Furthermore, we define $\mathcal{F}_\pi$ as the subset of vertices in $\mathcal{A}_\pi$ whose MIS-status is <u>f</u>lipped, i.e., $u \in \mathcal{F}_\pi$ if and only if u belongs to exactly one of I or I'.

Instead of rankings, it will be more convenient to consider random permutations instead of random ranks for the arguments of this section, recalling from Chapter 3 that there is no difference in the distribution of the two.

The following observation is very similar to Observation 7.6 of the previous section and will be very useful here too.

Observation 7.12. *If $\mathcal{A}_\pi$ is non-empty, then $v \in I$ and $v \in \mathcal{F}_\pi$. Furthermore, for every vertex $u \in \mathcal{A}_\pi \setminus \{v\}$, there is another vertex $w \in \mathcal{F}_\pi$ that is adjacent to u and $\pi(w) < \pi(u)$.*

Proof. We first prove that if $\mathcal{A}_\pi \neq \emptyset$ then $v \in I$. Assume for contradiction that $v \notin I$ and $\mathcal{A}_\pi \neq \emptyset$. Since v does not belong to G', we also have $v \notin I'$, i.e., v is in neither of the two maximal independent sets I and I'. Now take the minimum rank vertex u in $\mathcal{A}_\pi$ (which exists since $\mathcal{A}_\pi \neq \emptyset$). Since $u \in \mathcal{A}_\pi$, by definition, its eliminators should be different in I and I'. Therefore, there should exist a vertex w with $\pi(w) < \pi(u)$, that is in exactly one of the two maximal independent sets. Since v is in neither of I and I', $w \neq v$. However, in this case, w would also belong to $\mathcal{A}_\pi$, contradicting that u is the minimum rank vertex in $\mathcal{A}_\pi$, and completing the proof of this part.

For the second part, fix a vertex $u \in \mathcal{A}_\pi$ and let x and x' be its eliminators in I and I' respectively. Note that x and x' cannot be the same vertex or otherwise $u \notin \mathcal{A}_\pi$. Suppose that $\pi(x) < \pi(x')$. The fact that x is an eliminator of u in I means that $x \in I$. On the other hand, the fact that x', instead of x, is the eliminator of u in I' means that $x \notin I'$. This means that x has to belong to $\mathcal{F}_\pi$. A similar argument holds for the case where $\pi(x') < \pi(x)$. $\square$

For a vertex $u \in \mathcal{A}_\pi \setminus \{v\}$, we define the *parent* of u, denoted by $p_\pi(u)$, as its neighbor in $\mathcal{F}_\pi$ (which exists by observation above) with the lowest rank, i.e., $p_\pi(u) = \arg\min_{w \in N(u) \cap \mathcal{F}_\pi} \pi(w)$. Furthermore, we define the *propagation path* $P_\pi(u)$ of each vertex $u \in \mathcal{A}_\pi$ as:

$$P_\pi(u) = \begin{cases} (v) & \text{if } u = v, \\ (P_\pi(p_\pi(u)), u) & \text{otherwise.} \end{cases}$$

With a slight abuse of notation, $P_\pi(u)$ can be denoted by a sequence $(w_1, \ldots, w_k)$ where $w_1 = v$, $w_k = u$, and for every $i \in [k-1]$, $w_i = p_\pi(w_{i+1})$. Note that this sequence is a valid path of the graph because by definition each vertex is a neighbor of its parent and $\pi(p_\pi(u))$ is strictly smaller than $\pi(u)$ by Observation 7.12, thus, no vertex can be visited twice in the sequence. Furthermore, $w_1 = v$ because every vertex $w \in \mathcal{A}_\pi$ has a parent $p_\pi(w)$ except v.

Claim 7.13. *Fix an arbitrary permutation π, an arbitrary vertex $u \in \mathcal{A}_\pi$, and let $P_\pi(u) = (w_1, \ldots, w_k)$. For odd $i \in [k-1]$, $w_i \in \mathsf{GMIS}(G, \pi)$ and for even $i \in [k-1]$, $w_i \notin \mathsf{GMIS}(G, \pi)$.*

Proof. Since $u \in \mathcal{A}_\pi$ and thus $\mathcal{A}_\pi \neq \emptyset$, we already know from Observation 7.12 that vertex $v = w_1$ has to belong to $\mathsf{GMIS}(G, \pi)$, proving the claim for $i = 1$. To complete the proof, we show that for any $i \in [k-2]$, exactly one of w_i and w_{i+1} is in $\mathsf{GMIS}(G, \pi)$.

First, observe that since $\mathsf{GMIS}(G, \pi)$ is an independent set, no two adjacent vertices can belong to it. Therefore, we only have to show that for any $i \in [k-2]$, it cannot be the case that neither of w_i and w_{i-1} are in $\mathsf{GMIS}(G, \pi)$. Suppose for contradiction that this holds. By definition of propagation paths, and since $i \in [k-2]$, we get that w_i is the parent of w_{i+1} and w_{i+1} is the parent of w_{i+2}. Every vertex that is a parent of another vertex has to be in $\mathcal{F}_\pi$ by definition. Therefore, both w_i and w_{i+1} belong to $\mathcal{F}_\pi$. Combined with the assumption that neither of w_i and w_{i+1} are in $\mathsf{GMIS}(G, \pi)$, both have to belong to $\mathsf{GMIS}(G', \pi)$ (by definition of $\mathcal{F}_\pi$) which cannot be possible since $\mathsf{GMIS}(G', \pi)$ is also an independent set. $\square$

Let Π denote the set of all permutations over V. We say a permutation $\pi \in \Pi$ is *unlikely*, if for some vertex $u \in V$, $|P_\pi(u)| > \beta \log n$ where β is a constant that we fix later, and *likely* otherwise. Denoting the set of likely and unlikely permutations by Π_L and Π_U respectively, we have

$$\mathop{\mathbf{E}}_{\pi}[|\mathcal{A}_\pi|] = \mathbf{Pr}[\pi \in \Pi_L] \cdot \mathop{\mathbf{E}}_{\pi \sim \Pi_L}[|\mathcal{A}_\pi|] + \mathbf{Pr}[\pi \in \Pi_U] \cdot \mathop{\mathbf{E}}_{\pi \sim \Pi_U}[|\mathcal{A}_\pi|]. \tag{7.2}$$

We prove $\mathbf{E}_\pi[|\mathcal{A}_\pi|] = O(\log n)$ by bounding the two terms in (7.2) individually.

Lemma 7.14 ([60, 89])**.** *If β is a large enough constant, $\mathbf{Pr}[\pi \in \Pi_U] \leq n^{-2}$.*

Lemma 7.15 (likely permutations)**.** $\mathbf{E}_{\pi \sim \Pi_L}[|\mathcal{A}_\pi|] = O(\log n)$.

Lemma 7.14 almost directly follows from the earlier works of [60, 89] on bounding parallel round complexity of RGMIS; we provide the details in Section 7.4.3. Lemma 7.15, which is proven in Section 7.4.1, constitutes the novel part of the proof and is indeed where bulk of the whole analysis is concentrated on. Below, we first show why Lemmas 7.15 and 7.14 are sufficient to prove Theorem 7.10.

Proof of Theorem 7.10. By Lemma 7.15, we have $\mathbf{E}_{\pi\sim\Pi_L}[|\mathcal{A}_\pi|] = O(\log n)$. Since $\mathbf{Pr}[\pi \in \Pi_L] \leq 1$ for being a probability, we get $\mathbf{Pr}[\pi \in \Pi_L] \cdot \mathbf{E}_{\pi\sim\Pi_L}[|\mathcal{A}_\pi|] \leq O(\log n)$, i.e., the first term in (7.2) is bounded by $O(\log n)$. On the other hand, by Lemma 7.14, we have $\mathbf{Pr}[\pi \in \Pi_U] \leq n^{-2}$. Using this, we can bound the second term in (7.2) to be as small as n^{-1} even if $\mathcal{A}_\pi$ includes all n vertices for any $\pi \in \Pi_U$. Therefore overall, we get $\mathbf{E}_\pi[|\mathcal{A}_\pi|] \leq O(\log n) + n^{-1} = O(\log n)$, which is the desired bound. □

7.4.1 Handling Likely Permutations: Proof of Lemma 7.15

In the rest of this section, we focus on proving Lemma 7.15. The overall plan is as follows. For each permutation $\pi \in \Pi_L$, we *blame* a set of permutations $B(\pi) \subseteq \Pi$ such that:

(P1) $|B(\pi)| \geq |\mathcal{A}_\pi|$.

(P2) For each permutation $\pi' \in \Pi$, there are at most $\beta \log n$ permutations $\pi \in \Pi_L$ where $\pi' \in B(\pi)$.

We first prove that having such blaming sets satisfying properties P1 and P2 is sufficient for proving Lemma 7.15 and then describe how the blaming sets are constructed.

Proof of Lemma 7.15. Defining X as the sum $\sum_{\pi\in\Pi_L} |\mathcal{A}_\pi|$, we have:

$$\mathop{\mathbf{E}}_{\pi\sim\Pi_L}[|\mathcal{A}_\pi|] = \sum_{\pi\in\Pi_L} \mathbf{Pr}[\text{drawing } \pi \mid \pi \in \Pi_L] \cdot |\mathcal{A}_\pi| = \frac{1}{|\Pi_L|} \sum_{\pi\in\Pi_L} |\mathcal{A}_\pi| = \frac{X}{|\Pi_L|}. \tag{7.3}$$

By property P1, $|B(\pi)| \geq |\mathcal{A}_\pi|$ for every $\pi \in \Pi_L$. Thus,

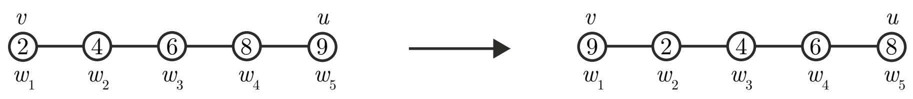
$$Y := \sum_{\pi\in\Pi_L} |B(\pi)| \geq \sum_{\pi\in\Pi_L} |\mathcal{A}_\pi| = X.$$

On the other hand, since by property P2, each permutation $\pi' \in \Pi$ belongs to $B(\pi)$ of at most $\beta \log n$ other permutations π, a simple double counting argument gives $Y \leq |\Pi|\beta \log n$; implying also that $X \leq |\Pi|\beta \log n$. Moreover, since $\Pi_L = \Pi \setminus \Pi_U$ and by Lemma 7.14, $\frac{|\Pi_U|}{|\Pi|} < n^{-2}$, it holds that $\frac{|\Pi_L|}{|\Pi|} > 1 - n^{-2}$, thus, $|\Pi| = O(|\Pi_L|)$. For this, $X \leq |\Pi|\beta \log n$ implies $X = O(|\Pi_L| \log n)$. Plugging this into (7.3) gives $\mathbf{E}_{\pi\sim\Pi_L}[|\mathcal{A}_\pi|] = \frac{O(|\Pi_L| \log n)}{|\Pi_L|} = O(\log n)$. □

For every permutation $\pi \in \Pi_L$, and each vertex $u \in \mathcal{A}_\pi$, we construct a permutation $\varphi_{\pi,u} \in \Pi$. The blaming set of π will then be the set $B(\pi) = \bigcup_{u\in\mathcal{A}_\pi} \{\varphi_{\pi,u}\}$. For a vertex $u \in \mathcal{A}_\pi$, with $P_\pi(u) = (v = w_1, w_2, \ldots, w_k = u)$, we construct permutation $\varphi_{\pi,u}$ as follows:

- For each vertex $w \notin P_\pi(u)$, $\varphi_{\pi,u}(w) \leftarrow \pi(w)$.
- $\varphi_{\pi,u}(w_1) \leftarrow \pi(w_k)$.
- For any $2 \leq i \leq k$, $\varphi_{\pi,u}(w_i) \leftarrow \pi(w_{i-1})$.

In other words, permutation $\varphi_{\pi,u}$ on all vertices outside $P_\pi(u)$ is exactly the same as π, however for the vertices in $P_\pi(u)$, $\varphi_{\pi,u}$ is obtained by rotating the π ranks by one index towards u. An example is shown in the figure below.



As captured by the following observation, it is not hard to show that with this construction, property P1 is indeed satisfied:

Observation 7.16. *By construction above, property P1 is satisfied.*

Proof. The reason is that we indeed construct $|\mathcal{A}_\pi|$ permutations to include in $B(\pi)$: $\varphi_{\pi,u}$ for each $u \in \mathcal{A}_\pi$. Note, however, that we still need to argue that for any two vertices u and w in $\mathcal{A}_\pi$, permutations $\varphi_{\pi,u}$ and $\varphi_{\pi,w}$ are not the same so that the set containing them has size $|\mathcal{A}_\pi|$. This follows because $\varphi_{\pi,w}(v) = \pi(w)$ and $\varphi_{\pi,u}(v) = \pi(u)$ but $\pi(u) \neq \pi(w)$, implying that $\varphi_{\pi,w}(v) \neq \varphi_{\pi,u}(v)$ and thus the two permutations $\varphi_{\pi,w}$ and $\varphi_{\pi,u}$ are not equal. □

The harder part is to show that our construction also satisfies property P2:

Claim 7.17. *By construction above, property P2 is also satisfied.*

Suppose that a permutation ρ is blamed by permutations π and π', i.e., $\rho \in B(\pi) \cap B(\pi')$. This means that there should exist vertices $u \in \mathcal{A}_\pi$ and $u' \in \mathcal{A}_{\pi'}$ where $\varphi_{\pi,u} = \varphi_{\pi',u'} = \rho$. To prove Claim 7.17, we analyze the circumstances under which this may occur. Consider the propagation paths $P_\pi(u) = (w_1, w_2, \ldots, w_k)$ and $P_{\pi'}(u') = (w'_1, w'_2, \ldots, w'_{k'})$ and recall that $w_k = u$, $w'_{k'} = u'$ and $w_1 = w'_1 = v$. Let j be the largest integer where for any $i \in \{1, \ldots, j\}$, we have $w_i = w'_i$. Note that clearly $j \geq 1$ since $w'_1 = w_1 = v$. We call w_j (or equivalently w'_j) *the branching vertex* and analyze the following scenarios which cover all possibilities individually (see Figure 7.1):

- **Scenario 1:** j is odd, $w_j \neq u$, and $w_j \neq u'$.
- **Scenario 2:** j is even, $w_j \neq u$, and $w_j \neq u'$.

- **Scenario 3:** at least one of u or u' is the same as w_j.

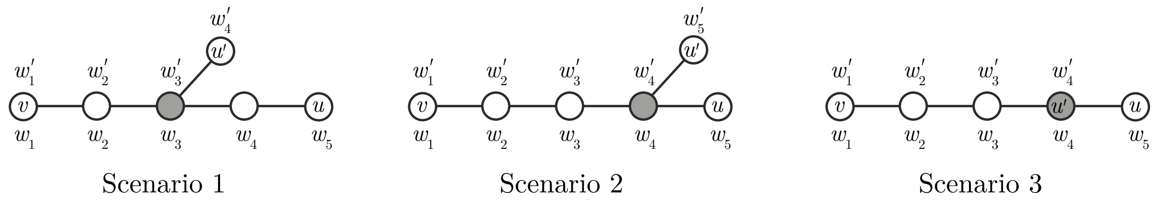


Figure 7.1: The grey vertex in each scenario, denotes the corresponding branching vertex w_j.

The claim below unveils several important structural properties of propagation paths and will be our main tool to prove Claim 7.17. See Figure 7.2 for an illustration of some of these properties.

Claim 7.18. *Consider two different permutations π and π' in Π_L and two (possibly the same) vertices u and u'. Let w_j be the branching vertex for propagation paths $P_\pi(u) = (w_1, \ldots, w_k)$ and $P_{\pi'}(u') = (w'_1, \ldots, w'_{k'})$. If $\varphi_{\pi,u} = \varphi_{\pi',u'}$ and $\pi'(w_j) \geq \pi(w_j)$, then:*

1. *for every vertex w that does not belong to either of $P_\pi(u)$ and $P_{\pi'}(u')$, $\pi(w) = \pi'(w)$.*
2. *$\pi(w) = \pi'(w)$ for every vertex w with $\pi(w) < \pi(w_j)$.*
3. *$\pi(w_k) = \pi'(w'_{k'})$.*
4. *$k \geq j + 1$ (i.e., vertex w_{j+1} should exist) and $\pi'(w_{j+1}) = \pi(w_j)$.*
5. *$w_{j+1} \in \mathsf{GMIS}(G, \pi')$.*

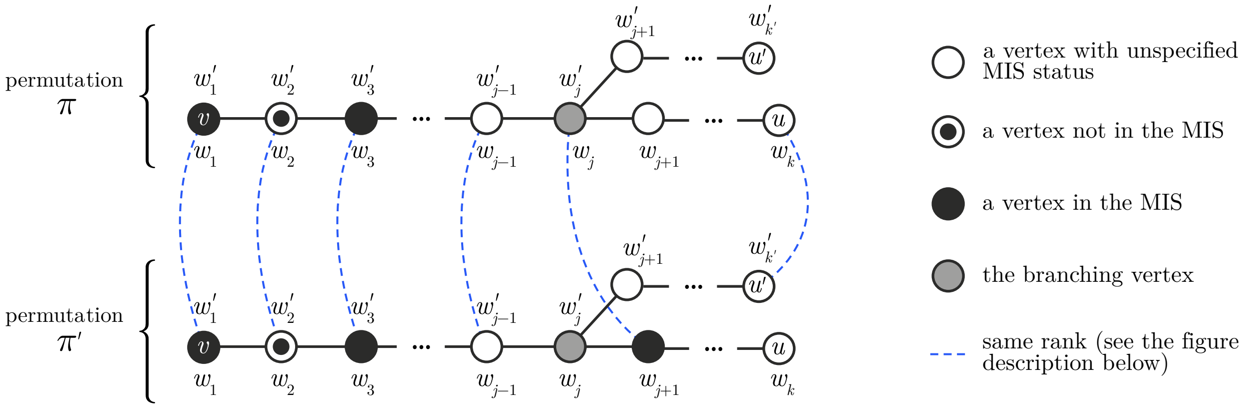


Figure 7.2: Illustration of some of the properties obtained from Claims 7.13 and 7.18 for vertices in $P_\pi(u) = (w_1, \ldots, w_k)$ and $P_{\pi'}(u') = (w'_1, \ldots, w'_{k'})$ given that $\varphi_{\pi,u} = \varphi_{\pi',u'}$ and $\pi'(w_j) \geq \pi(w_j)$ where w_j is the branching vertex. A dashed line between vertices x on the top and y on the bottom implies $\pi(x) = \pi'(y)$. Note that for the illustration purpose, this figure models only scenarios 1 and 2; however, Claims 7.13 and 7.18 are general and hold for all three scenarios.

We first show how these properties suffice to prove Claim 7.17, then prove Claim 7.18.

Proof of Claim 7.17. Suppose that a permutation ρ is blamed by two permutations π and π', and let u and u' be the vertices where $\varphi_{\pi,u} = \varphi_{\pi',u'}$ (we note that u and u' may be the same vertex). We show that if these assumptions hold, then scenarios 1 and 2 defined above would lead to contradictions, implying that scenario 3 is the only case for which this may occur. We assume w.l.o.g. that $\pi'(w_j) \geq \pi(w_j)$ so that all conditions of Claim 7.18 are satisfied.

Scenario 1. Since in this scenario $w_j \neq u'$, we get $j < k'$; more precisely, $j \in [k'-1]$. Furthermore, recall that j is assumed to be odd in scenario 1. Combining the two conditions, Claim 7.13 implies $w_j \in \mathsf{GMIS}(G, \pi')$. On the other hand, by Claim 7.18 part 5, $w_{j+1} \in \mathsf{GMIS}(G, \pi')$. However, this is a contradiction since by definition of $P_\pi(u)$, $w_j = p_\pi(w_{j+1})$ and thus w_j and w_{j+1} are neighbors; meaning that they both cannot belong to independent set $\mathsf{GMIS}(G, \pi')$.

Scenario 2. The assumption $w_j \neq u'$ implies that there is a vertex w'_{j+1} and that $w_j = p_{\pi'}(w'_{j+1})$; thus by definition of $p_{\pi'}(w'_{j+1})$, $w_j \in \mathcal{F}_{\pi'}$. It also implies that $j \in [k'-1]$ (as argued in scenario 1). But since j is even in this scenario, Claim 7.13 implies $w_j \notin \mathsf{GMIS}(G, \pi')$. Let us use H to denote the graph $G[V \setminus \{v\}]$ obtained by removing vertex v from G. Recall that by definition, w_j is in $F_{\pi'}$ iff its MIS-status is different in $\mathsf{GMIS}(G, \pi')$ and $\mathsf{GMIS}(H, \pi')$. Therefore, since $w_j \notin \mathsf{GMIS}(G, \pi')$ we have to have $w_j \in \mathsf{GMIS}(H, \pi')$. This also implies that $w_{j+1} \notin \mathsf{GMIS}(H, \pi')$ since as argued in scenario 1, w_j and w_{j+1} are neighbors in G and thus H. On the other hand, similar to scenario 1, we should have $w_{j+1} \in \mathsf{GMIS}(G, \pi')$ by Claim 7.18 part 5. Therefore, since w_{j+1} has a different MIS-status in $\mathsf{GMIS}(G, \pi')$ and $\mathsf{GMIS}(H, \pi')$, we have $w_{j+1} \in \mathcal{F}_{\pi'}$. By definition, $\mathcal{F}_{\pi'} \subseteq \mathcal{A}_{\pi'}$ thus by Observation 7.12,

$$\text{there exists a vertex } x \in N_H(w_{j+1}) \text{ such that } x \in \mathcal{F}_{\pi'} \text{ and } \pi'(x) < \pi'(w_{j+1}).$$

Furthermore, by Claim 7.18 part 4, $\pi'(w_{j+1}) = \pi(w_j)$; combined with $\pi'(x) < \pi'(w_{j+1})$, this implies $\pi'(x) < \pi(w_j)$. Observe that by Claim 7.18 part 2 the two permutations π and π' are exactly the same on the set of vertices with rank less than $\pi(w_j)$. Therefore, $x \in \mathsf{GMIS}(G, \pi')$ iff $x \in \mathsf{GMIS}(G, \pi)$, and $x \in \mathsf{GMIS}(H, \pi')$ iff $x \in \mathsf{GMIS}(H, \pi)$. As a result, $x \in \mathcal{F}_{\pi'}$ implies that $x \in \mathcal{F}_\pi$.

Finally, recall that $p_\pi(w_{j+1})$ is by definition the lowest-rank neighbor of w_{j+1} in $\mathcal{F}_\pi$. Therefore, since $x \in \mathcal{F}_\pi$ and $\pi(x) < \pi(w_j)$, we have $p_\pi(w_{j+1}) \neq w_j$. This contradicts the definition of $P_\pi(u)$ which guarantees $w_j = p_\pi(w_{j+1})$.

As shown above, the only case for which we might get $\varphi_{\pi,u} = \varphi_{\pi',u'}$ is scenario 3 as the other two scenarios lead to contradictions. We now show that because of the very specific structure of scenario 3, each permutation is blamed by at most $\beta \log n$ permutations.

Fix a permutation ρ and let C_ρ be a set that includes every pair (π, u) with $\pi \in \Pi_L$ and $u \in V$ for which $\varphi_{\pi,u} = \rho$. Clearly, $|C_\rho|$ is an upper bound on the number of permutations that blame ρ, thus it suffices to bound $|C_\rho|$ by $\beta \log n$.

First, we show that for any two different pairs (π, u) and (π', u') in C_ρ, we have $|P_\pi(u)| \neq |P_{\pi'}(u')|$. Suppose for the sake of contradiction that $|P_\pi(u)| = |P_{\pi'}(u')|$. Let $P_\pi(u) = (w_1, \ldots, w_k)$ and $P_{\pi'}(u') = (w'_1, \ldots, w'_k)$ be the vertices in the two paths and let w_j be the branching vertex. We know that $\varphi_{\pi,u} = \varphi_{\pi',u'} = \rho$ since the pairs belong to C_ρ. Therefore, scenario 3 has to occur and thus either $w_j = u'$ or $w_j = u$. In either case, we get $j = k$ since $u = w_k$ and $u' = w'_k$. Furthermore, by definition of the branching vertex we have $w_i = w'_i$ for any $i \in [j]$. Moreover, by Claim 7.18 parts 2 and 3, for any $i \in [j]$, we have $\pi(w_i) = \pi'(w'_i)$. Meaning that the set of vertices and their ranks in the two permutations are exactly the same on the propagation paths. On the other hand, for any vertex x that does not belong to the propagation paths, we also have $\pi(x) = \pi'(x)$ due to Claim 7.18 part 1. Combining these, we get that $\pi = \pi'$. We also showed that $w_k = w'_k$ and thus $u = u'$. Therefore, the two pairs (π, u) and (π', u') are identical, which is in contradiction with our initial assumption that they are different.

Now we show that $|C_\rho| \leq \beta \log n$. Suppose for contradiction that there are at least $\beta \log n + 1$ pairs in C_ρ. As shown in the previous paragraph, for each pair $(\pi, u) \in C_\rho$, $|P_\pi(u)|$ is unique. Therefore, if $|C_\rho| > \beta \log n$, there should be at least a pair (π, u) with $|P_\pi(u)| \geq \beta \log n + 1$. However, by definition, the propagation-path of every vertex in every permutation $\pi \in \Pi_L$, has size at most $\beta \log n$ which is a contradiction. Therefore, $|C_\rho| \leq \beta \log n$ for any permutation ρ, thus every permutation ρ is blamed by at most $\beta \log n$ other permutations. This means that property P2 is also satisfied by our mapping, as desired. □

7.4.2 The Mapping's Structural Properties: Proof of Claim 7.18

In what follows, we prove the parts of Claim 7.18 one by one. Note that the proof of each part may depend on the correctness of the previous parts.

Proof of Claim 7.18 part 1. For any vertex w that does not belong to the propagation paths $P_\pi(u)$ and $P_{\pi'}(u')$, we have $\varphi_{\pi,u}(w) = \pi(w)$ and $\varphi_{\pi',u'}(w) = \pi'(w)$ by construction of permutations $\varphi_{\pi,u}$ and $\varphi_{\pi',u'}$; hence, to have $\varphi_{\pi,u} = \varphi_{\pi',u'}$ it should hold that $\pi(w) = \pi'(w)$. □

Proof of Claim 7.18 part 2. Consider a vertex w with $\pi(w) < \pi(w_j)$, we prove $\pi(w) = \pi'(w)$.

Case 1: $w \notin P_\pi(u)$ and $w \notin P_{\pi'}(u')$. In this case, by Claim 7.18 part 1 we have $\pi(w) = \pi'(w)$.

Case 2: $w \in P_\pi(u)$. By definition of propagation-path $P_\pi(u)$, we have $\pi(w_1) < \ldots < \pi(w_k)$. Therefore, since $w \in P_\pi(u)$ and $\pi(w) < \pi(w_j)$, we should have $w = w_i$ for some $i < j$. Since w_j is the branching vertex, this means $w_{i+1} = w'_{i+1}$ and $w_i = w'_i$. By construction of $\varphi_{\pi,u}$ and $\varphi_{\pi',u'}$, we have $\varphi_{\pi,u}(w_{i+1}) = \pi(w_i)$ and $\varphi_{\pi',u'}(w'_{i+1}) = \pi'(w'_i)$. Combined with $w_{i+1} = w'_{i+1}$, $w_i = w'_i$, and $\varphi_{\pi,u} = \varphi_{\pi',u'}$, this means $\pi(w_i) = \pi'(w_i)$.

Case 3: $w \notin P_\pi(u)$ and $w \in P_{\pi'}(u')$. We show that it is essentially impossible to satisfy the property's condition $\pi(w) < \pi(w_j)$ in this case, implying that the property holds automatically. First, observe that $w = w'_i$ should hold for some $i > j$, or otherwise $w \in P_\pi(u)$ by definition of the branching vertex w_j. By construction of $\varphi_{\pi',u'}$, this implies $\varphi_{\pi',u'}(w) \geq \pi'(w'_j) = \pi'(w_j)$. Combined with assumption $\pi'(w_j) \geq \pi(w_j)$, we get $\varphi_{\pi',u'}(w) \geq \pi(w_j)$. Moreover, since $w \notin P_\pi(u)$, we get $\varphi_{\pi,u}(w) = \pi(w)$. Thus, to have $\varphi_{\pi,u} = \varphi_{\pi',u'}$, it should hold that $\pi(w) = \varphi_{\pi',u'}(w)$. Since we just showed $\varphi_{\pi',u'}(w) \geq \pi(w_j)$, this would imply $\pi(w) \geq \pi(w_j)$ which as outlined at the start of this case, is sufficient for our purpose.

The cases above clearly cover all possibilities; thus the proof is complete. □

Proof of Claim 7.18 part 3. Recall that $w_1 = w'_1 = v$. We have $\varphi_{\pi,u}(v) = \pi(w_k)$ and $\varphi_{\pi',u'}(v) = \pi(w'_{k'})$ simply by construction of these permutations. Therefore, to have $\varphi_{\pi,u}(v) = \varphi_{\pi',u'}(v)$, we should have $\pi(w_k) = \pi'(w'_{k'})$. □

Proof of Claim 7.18 part 4. Suppose for contradiction that $k \leq j$, i.e., vertex w_{j+1} does not exist. Since w_j is the branching vertex, it has to belong to $P_\pi(u)$ by definition, thus, $k \geq j$. Combined with $k \leq j$, the only possibility would be $k = j$. By Claim 7.18 part 3, we have $\pi(w_k) = \pi'(w'_{k'})$ and since $j = k$, we get

$$\pi(w_j) = \pi'(w'_{k'}). \tag{7.4}$$

On the other hand, by definition of $P_{\pi'}(u')$, we have

$$\pi'(w'_{k'}) > \pi'(w'_{k'-1}) \ldots > \pi'(w'_1). \tag{7.5}$$

Moreover, recall from the claim's assumption that $\pi'(w_j) \geq \pi(w_j)$. Combining this with (7.4) and (7.5), the only option is if $j = k'$. To see this, observe that $j \leq k'$ by definition of the branching vertex; now, if $j < k'$, then from (7.5) we obtain $\pi'(w'_{k'}) > \pi'(w_j)$ which due to (7.4) would imply $\pi(w_j) > \pi'(w_j)$ contradicting the claim's assumption that $\pi(w_j) \leq \pi'(w_j)$; thus, $j = k'$. Recall that we also assumed $j = k$ at the beginning of the proof, therefore $j = k = k'$. This implies by definition of the branching vertex that $w_i = w'_i$ for any $i \in [k]$ (or equivalently $[k']$), i.e., the two paths $P_\pi(u)$ and $P_{\pi'}(u')$ are exactly the same. Moreover, due to $j = k = k'$ and Claim 7.18 parts 2 and 3, for any vertex w_i in the propagation paths, $\pi(w_i) = \pi'(w_i)$. On the other hand, for any vertex x outside the two paths, we have $\pi(x) = \pi'(x)$ by Claim 7.18 part 1. Therefore, overall, the two permutations π and π' have to be exactly the same on all vertices, which is a contradiction with the claim's assumption that π and π' are different. Therefore, our initial assumption that $k \leq j$ cannot hold and vertex w_{j+1} should exist.

Finally, by construction of $\varphi_{\pi,u}$, we have $\varphi_{\pi,u}(w_{j+1}) = \pi(w_j)$. Now, since $w_{j+1} \notin P_{\pi'}(u')$ (otherwise w_{j+1} would be be the branching vertex instead of w_j), we have $\varphi_{\pi',u'}(w_{j+1}) = \pi'(w_{j+1})$. From $\varphi_{\pi,u} = \varphi_{\pi',u'}$, we get $\varphi_{\pi,u}(w_{j+1}) = \varphi_{\pi',u'}(w_{j+1})$. Combining these three equalities, we get $\pi(w_j) = \pi'(w_{j+1})$ as desired. □

Proof of Claim 7.18 part 5. Suppose for the sake of contradiction that $w_{i+1} \notin \mathsf{GMIS}(G, \pi')$ and let $x := \text{elim}_{G,\pi'}(w_{j+1})$ be the eliminator of w_{j+1} in $\mathsf{GMIS}(G, \pi')$. Since $w_{j+1} \notin \mathsf{GMIS}(G, \pi')$, it holds that $\pi'(x) < \pi'(w_{i+1})$. Moreover, by Claim 7.18 part 4, $\pi'(w_{j+1}) = \pi(w_j)$; combined with inequality $\pi'(x) < \pi'(w_{j+1})$, this implies that $\pi'(x) < \pi(w_j)$. Note also that, by Claim 7.18 part 2, the two permutations π and π' are exactly the same on the set of vertices with rank less than $\pi(w_j)$; since x is among such vertices,

$$\pi(x) = \pi'(x) < \pi(w_j). \tag{7.6}$$

Another implication of the equivalence of the two permutations on vertices with rank less than $\pi(w_j)$ is that since $x \in \mathsf{GMIS}(G, \pi')$ (which holds since x is the eliminator of w_{j+1} in $\mathsf{GMIS}(G, \pi')$) we also have $x \in \mathsf{GMIS}(G, \pi)$. This in turn, implies that x is the eliminator of w_{j+1} in $\mathsf{GMIS}(G, \pi)$ as well. On the other hand, since w_{j+1} is a vertex in path $P_\pi(u)$, by definition of the propagation-paths, it should hold that $w_{j+1} \in \mathcal{A}_\pi$. Moreover, by definition of $\mathcal{A}_\pi$, we have $\text{elim}_{G,\pi}(w_{j+1}) \neq \text{elim}_{G',\pi}(w_{j+1})$ where G' is defined as $G[V \setminus \{v\}]$. Denoting $\text{elim}_{G',\pi}(w_{j+1})$ by y and noting that $x = \text{elim}_{G,\pi}(w_{j+1})$, we get $y \neq x$. Therefore, one of the following cases should occur:

Case 1: $\pi(y) < \pi(x)$. In this case, the fact that x is the eliminator of w_{j+1} in $\mathsf{GMIS}(G, \pi)$ even though $\pi(y) < \pi(x)$ means $y \notin \mathsf{GMIS}(G, \pi)$. On the other hand, $y \in \mathsf{GMIS}(G', \pi)$ since $y = \text{elim}_{G',\pi}(w_{j+1})$, therefore $y \in \mathcal{F}_\pi$ by definition. However, this contradicts $w_j = p_\pi(w_{j+1})$ since by (7.6), $\pi(y) < \pi(x) < \pi(w_j)$, thus, y should be the parent of w_{j+1} instead of w_j.

Case 2: $\pi(y) > \pi(x)$. Similarly, in this case, the fact that x is not the eliminator of w_{j+1} in $\mathsf{GMIS}(G', \pi)$ even though $\pi(x) < \pi(y)$ implies that $x \notin \mathsf{GMIS}(G', \pi)$. This means that $x \in \mathcal{F}_\pi$ and again, since $\pi(x) < \pi(w_j)$, x has to be the parent of w_{j+1} instead of w_j.

To wrap up, $w_{j+1} \notin \mathsf{GMIS}(G, \pi')$ leads to a contradiction, thus $w_{j+1} \in \mathsf{GMIS}(G, \pi')$. □

7.4.3 Unlikely Permutations: Proof of Lemma 7.14

RGMIS can be parallelized in the following way. In each round, each vertex that holds the minimum rank among its neighbors joins the MIS and then is removed from the graph along with its neighbors (note that this, in parallel, happens for several vertices in each round). Fischer and Noever [89], building on an earlier approach of Blelloch, Fineman, and Shun [60], showed that if permutation π is chosen randomly, with probability at least $1 - n^{-2}$, it takes $O(\log n)$ rounds until the graph becomes empty.[4] This result as a black-box does not prove Lemma 7.14. However, to prove this upper-bound on round-complexity, they indeed upper bound the maximum size of *dependency-paths* which are structures that are very close to propagation-paths:

Definition 7.19 ([89]). *A path $w_1, w_2, \ldots, w_k$ in the graph is a dependency-path according to permutation π, if for any odd $i \in [k]$, vertex w_i is in $\mathsf{GMIS}(G, \pi)$ and for any even $i \in [k]$, $w_i \notin \mathsf{GMIS}(G, \pi)$ and $w_{i-1} = \text{elim}_{G,\pi}(w_i)$.*

Recall that indeed, if $u_1, \ldots, u_k$ is a propagation-path, then for every $i \in [k-1]$, $u_i = \text{elim}_{G,\pi}(u_{i+1})$ by definition. Moreover, by Claim 7.13, except for the last vertex in the propagation-path, the odd vertices are in the MIS and the even vertices are not. Therefore:

Observation 7.20. *If there exists a propagation-path of size ℓ in the graph, then its first $\ell - 1$ vertices form a dependency-path.*

[4]We note that the success probability of these works is actually $1 - n^{-c}$ for any desirable constant $c > 1$ affecting the hidden constants in the round-complexity. For our purpose, $c = 2$ is sufficient.

Fischer and Noever [89] prove that with probability $1 - n^{-2}$, every dependency-path has size $O(\log n)$ if π is chosen at random. Therefore, from Observation 7.20, we get that the probability of having a propagation-path with size $\beta \log n$, if β is a large enough constant, is at most n^{-2}, which completes the proof of Lemma 7.14.

7.5 Fully Dynamic MIS: Putting Everything Together

7.5.1 The (Concrete, Non-Parametrized) Running Time

In this section, we show how combining Lemma 7.4 with Lemma 7.11 proves the main claim of this chapter that MIS can be maintained in polylogarithmic update-time.

Theorem 7.1 (restated)**.** *There is a data structure to maintain an MIS against an oblivious adversary in a fully-dynamic graph that, per update, takes $O(\log^2 \Delta \cdot \log^2 n)$ expected time. Furthermore, the number of adjustments to the MIS per update is $O(1)$ in expectation.*

Proof of Theorem 7.1. Consider insertion or deletion of an edge $e = (a, b)$. As before, we use λ to denote random variable $\min\{\pi(a), \pi(b)\}$ and use $\mathcal{A}$ to denote the set of vertices whose eliminators change as a result of this edge update. By Lemma 7.4, we have

$$\begin{aligned}\mathbf{E}[\text{update-time for an edge } e = (a, b)] &= \mathbf{E}\left[O\left(|\mathcal{A}| \cdot \log \Delta \cdot \min\left\{\lambda^{-1} \cdot \log n, \Delta\right\}\right)\right] \\ &= O(\log \Delta) \cdot \mathbf{E}\left[|\mathcal{A}| \cdot \min\left\{\lambda^{-1} \cdot \log n, \Delta\right\}\right] \\ &= O(\log \Delta \cdot \log n) \cdot \mathbf{E}\left[\min\left\{\lambda^{-1} \cdot \log n, \Delta\right\}\right]. \qquad (7.7)\end{aligned}$$

The third equation follows from $\mathbf{E}[|\mathcal{A}| \mid \lambda] \leq O(\log n)$ which was proved in Lemma 7.11, combined with the fact that if for two possibly dependent random variables y_1 and y_2, $\mathbf{E}[y_1|y_2] \leq \beta$, then $\mathbf{E}[y_1 \cdot y_2] \leq \beta\, \mathbf{E}[y_2]$. To bound the random variable inside the expectation, suppose we partition the $[0, 1]$ interval into Δ sub-intervals $I_1, \ldots, I_\Delta$ where $I_i = [\frac{i-1}{\Delta}, \frac{i}{\Delta}]$ for any $i \in [\Delta]$. Note that if $\lambda \in I_i$ then at least one of $\pi(a)$ and $\pi(b)$ is in I_i. Therefore, a simple union bound implies that $\mathbf{Pr}[\lambda \in I_i] \leq \mathbf{Pr}[\pi(a) \in I_i] + \mathbf{Pr}[\pi(b) \in I_i] = 2/\Delta$. We, thus, have:

$$\begin{aligned}\mathbf{E}\left[\min\left\{\lambda^{-1} \cdot \log n, \Delta\right\}\right] &= \sum_{i=1}^{\Delta} \mathbf{Pr}[\lambda \in I_i] \cdot \mathbf{E}\left[\min\left\{\lambda^{-1} \cdot \log n, \Delta\right\} \mid \lambda \in I_i\right] \\ &\leq \sum_{i=1}^{\Delta} \frac{2}{\Delta}\left(\min\left\{\frac{\Delta}{i-1} \log n, \Delta\right\}\right) = O(\log n) \sum_{i=1}^{\Delta} \frac{1}{i} = O(\log \Delta \cdot \log n).\end{aligned}$$

Replacing this into (7.7) suffices to bound the expected update-time by $O(\log^2 \Delta \cdot \log^2 n)$. Furthermore, as mentioned before in Section 7.1, we already know from [64, Theorem 1] that the expected adjustment complexity of RGMIS is $O(1)$, completing the proof. $\square$

7.5.2 Deferred Proofs

We start by proving Observation 7.6 which is crucial for the algorithm's correctness.

Observation 7.6 (restated)**.** *For any vertex $v \in \mathcal{A}$, the following properties hold:*

1. *$k_{t-1}(v) \geq \pi(a)$ and $k_t(v) \geq \pi(a)$.*
2. *if $v \neq b$, then v has a neighbor u such that $\pi(u) < \pi(v)$ and $u \in \mathcal{F}$.*

Proof of Observation 7.6 part 1. Let U denote the set of vertices v in V with $\pi(v) < \pi(a)$. Observe that the two induced subgraphs $G_t[U]$ and $G_{t-1}[U]$ are identical since the only difference between G_t and G_{t-1} is insertion/deletion of edge $e = (a, b)$ whose endpoints both have rank at least $\pi(a)$ (recall that $\pi(a) < \pi(b)$) and thus neither belongs to U. Since the MIS is constructed greedily on lower rank vertices first, the set of MIS vertices in $G_t[U]$ and $G_{t-1}[U]$ according to π are exactly the same. Let I_U denote these MIS nodes. Note that any vertex v with $k_{t-1}(v) < \pi(a)$ should have a neighbor in I_U. Since both end-points of edge e are in $V \setminus U$, the set of neighbors of I_U in both graphs G_t and G_{t-1} are also identical. Therefore for each vertex v with $k_{t-1}(v) < \pi(a)$, we have $k_t(v) = k_{t-1}(v)$ and thus v cannot be in $\mathcal{A}$ by definition. By a similar argument, for any vertex v with $k_t(v) < \pi(a)$ we also have $k_{t-1}(v) = k_t(v)$ and thus $v \notin \mathcal{A}$. □

Proof of Observation 7.6 part 2. The assumption $v \in \mathcal{A}$ implies that the eliminator of v has changed after the update. Let w be the eliminator of v before the update. If the MIS-status of no neighbor u of v with $\pi(u) \leq \pi(w)$ changes, since $v \neq b$ and the set of neighbors of v are the same before and after the update, then w remains to be the eliminator of v. Therefore, to have $v \in \mathcal{A}$, the MIS-status of at least one of v's neighbors changes and this vertex is in $\mathcal{F}$ by definition. □

We first prove the correctness of each of the subroutines and then that of the overall algorithm. These subroutines are proven to be correct by the end of any iteration i conditioned on the assumption that Invariants 7.7-7.9 (or a subset of them) hold at the start of iteration i. We later inductively prove that these invariants hold and that indeed the whole algorithm is correct.

Claim 7.21. *By the end of any iteration i, subroutine* IsAffected(v) *correctly decides whether the lowest-rank vertex $v \in \mathcal{S}$ is in set $\mathcal{A}$ in time $O(|\mathsf{P}v|)$ given that Invariants 7.7-7.9 hold by the start of iteration i.*

Proof. The algorithm clearly takes $O(|\mathsf{P}v|)$ time since it only iterates over the vertices in $\mathsf{P}v$ to decide on the output. In what follows, we prove its correctness. As in the algorithm's description, we consider two cases where $v = b$ and $v \neq b$ individually.

Case 1 $v = b$. In this case, the algorithm decides $b \in \mathcal{A}$ if and only if $m(a) = 1$ and $k(b) \geq \pi(a)$. We show that this is indeed correct.

The if part. We show that if $m(a) = 1$ and $k(b) \geq \pi(a)$, then $b \in \mathcal{A}$. Observe from Invariant 7.7 that at this point in the algorithm, we have $k(b) = k_{t-1}(b)$. Therefore, the $k(b) \geq \pi(a)$ assumption implies $k_{t-1}(b) \geq \pi(a)$. Moreover, the MIS-status of vertex a cannot change as it is the lower-rank vertex of the updated edge, thus, it holds that $m_t(a) = m_{t-1}(a)$ and consequently $m(a) = 1$ implies $a \in \mathsf{GMIS}(G_{t-1}, \pi)$. Combining these, the eliminator of b has to be a iff there is an edge between a and b. Therefore, updating edge $e = (a, b)$ definitely changes b's eliminator and thus $b \in \mathcal{A}$.

The only if part. Suppose that one of the conditions do not hold, we show $b \notin \mathcal{A}$. First, if $k < \pi(a)$, then by Observation 7.6 part 1, $b \notin \mathcal{A}$ as desired. Moreover, if $m(a) = 0$, as before, we should have $m_{t-1}(a) = m_t(a) = 0$ since a is the lower-rank vertex of the update. As a result, insertion or deletion of e cannot have an effect on the eliminator of b and thus $b \notin \mathcal{A}$.

Case 2 $v \neq b$. In this case, the eliminator of v changes if and only if at least one of the following conditions hold: (1) the eliminator of v in time $t-1$ leaves the MIS, (2) at least a vertex u adjacent to v with $\pi(u) < k(v)$ joins the MIS. If none of these conditions hold, then $\text{elim}_{G_{t-1},\pi}(v)$ remains to be the smallest-rank vertex in $\{b\} \cup N(b)$ that is in the MIS after the update; therefore by definition of eliminator, $k_{t-1}(v) = k_t(v)$ and thus $v \notin \mathcal{A}$.

Our algorithm precisely checks these conditions. For condition (1), if the eliminator $u := \text{elim}_{G_{t-1},\pi}(v)$ leaves the MIS after the update, it should by definition belong to $\mathcal{F}$. Note that by invariant 7.8, $\mathsf{P}\text{v}$ exactly contains the neighbors w of v with $w \in \mathcal{F}$ and $\pi(w) < \pi(v)$. Therefore if $u \in \mathsf{P}v$, then condition (1) holds and $v \in \mathcal{A}$. Our algorithm also checks condition (2) by finding the lowest-rank vertex w in $\mathsf{P}\text{v}$ with $m(w) = 1$ and then comparing its rank with $k_{t-1}(v)$. □

Claim 7.22. *At any iteration i, with probability at least $1-n^{-(c+1)}$, $\mathcal{H}_v$ has size $O(\min\{\Delta, \frac{\log n}{\pi(a)}\})$. Furthermore, subroutine* FindRelevantNeighbors$(v, \pi(a))$ *correctly finds the set $\mathcal{H}_v$ in time $O(|\mathcal{H}_v| \cdot \log \Delta)$, given that Invariant 7.9 holds by the start of iteration i.*

Proof. **Size of $\mathcal{H}_v$:** Observe that if $\mathcal{H}_v$ is defined, then as assured by the condition in Line 5 of Algorithm 11, $v \in \mathcal{A}$ thus by Observation 7.6, $k_{t-1}(v) \geq \pi(a)$. Furthermore, by definition, every vertex $u \in \mathcal{H}_v$ has $k_{t-1}(u) \geq \pi(a)$. This means that if we take GMIS of G_{t-1} induced on vertices with rank in $[0, \pi(a))$ and remove them and their neighbors from the graph, v and all of its neighbors in $\mathcal{H}_v$ will survive. Recall that the adversary is oblivious and the graph G_{t-1} and random permutation π are chosen independently. Therefore, applying Lemma 7.3 on graph G_{t-1} with $p = \pi(a)$ bounds $|\mathcal{H}_v|$ by $O(\pi(a)^{-1} \log n)$ w.h.p. Moreover, clearly $|\mathcal{H}_v| \leq \Delta$ since they are neighbors of v, concluding the bound on the size of $\mathcal{H}_v$.

Correctness: The assumption that Invariant 7.9 holds implies that $N^-(v) = N^-_{t-1}(v)$ and $N^+(v) = N^+_{t-1}(v)$. Therefore, FindRelevantNeighbors$(v, \pi(a))$ correctly finds $\mathcal{H}_v$.

Running time: Since the vertices $u \in N^-(v)$ are indexed by $k_{t-1}(u)$ and the algorithm iterates only over the neighbors u of v in this set with $k_{t-1}(u) \geq \pi(a)$, the running time of this part is $O(|\mathcal{H}_v| \log \Delta)$ where the $\log \Delta$ factor comes from searching in this BST which has size Δ at most. However, note that the algorithm iterates over all vertices in $N^+(v)$ since it is not indexed by $k_{t-1}(.)$. Therefore, we have to prove $|N^+(v)|$ cannot be larger than $|\mathcal{H}_v|$. We know from Invariant 7.9 that for any vertex $u \in N^+(v)$, we have $k_{t-1}(u) \geq k_{t-1}(v)$. Moreover, since $v \in \mathcal{A}$, by Observation 7.6, $k_{t-1}(v) \geq \pi(a)$. Combining the two, we get that $k_{t-1}(u) \geq \pi(a)$. This means that every vertex $u \in N^+(v)$ that is still a neighbor of v after the update, should be in set $|\mathcal{H}_v|$. Since at most one edge is removed from the graph at time t, we have $|N^+(v)| \leq |\mathcal{H}_v| + 1$, completing the proof. □

Claim 7.23. *Let v be the lowest-rank vertex at the start of an arbitrary iteration. Subroutine* UpdateEliminator$(v, \mathcal{H}_v)$ *correctly updates $k(v)$ and $m(v)$ of vertex v in time $O(|\mathcal{H}_v|)$ assuming that Invariant 7.7 holds by this iteration.*

Proof. It is clear that the algorithm takes $O(|\mathcal{H}_v|)$ time, here we prove its correctness. Note that at the time of using subroutine UpdateEliminator$(v, \mathcal{H}_v)$, we know $v \in \mathcal{A}$. Therefore, from Observation 7.6 part 1, we know $k_t(v) \geq \pi(a)$ and $k_{t-1}(v) \geq \pi(a)$. We consider the two cases where $m_t(v) = 1$ and $m_t(v) = 0$ differently.

Suppose that $m_t(v) = 0$ and let w be the eliminator of v after the update, i.e., $\pi(w) = k_t(v)$ (note that since $m_t(v) = 0$, $w \neq v$). We first show $w \in \mathcal{H}_v$ by proving that $k_{t-1}(w) \geq \pi(a)$. Suppose for contradiction that $k_{t-1}(w) < \pi(a)$. Then by Observation 7.6 part 1, $w \notin \mathcal{A}$ and consequently $w \notin \mathcal{F}$ since $\mathcal{F} \subseteq \mathcal{A}$. Since w is the eliminator of v in G_t, we have $m_t(w) = 1$. Moreover, for $w \notin \mathcal{F}$, we also get $m_{t-1}(w) = 1$ which, by definition, means w has to be its own eliminator in G_{t-1} and thus $k_{t-1}(w) = \pi(w)$. Combined with $k_{t-1}(w) < \pi(a)$, this would mean $\pi(w) < \pi(a)$. This, however, contradicts $k_{t-1}(v) \geq \pi(a)$ since v has a neighbor w in MIS of G_{t-1} with rank smaller than $\pi(a)$ and thus it should hold that $k_{t-1}(v) < \pi(a)$. This contradiction implies that indeed $k_{t-1}(w) \geq \pi(a)$ and thus $w \in \mathcal{H}_v$. Furthermore, in this case, since $\pi(w) < \pi(v)$, by Invariant 7.7, $m(w) = m_t(w) = 1$ and indeed the lowest-rank vertex u in $\mathcal{H}_v$ with $m(u) = 1$ should be vertex w and the algorithm is correct.

On the other hand, if $m_t(v) = 1$, then no lower-rank neighbor of v should be in the MIS. In this case, once we scan the set $\mathcal{H}_v$, we will not find any vertex u with a lower-rank than $\pi(v)$ and $m(u) = 1$, thus we correctly decide that v is in the MIS and update $m(v)$ and $k(v)$ correctly. □

Claim 7.24. *Subroutine* UpdateAdjacencyLists() *correctly updates the adjacency lists and with probability at least* $1 - n^{-c}$*, takes* $O(|\mathcal{A}| \cdot \min\{\Delta, \frac{\log n}{\pi(a)}\} \cdot \log \Delta)$ *time given that for any vertex* v*,* $k(v) = k_t(v)$*.*

Proof. The only edge update is between vertices a and b and the algorithm first accordingly addresses this change by updating $N^+(a)$, $N^-(a)$, $N^+(b)$, and $N^-(b)$. For the rest of the vertices, we do not have an edge update but the changes to the adjacency lists are resulted by the changes to the eliminators. For a vertex v, these changes are limited to moving its neighbors between $N^+(v)$ and $N^-(v)$ or possibly re-indexing its neighbors in $N^-(v)$ whose eliminator has changed.

We say an edge (v, u) causes an update iff position of u and v or their indexing in each others' adjacency lists (N^+ or N^-) needs to be updated. Let T denote the set of these edges. Note that by definition of N^+ and N^-, if $u \notin \mathcal{A}$ and $v \notin \mathcal{A}$, then $(v, u) \notin T$. This means that at least one end-point of any edge in T is in $\mathcal{A}$.

Āssume w.l.o.g. that for edge $(v, u) \in T$, we have $v \in \mathcal{A}$. We claim that $u \in \mathcal{H}_v$ should hold. To show this, we assume that $u \notin \mathcal{H}_v$ and obtain a contradiction. Recall that we have $u \notin \mathcal{H}_v$ iff $k(u) < \pi(a)$. By Observation 7.6 part 1, this would imply $k_{t-1}(u) < k_{t-1}(v)$, $k_t(u) < k_t(v)$, and $u \notin \mathcal{A}$. Because of $k_{t-1}(u) < k_{t-1}(v)$ and $k_t(u) < k_t(v)$, the position of vertices u and v in each others adjacency lists remains unchanged. That is, we have $v \in N^+(u)$, $v \notin N^-(u)$, $u \in N^-(v)$, and $u \notin N^-(v)$ at both times t and $t - 1$. Moreover, since $u \notin \mathcal{A}$, we have $k_{t-1}(u) = k_t(u)$ and thus u is already correctly indexed in $N^-(v)$. This is, however, a contradiction since position of u and v and their indexing in each others' adjacency lists is already updated and as a result $(v, u) \notin T$. Therefore, it should indeed hold that $u \in \mathcal{H}_v$.

In subroutine UpdateAdjacencyLists(), for any vertex $v \in \mathcal{A}$ we go over its neighbors $u \in \mathcal{H}_v$ and determine the membership of vertex v in adjacency lists of vertex u and vice versa. To do so, by definition of N^+ and N^- we only need values of $k_t(v)$ and $k_t(u)$ which are assumed to be updated (in the statement of the claim). We then update $N^-(v)$, $N^+(v)$, $N^-(u)$ and $N^+(u)$ accordingly; thus the algorithm correctly updates the adjacency lists.

To analyze the running time, using Claim 7.22, we know that for any vertex $v \in \mathcal{A}$, set $\mathcal{H}_v$ has size $O(\min\{\Delta, \frac{\log n}{\pi(a)}\})$ with probability at least $1 - n^{-(c+1)}$. Also, each update takes $O(\log \Delta)$ time since it consists of at most four insertions and deletions in adjacency lists which are stored as BSTs. Overall, this means that the running time can be bounded by $O(|\mathcal{A}| \cdot \min\{\Delta, \frac{\log n}{\pi(a)}\} \cdot \log \Delta)$ with probability at least $1 - n^{-c}$. □

Claim 7.25. *If Invariant 7.7 holds by some iteration i, then Invariant 7.8 also holds by iteration i.*

Proof. Let u be any vertex adjacent to v with $\pi(u) < \pi(v)$ and $u \in \mathcal{F}$. In other words, any vertex that should be in set $\mathsf{P}v$ for the Invariant 7.8 to hold. Assuming that Invariant 7.7 holds, we know that $m(u) = m_t(u)$ and $m(u) \neq m_{t-1}(u)$. Observe that in the algorithm, updating $m(u)$ only happens in subroutine UpdateEliminator$(v, \mathcal{H}_v)$ which is followed by adding u to set P. of any vertex in set $\mathcal{H}_u$ if u is flipped. Set $\mathcal{H}_u$ by definition includes vertex v since $k(v) \geq \pi(a)$ and $\pi(v) > \pi(u)$. This proves that set $\mathsf{P}v$ satisfies Invariant 7.8. □

Claim 7.26. *Let v be the lowest-rank vertex in $\mathcal{S}$ in an arbitrary iteration i of the algorithm. Assuming that Invariant 7.7 holds at the start of iteration i we have:*

1. *If $\mathcal{S} = \emptyset$ at the end of iteration i, for any vertex $u \in V$, $m(u) = m_t(u)$ and $k(u) = k_t(u)$.*
2. *If $\mathcal{S} \neq \emptyset$ at the end of iteration i, then Invariant 7.7 holds at the start of iteration $i+1$ as well.*

Proof. Let $\mathcal{S}'$ denote set $\mathcal{S}$ at the end of iteration i and let v' be the lowest-rank vertex in that. Throughout the proof, by $\mathcal{S}$ we mean set $\mathcal{S}$ at the start of iteration i and we use v to refer to its lowest-rank vertex. Let us first review Invariant 7.7. It states that for any vertex u, if $\pi(u) < \pi(v)$ then $k(u) = k_{t-1}(u)$, and $m(u) = m_{t-1}(u)$ hold and otherwise we have $k(u) = k_{t-1}(u)$, and $m(u) = m_{t-1}(u)$. We first show that $k(v)$ and $m(v)$ are updated at the end of iteration i. By Claim 7.25, we know that Invariant 7.8 holds at the start of iteration i and by Claim 7.21, we know that holding Invariant 7.8 means that subroutine IsAffected(v) correctly detects if $v \in \mathcal{A}$ or not. Moreover, by Claim 7.23 if $v \in \mathcal{A}$, in the next step, algorithm correctly updates $k(v)$ and $m(v)$. At this point of the algorithm, we know that for any vertex u with $\pi(u) \leq \pi(v)$, we have $k(u) = k_t(u)$ and $m(u) = m_t(u)$.

Now, let u be the vertex with the lowest-rank among the vertices in $\mathcal{A}$ whose rank is greater than $\pi(v)$. To complete the proof it suffices if we show that if such a vertex exists, then $u \in \mathcal{S}'$. This means that if $S' = \emptyset$, then for any vertex $u \in V$, we have $m(u) = m_t(u)$ and $k(u) = k_t(u)$. Moreover, for the case of $S' \neq \emptyset$, it results that for any vertex u, with $\pi(u) < \pi(v')$ we have $m(u) = m_t(u)$ and $k(u) = k_t(u)$ or in the other words that Invariant 7.7 holds at the start of iteration $i + 1$. We use proof by contradiction by assuming that there exists a vertex u in set $\mathcal{A}$ but not in S' such that for any vertex u' with $\pi(u') < \pi(u)$ we have $m(u') = m_t(v')$, and $k(u') = k_t(u')$. By Observation 7.12, any vertex in $\mathcal{A}$ has a neighbor in $\mathcal{F}$ with a lower rank. Let u' be such a neighbor of u. By the assumption that all neighbors of u with a lower rank has updated $m(.)$, we have $m(u') \neq m_{t-1}(u')$. Observe that in the algorithm, updating $m(u')$ only happens in subroutine UpdateEliminator$(v, \mathcal{H}_{u'})$ which is followed by adding vertices in $\mathcal{H}_u$ to $\mathcal{S}$. Set $\mathcal{H}_u$, by definition, includes vertex u since $k(u) \geq \pi(a)$ (otherwise by Observation 7.6, $u \notin \mathcal{A}$) and $\pi(u) > \pi(u')$. Thus, we obtain a contradiction and the proof is completed. □

Claim 7.27. *Invariants 7.7, 7.8, and 7.9 hold throughout the algorithm with probability 1.*

Proof. First, observe that Invariant 7.9 holds since Line 12 is the only part of the algorithm that we modify the adjacency lists. Moreover, by Claim 7.25, the correctness of Invariant 7.8 results from Invariant 7.8. Thus, we only need to show that Invariants 7.7 holds throughout the algorithm. We do so using induction. As the base case, in the first iteration of the algorithm we have $S = \{b\}$ (or $S = \emptyset$ which does not need a proof). We need to show that for any vertex u if $\pi(u) < \pi(b)$ we have $k(u) = k_t(u)$, and $m(u) = m_t(u)$ and if $\pi(u) > \pi(b)$ we have $k(u) = k_{t-1}(u)$, and $m(u) = m_{t-1}(u)$. Before the start of this iteration we have not changed $k(u)$ and $m(u)$ of any vertex u thus for all of them $k(u) = k_{t-1}(u)$ and $m(u) = m_{t-1}(u)$. Moreover, by Observation 7.12, updating edge e does not affect a vertex u with $\pi(u) < \pi(b)$ which means that for any such vertex we have $k_t(u) = k_{t-1}(u)$. Therefore, we conclude that Invariants 7.7 holds for the base case. This completes the proof since the induction step is a direct result of Claim 7.26. □

We continue with a simple observation and then turn to formally prove the runtime.

Observation 7.28. *Let v_i and v_j respectively denote the lowest-rank vertices of $\mathcal{S}$ in two arbitrary iterations i and j of Algorithm 11. If $i < j$ then $\pi(v_i) < \pi(v_j)$.*

Proof. We show that this claim holds for $j = i+1$ which can be inductively used to generalize it to any arbitrary i and j. Let $\mathcal{S}_i$ and $\mathcal{S}_{i+1}$ respectively denote set $\mathcal{S}$ at the beginning of iteration i and set $\mathcal{S}$ at the beginning of iteration $i+1$. We know that v_{i+1} is either inserted to $\mathcal{S}$ in iteration i or that it is in set $\mathcal{S}_i$. Observe that any vertex added to $\mathcal{S}$ in the i-th iteration has rank lower than $\pi(v_i)$ and that v_i is the lowest-rank vertex in $\mathcal{S}_i$. As a result $\pi(v_i) < \pi(v_{i+1})$. $\square$

Claim 7.29. *With probability at least $1 - n^{-c}$, the total running time of the algorithm until the set $\mathcal{S}$ becomes empty is at most $O(|\mathcal{A}| \cdot \log \Delta \cdot \min\{\frac{\log n}{\pi(a)}, \Delta\})$.*

Proof. To prove this claim, we first show that $|\mathcal{S}|$ and $\sum_{v \in \mathcal{S}} |\mathsf{P}v|$ are both $O(|\mathcal{A}| \cdot \log \Delta \cdot \min\{\frac{\log n}{\pi(a)}, \Delta\})$ with probability at least $1 - n^{-c}$. Observe that in the algorithm, we only add vertices to these sets in Line 10. Moreover, by Observation 7.28, each vertex is removed from $\mathcal{S}$ at most once. Thus, the algorithm runs this line for any vertex $v \in \mathcal{A}$ and any vertex u in its $\mathcal{H}_v$ only once. Therefore, by Claim 7.22, the number of times the algorithm adds a vertex to these sets adds up to $O(|\mathcal{A}| \cdot \log \Delta \cdot \min\{\frac{\log n}{\pi(a)}, \Delta\})$ with probability at least $1 - n^{-c}$. Note that $|\mathcal{S}|$ is equal to the number of iterations in the algorithm and $\sum_{v \in \mathcal{S}} |\mathsf{P}v|$ is the overall time that the subroutine IsAffected(v) takes over all iterations. Moreover, for any vertex $v \in \mathcal{A}$ we run Lines 5-10 of the algorithm which by Claim 7.23 and Claim 7.22 take $O(\log \Delta \cdot \min\{\frac{\log n}{\pi(a)}, \Delta\})$ time. To sum up, the total running time of the algorithm until the set $\mathcal{S}$ becomes empty is $O(|A| \cdot \log \Delta \cdot \min\{\frac{\log n}{\pi(a)}, \Delta\})$ with probability at least $1 - n^{-c}$. $\square$

We are now ready to prove Lemma 7.4.

Lemma 7.4 (restated)**.** *There is an algorithm to update* $\mathsf{GMIS}(G, \pi)$ *and the data structures required for it after insertion or deletion of any edge $e = (a, b)$ in*

$$O\left(|\mathcal{A}| \min\{\Delta, \frac{\log n}{\min\{\pi(a), \pi(b)\}}\} \log \Delta\right)$$

time w.h.p.

Proof. By Claim 7.29, with probability at least $1 - n^{-c}$ it takes $O(|\mathcal{A}| \cdot \log \Delta \cdot \min\{\frac{\log n}{\pi(a)}, \Delta\})$ time until set $\mathcal{S}$ becomes empty. We further show that when this happens we have $m(v) = m_t(v)$ and $k(v) = k_t(v)$. This is a direct result of Claim 7.27 and Claim 7.26. The former stated that Invariant 7.7 holds throughout the algorithm and the latter states that if Invariant 7.7 holds in the last iteration of the algorithm, then for any vertex u we have $m(v) = m_t(v)$ and $k(v) = k_t(v)$. Moreover, using Claim 7.24 we know that subroutine UpdateAdjacencyLists() correctly updates the adjacency lists and with probability at least $1 - n^{-c}$, it takes $O(|\mathcal{A}| \cdot \log \Delta \cdot \min\{\Delta, \frac{\log n}{\pi(a)}\})$ time given that for any vertex v we have $k(v) = k_t(v)$. This completes the proof and we obtain that with probability at least $1 - n^{-c}$, Algorithm 11 correctly updates all the data structures in $O(|\mathcal{A}| \cdot \log \Delta \cdot \min\{\frac{\log n}{\pi(a)}, \Delta\})$ time. $\square$

Chapter 8

Fully Dynamic Maximal Matching

In this chapter, we show that a variant of our MIS algorithm of Chapter 7 can also maintain a random greedy maximal matching (RGMM) over a random order on the edges, with essentially the same update-time. The end-result is the following:

Theorem 8.1. *There is a data structure to maintain a random greedy maximal matching against an oblivious adversary in a fully-dynamic graph that per update, takes* $O(\log^2 \Delta \cdot \log^2 n)$ *expected time. Furthermore, per update,* $O(1)$ *in expected edges leave or are added to the matching.*

This also leads to the following worst-case guarantee when used as a black-box [53, Theorem 1.1].

Corollary 8.2. *There is a data structure to maintain a maximal matching against an oblivious adversary in a fully-dynamic graph that w.h.p. has* $O(\log^2 \Delta \cdot \log^4 n)$ *worst-case update-time.*

We note that there has been a huge body of work on the matching problem in dynamic graphs, see e.g. [135, 22, 132, 102, 58, 55, 147, 57, 56, 100, 65, 10, 53] and the references therein. If one allows amortization, then one can get much more efficient algorithms for MM due to the seminal works of Baswana, Gupta, and Sen [22] and Solomon [147]. However, our approach of maintaining RGMM significantly deviates from the prior works on MM in dynamic graphs. We believe this is an important feature on its own and may find further applications.

Comparison to the Dynamic MIS Algorithm of Chapter 7

It is well-known that a MM of a graph can be found by first taking its line-graph and then constructing an MIS on it. Doing so, the edges in the original graph that correspond to the

MIS nodes in the line-graph will form an MM. However, the line-graph may be much larger than the original graph and thus expensive to construct and maintain. Nonetheless, because of the very specific structure of line-graphs, we can indeed implement (a simpler variant of) the same algorithm for MM without going through an explicit construction of the line-graph. In what follows, we highlight the main differences between our MIS algorithm and its MM implementation.

The first difference is that for RGMM, the random ranking π has to be drawn on the edges instead of the vertices and thus we cannot fix π in the pre-processing step. However, this is easy to handle: We draw the rank $\pi(e) \in [0, 1]$ of any edge e randomly upon its arrival.

The second difference is where the specific structure of line-graphs helps significantly. The set of edges whose MM-statuses change as a result of an edge update form a single path or a single cycle. In fact, this holds true for any arbitrary ranking π over the edges. This is in sharp contrast with MIS, where the propagations may branch (consider a star and assume that the center leaves the MIS). This branching is precisely what complicates the proof of Theorem 7.10 for MIS. Since we do not have this problem for MM, we can directly bound the set of edges with different MM-statuses by $O(\log n)$, w.h.p., using a reduction to the parallel round complexity of RGMM [60, 89]. Therefore, the analog of Theorem 7.10 for MM is significantly easier to prove. It also simplifies the algorithm we use to detect the changes to MM (compared to MIS).

The third difference is simple, but plays a crucial role in both adapting the MIS algorithm to MM and also simplifying it. Instead of storing the adjacency lists on the edges, which is the natural idea if one constructs the line-graph explicitly, we can simply store them on the vertices. In fact, because of this difference, it also turns out that for MM, we do not need to partition the adjacency lists into N^+ and N^-. That is, we can afford to keep an adjacency list $N(v)$ on each vertex v including all incident edges to v, where each edge $e \in N(v)$ is indexed by its eliminator's rank. The main reason that this is feasible, here, is that if the eliminator of an edge $e = (u, v)$ changes, we only need to re-index e in $N(u)$ and $N(v)$. However, for MIS, if the eliminator of a vertex u changes, we may have to re-index u in the adjacency lists of all of its neighbors.

Algorithm Setup. Suppose that we have fixed the ranking π on the edges. As described above, we can draw $\pi(e) \in [0, 1]$ for any edge e in the graph at the time of its arrival. In what follows, considering update number t, which can be an edge insertion or deletion, we describe how to address it and update $\mathsf{GMM}(G_{t-1}, \pi)$ to $\mathsf{GMM}(G_t, \pi)$ in $\operatorname{polylog} n$ time.

Analogous to the MIS algorithm, we define $\mathcal{A} := \{w \mid \text{elim}_{G_t,\pi}(w) \neq \text{elim}_{G_{t-1},\pi}(w)\}$ to be the set of edges whose eliminator changes after the update and call these the *affected* edges. Moreover, we define $\mathcal{F}$ to be the set of edges whose MM-status changes after the update; we call these the *flipped* edges. Note that $\mathcal{F} \subseteq \mathcal{A}$. We first provide the following algorithm.

Lemma 8.3. *There is an algorithm to update* $\mathsf{GMM}(G, \pi)$ *and the data structures needed for insertion/deletion of any edge* $f = (a, b)$ *in* $O\left(|\mathcal{F}| \min\{\Delta, \frac{\log n}{\pi(f)}\} \log \Delta\right)$ *time, w.h.p.*

Note a subtle difference between Lemma 8.3 and the similar Lemma 7.4 we had for MIS: Here, the running time is parametrized by $|\mathcal{F}|$ whereas in Lemma 7.4 it is by $|\mathcal{A}|$.

We will later prove in Section 8.2.4 that the running time in Lemma 8.3 is actually bounded by $O(\log^2 \Delta \log^2 n)$ in expectation, thus, proving Theorem 8.1.

8.1 Some Notation and Basic Tools

We will follow the generic definitions and notation of Chapters 2 and 3. The following is the additional notation we will use throughout this chapter.

Similar to Chapter 7, we define eliminators for RGMM. For each edge e, we define the *eliminator* of e, denoted by $\text{elim}_{G,\pi}(e)$, as the (unique) edge incident to e that belongs to $\mathsf{GMM}(G, \pi)$ and has the lowest rank. If e is in the MM itself, we have $\text{elim}_{G,\pi}(e) = e$; otherwise, $\text{elim}_{G,\pi}(e) \neq e$ and $\pi(\text{elim}_{G,\pi}(e)) < \pi(e)$. When no confusion is possible, we may write $\text{elim}(e)$ instead of $\text{elim}_{G,\pi}(e)$ for brevity.

Also similar to Chapter 7, instead of a random permutation we draw $\Theta(\log n)$ bit random ranks and use the following sparsification property.

Lemma 8.4. *Consider a graph* $G = (V, E)$*, let* $\pi : E \to [0, 1]$ *be a random ranking, and for any real* $p \in [0, 1]$*, define* E_p *to be the subset of* E *including any edge* e *with* $\pi(\text{elim}_{G,\pi}(e)) > p$*. W.h.p., for all* $O(\log n)$ *bit values of* $p \in [0, 1]$*, every vertex has* $O(p^{-1} \cdot \log n)$ *incident edges in* E_p*.*

Proof sketch. The proof is similar to that of Lemma 3.7 except that we also union bound over all $\text{poly}(n)$ choices of p. □

8.2 The Formal Algorithm and its Analysis

8.2.1 Data Structures

We maintain the following data structures on each edge e in graph G.

- $m(e)$: A binary variable that is 1 if edge $e \in \mathsf{GMM}(G,\pi)$ and 0 otherwise.
- $k(e)$: The rank of e's eliminator, i.e., $k(e) = \pi(\text{elim}_{G,\pi}(e))$. Note that $m(e) = 1$ iff $k(e) = \pi(e)$.

Furthermore, for any vertex v, we maintain the following data structures.

- $k(v)$: If an edge $e \in \mathsf{GMM}(G,\pi)$ is connected to v, then $k(v) = \pi(e)$; otherwise, $k(v) = \infty$.
- $N(v)$: The set of edges connected to vertex v. The set $N(v)$ is stored as a self-balancing binary search tree and each edge e in it is indexed by $k(e)$.

Similar to MIS, in the pre-processing step, we can simply construct the RGMM of the original graph $G_0 = (V, E_0)$ and fill in the data structures above in $O((|V|+|E_0|)\log n)$ time.

8.2.2 The Algorithm

The following observation is analogous to Observation 7.6 for MIS and motivates the same iterative approach in determining the changes in MM.

Observation 8.5. *For any edge $e \in \mathcal{A}$, the following properties hold:*

1. $k_{t-1}(e) \geq \pi(f)$ *and* $k_t(e) \geq \pi(f)$.
2. *if* $e \neq f$, *then* e *has a neighbor* e' *such that* $\pi(e') < \pi(e)$ *and* $e' \in \mathcal{F}$.

Algorithm 12 formalizes how our data structures can be updated after each edge insertion/deletion. The subroutines not formalized in the algorithm will be formalized subsequently.

We use *iteration* to refer to iterations of the while loop in Algorithm 12. The following invariants will hold throughout the algorithm.

Invariant 8.6. *Consider the start of any iteration and let e be the lowest-rank vertex in $\mathcal{S}$. It holds true that $k(e') = k_t(e')$ and $m(e') = m_t(e')$ for any edge e' with $\pi(e') < \pi(e)$, i.e., $k(e')$*

Algorithm 12: Maintaining the data structures after insertion or deletion of $f = (a, b)$.

```
1  $\mathcal{S} \leftarrow \{f\}$
2  while $\mathcal{S}$ is not empty do
3      Let $e = (u, v) \leftarrow \arg\min_{e' \in \mathcal{S}} \pi(e')$ be the minimum rank edge in $\mathcal{S}$.
4      UpdateDataStructures($e$)        // Updates $k(e)$, $m(e)$, $k(v)$, $k(u)$, $\mathcal{A}$, and $\mathcal{F}$.
5      if $e \in \mathcal{F}$ then
6          $\mathcal{H}_e \leftarrow \{e' \in N(v) \cup N(u) \mid k_{t-1}(e') \geq \pi(f)\}$
           // It can be found in time $O(\log \Delta \cdot |\mathcal{H}_e|)$ since $N(v)$, and $N(u)$ are indexed by $k(.)$.
7          for any edge $e' \in \mathcal{H}_e$ with $\pi(e') > \pi(e)$ do
8              insert $e'$ to $\mathcal{S}$.
9      Remove $e$ from $\mathcal{S}$.
10 UpdateAdjacencyLists()        // Updates adjacency lists where necessary.
```

and $m(e')$ already hold the correct values. Moreover, $k(e') = k_{t-1}(e')$ and $m(e') = m_{t-1}(e')$ for every other edge e' with $\pi(e') \geq \pi(e)$.

Invariant 8.7. *Consider any vertex v in an arbitrary iteration of the algorithm, and let $M_v = \{e \in E \mid m(e) = 1\}$. Throughout the algorithm, it holds that if $M_v \neq \emptyset$, then $k(v) = \min_{e \in M_v} \pi(e)$, and otherwise $k(v) = \infty$.*

We continue by formalizing all subroutines used in Algorithm 12.

Subroutine UpdateDataStructures(e)**.** Let u and v denote the two end-points of edge e. This function updates $k(e)$, $m(e)$, $k(v)$, and $k(u)$ which also determines the membership of e to sets $\mathcal{A}$ and $\mathcal{F}$. Let $x = \min(k(v), k(u))$. We show that e joins the matching iff $x \geq \pi(e)$ which results in $m(e) \leftarrow 1$, $k(e) \leftarrow \pi(e)$, $k(v) \leftarrow \pi(e)$, and $k(u) \leftarrow \pi(e)$. Otherwise, we have $m(e) \leftarrow 0$ and $k(e) \leftarrow x$. Note that if e was previously in the matching and is flipped now, we need to update $k(v)$ and $k(u)$ if they are equal to $\pi(e)$. We show that if e is removed from the matching and $k(v) = \pi(e)$ then we should set $k(v) \leftarrow \infty$ and the same for vertex u.

Subroutine UpdateAdjacencyLists()**.** We first update $N(a)$ and $N(b)$. We remove f from both these sets if f is deleted and add it otherwise. Also, for any affected edge $e = (u, v)$ we need to update its index in sets $N(v)$ and $N(u)$. We do so by a single iteration over set $\mathcal{A}$. Due to the fact that adjacency lists are BSTs with size $O(\Delta)$, this takes $O(|\mathcal{A}| \log \Delta)$ time.

8.2.3 Correctness & (Parametrized) Running Time

The correctness of Algorithm 12 follows from basic arguments and the greedy structure of RGMM and hence we defer it to Section 8.2.5. Here, we discuss why the running time of the algorithm is $O\big(|\mathcal{F}|\min\{\Delta, \frac{\log n}{\pi(f)}\}\log\Delta\big)$ as claimed in Lemma 8.3. The complete proof of both the correctness and running time of the algorithm is presented in Section 8.2.5.

Using a similar argument used for MIS, we can use Lemma 8.4 to prove (see Section 8.2.5):

Claim 8.8. *At any iteration i, with probability $1-n^{-(c+1)}$, set $\mathcal{H}_e$ has size $O(\min\{\Delta, \frac{\log n}{\pi(f)}\})$ and can be constructed in time $O(|\mathcal{H}_e|\log\Delta)$.*

Let us first analyze the running time before the last line where we update adjacency lists. Observe that any edge e' that is added to set $\mathcal{S}$ belongs to $\mathcal{H}_e$ of an edge $e \in \mathcal{F}$. Therefore, at most $O\big(|\mathcal{F}|\min\{\Delta, \frac{\log n}{\pi(f)}\}\big)$ edges are added to $\mathcal{S}$. Note that, if an edge $e' \in \mathcal{S}$ is not in set $\mathcal{F}$, we only spend $O(1)$ time for it in subroutine UpdateDataStructures(e'). Thus, the total time spent on all edges not in $\mathcal{F}$ is indeed $O\big(|\mathcal{F}|\min\{\Delta, \frac{\log n}{\pi(f)}\}\big)$. On the other hand, for each edge $e \in \mathcal{F}$, the most expensive operation is to find set $\mathcal{H}_e$ which Claim 8.8 shows can be done in $O(|\mathcal{H}_e|\log\Delta)$ time. Therefore, the total running time before UpdateAdjacencyLists() can be bounded by $O\big(|\mathcal{F}|\min\{\Delta, \frac{\log n}{\pi(f)}\}\log\Delta\big)$.

Next, in the UpdateAdjacencyLists(), we only iterate over all edges in $\mathcal{A}$ and update their position in their end-points. This takes $O(|\mathcal{A}|\log\Delta)$ time. Note that by Observation 8.5, any edge $e' \in \mathcal{A}$ is adjacent to an edge $e \in \mathcal{F}$ and $k_{t-1}(e') \geq \pi(f)$. This means that $e' \in \mathcal{H}_e$ and by Claim 8.8:

Observation 8.9. *W.h.p., $|\mathcal{A}| \leq O\big(|\mathcal{F}|\min\{\Delta, \frac{\log n}{\pi(f)}\}\big)$.*

Therefore, the overall running time is indeed $O\big(|\mathcal{F}|\min\{\Delta, \frac{\log n}{\pi(f)}\}\log\Delta\big)$ as claimed in Lemma 8.3.

8.2.4 Putting Everything Together: Proof of Theorem 8.1

Before proving Theorem 8.1 we need the following high probability bound of $O(\log n)$ on $|\mathcal{F}|$ which we prove in Section 8.2.5.

Claim 8.10. *Let G and G' be two graphs that differ in only one edge and let π be a random ranking on their edges. Then, w.h.p., there are at most $O(\log n)$ edges that have different MM-statuses in $\mathsf{GMM}(G,\pi)$ and $\mathsf{GMM}(G',\pi)$.*

Now, we are ready to prove Theorem 8.1.

Theorem 8.1 (restated). *There is a data structure to maintain a random greedy maximal matching against an oblivious adversary in a fully-dynamic graph that per update, takes*

$O(\log^2 \Delta \cdot \log^2 n)$ *expected time. Furthermore, per update,* $O(1)$ *in expected edges leave or are added to the matching.*

Proof. We use Algorithm 12. Combination of Lemma 7.4, and the fact that $|\mathcal{F}| \leq O(\log n)$ w.h.p. due to Claim 8.10, bounds the update-time of this algorithm, w.h.p., by

$$O(\log \Delta \log n) \min \left\{ \Delta, \frac{\log n}{\pi(f)} \right\} = O(\log \Delta \log^2 n) \min \left\{ \Delta, \frac{1}{\pi(f)} \right\}.$$

Since $\pi(f)$ is chosen from $[0,1]$ uniformly at random, $\mathbf{E}\left[\min\left\{\Delta, \frac{1}{\pi(f)}\right\}\right] = O(\log \Delta)$. Thus, the total running time is $O(\log^2 \Delta \log^2 n)$ in expectation, as required by the theorem.

For the adjustment-complexity, similar to MIS, it is shown in [64, Theorem 1] that RGMIS requires $O(1)$ expected adjustments under *vertex* updates. On the line-graph, this implies that if an edge is added or removed, the number of changes to RGMM is $O(1)$ in expectation; concluding the proof. □

8.2.5 Deferred Proofs

Observation 8.5 (restated)**.** *For any edge $e \in \mathcal{A}$, the following properties hold:*

1. $k_{t-1}(e) \geq \pi(f)$ *and* $k_t(e) \geq \pi(f)$.
2. *if $e \neq f$, then e has a neighbor e' such that $\pi(e') < \pi(e)$ and $e' \in \mathcal{F}$.*

Proof of part 1. Let U denote the set of edges e in E with $\pi(e) < \pi(f)$. Consider the subgraph only containing these edges. Since the matching is constructed greedily on the lower rank edges first, the set of matching edges in U does not change after the update. Let M_U denote the matching edges in U. Note that any edge e with $k_{t-1}(e) < \pi(f)$ is incident to an edge e' in M_U. Since e and e' are still incident after the update and that updating f does not change $k(e')$ we have $k_t(e) = k_{t-1}(e)$. This means that for each edge e with $k_{t-1}(e) < \pi(f)$, we have $k_t(e) = k_{t-1}(e)$ and thus e cannot be in $\mathcal{A}$ by definition. By a similar argument, for any edge e with $k_t(e) < \pi(f)$ we also have $k_{t-1}(e) = k_t(e)$ and thus $e \notin \mathcal{A}$. □

Proof of part 2. The fact that $e \in \mathcal{A}$ means that eliminator of edge e changes after the update. Let e' be its eliminator before the update. By definition of the eliminator, for $e \neq f$ we have $e \in \mathcal{A}$ iff the matching status of at least an edge incident to e with rank at most $\pi(e')$ changes. This means that if e is not incident to any edge in $\mathcal{F}$, then $e \notin \mathcal{A}$. □

Claim 8.11. *Let $e = (u, v)$ be the lowest-rank edge in $\mathcal{S}$ at the start of an arbitrary iteration. Subroutine* UpdateDataStructures(e) *correctly updates $k(e)$ and $m(e)$ in constant time assuming that Invariants 8.6 and 8.7 hold by this iteration.*

Proof. By definition, we know that eliminator of edge e is its lowest-rank edge in $N(v) \cup N(u)$ that is in the matching after the update. By Invariants 8.7 $\min(k(u), k(v))$ is the rank of an edge who has the lowest-rank amongst the edges e' in $N(v) \cup N(u)$ with $m(e') = 1$. Moreover, by Invariants 8.6, we know that for any edge e' with $\pi(e') < \pi(e)$, we have $m(e') = m_t(e')$. This means that $\min(k(u), k(v)) < \pi(e)$ iff there is at least one edge adjacent to e that is in the matching after the update. In the subroutine UpdateDataStructures(e), we use this condition to determine $m(e)$. Further in the subroutine if $m(e) = 1$ we set $k(e) = \pi(e)$ and otherwise set it to $\min(k(u), k(v))$ which is correct by definition of eliminator. To sum up, subroutine UpdateDataStructures(e) correctly updates $k(e)$ and $m(e)$ for edge e the lowest-rank edge in $\mathcal{S}$. □

Observation 8.12. *Let e and e' respectively denote two edges removed from $\mathcal{S}$ in two consecutive iteration of the algorithm in Line 9. We have $\pi(e) < \pi(e')$.*

Proof. Let $\mathcal{S}_i$ and $\mathcal{S}_{i+1}$ respectively denote set $\mathcal{S}$ at the beginning of iteration i and set $\mathcal{S}$ at the beginning of iteration $i+1$ and let e_i and e_{i+1} be the lowest-rank edges in these sets. We know that e_{i+1} is either inserted to $\mathcal{S}$ in iteration i or that it is in set $\mathcal{S}_i$. Observe that any edge added to $\mathcal{S}$ in the i-th iteration has rank lower than $\pi(e_i)$ and that e_i is the lowest-rank vertex in $\mathcal{S}_i$. As a result $\pi(e_i) < \pi(e_{i+1})$. □

Claim 8.13. *Let e be the lowest-rank edge in $\mathcal{S}$ in an arbitrary iteration i of the algorithm. Assuming that Invariants 8.6, and 8.7 hold at the start of iteration i we have:*

1. *If $\mathcal{S} = \emptyset$ at the end of iteration i, then for any edge e we have $k(e) = k_t(e)$, and $m(e) = m_t(e)$ and for any vertex v we have $k(v) = k_t(v)$.*

2. *If $\mathcal{S} \neq \emptyset$ at the end of iteration i, then Invariants 8.6 and 8.7 hold at the start of iteration $i+1$ as well.*

Proof. By Claim 8.11, we know that by the end of iteration i, for any edge e' with $\pi(e') \leq \pi(e)$ we have $m(e') = m_t(e')$, and $k(e') = k_t(e')$. Let g be the lowest-rank edge in $\mathcal{A}$ whose $k(g)$ or $m(g)$ are not updated at the end of iteration i. We show that if such an edge exists it is in set $\mathcal{S}$. Note that by Observation 8.5, edge g has at least one incident edge g' where $\pi(g') < \pi(g)$

and $g' \in \mathcal{F}$. Since g is the lowest-rank edge whose $k(g)$ or $m(g)$ are not updated, we get that $k(g')$ or $m(g')$ are updated. This means that there was an iteration $j < i$ of the algorithm in which g' was the lowest-rank edge in $\mathcal{S}$ since $k_t(g') = k_{t-1}(g')$ and that in each iteration we only update $k_t()$ for the lowest-rank edge in $\mathcal{S}$. Since g' is in $\mathcal{F}$ in iteration j, the algorithm adds all the edges in $\mathcal{H}_e$ to set $\mathcal{S}$ in that iteration. By definition of $\mathcal{H}_e$, this set includes edge g. Also, note that by Observation 8.12, the rank of vertices removed from set $\mathcal{S}$ is increasing; thus g is still in set $\mathcal{S}$ in iteration i. This means that if set $\mathcal{S}$ is empty then for all edges g, we have $m(g) = m_t(g)$ and $k(g) = k_t(g)$ and if it is nonempty Invariants 8.6 holds in the next iteration.

To complete the proof it suffices to show that if $\mathcal{S}$ is empty at the end of iteration i, for any vertex v we have $k(v) = k_t(v)$ and that otherwise Invariants 8.7 still holds at iteration $i + 1$. Note that in the i-th iteration we do not change $m(e')$ if $e' \neq e$. Thus, given that Invariants 8.7 holds at the beginning of iteration i, for any vertex v that is not incident to e we have $k = k_t(v)$ at the end of the iteration as well. Now consider vertex u that is incident to e. If e is not flipped or if $k(u) < \pi(e)$ the algorithm does not change $k(u)$ which is correct by definition of $k(u)$. Therefore, we only need to consider the case that e is flipped and $k(u) \geq \pi(e)$. In this case, if $m_t(e) = 1$, the it is be the lowest-rank edge adjacent to u with $m(.) = 1$. Algorithm correctly detects this and sets $k(u) = \pi(e)$ in this scenario. Further, if $m_t(e) = 0$ (which means $m_{t-1}(e) = 1$), then there is no other edge adjacent to u with $m(.) = 1$ in which case, as well, the algorithm correctly sets $k(u) = \infty$. We achieved this from the fact that each vertex has at most one edge with $m_{t-1}(.) = 1$ and by Invariants 8.6 any edge u_1 with a higher rank than u has $k(u_1) = k_{t-1}(u_1)$. To sum up, Invariant 8.7 still holds at the end of iteration i and the proof of the claim is completed. $\square$

Claim 8.8 (restated). *At any iteration i, with probability $1 - n^{-(c+1)}$, set $\mathcal{H}_e$ has size $O(\min\{\Delta, \frac{\log n}{\pi(f)}\})$ and can be constructed in time $O(|\mathcal{H}_e| \log \Delta)$.*

Proof. **Size of $\mathcal{H}_e$.** Observe that if $\mathcal{H}_e$ is defined, then as assured by the condition in Line 5 of Algorithm 12, $e \in \mathcal{F} \subseteq \mathcal{A}$ thus by Observation 7.6, $k_{t-1}(e) \geq \pi(f)$. Furthermore, by definition, every edge $e' \in \mathcal{H}_e$ has $k_{t-1}(e') \geq \pi(f)$. This means that if we take RGMM of G_{t-1} induced on edges with rank in $[0, \pi(f))$ and remove them and their neighbors from the graph, e and all of its neighbors in $\mathcal{H}_e$ will survive. Recall that the adversary is oblivious and the graph G_{t-1} and random permutation π are chosen independently. Therefore, applying Lemma 8.4 on graph G_{t-1} with $p = \pi(f)$ bounds $|\mathcal{H}_e|$ by $O(\pi(f)^{-1} \log n)$ w.h.p. Moreover, clearly $|\mathcal{H}_e| \leq 2\Delta - 2$ since all edges in it are incident to e, concluding the bound on the size of $\mathcal{H}_e$.

Construction of $\mathcal{H}_e$. Note that we do not change the adjacency lists $N(.)$ stored on the vertices until the very last line of Algorithm 12. Therefore, for any vertex v, we have $N(v) = N_{t-1}(v)$ before this line. This means that throughout the algorithm, for any edge $e = (u, v)$ we can iterate over edges in $N(u)$ and $N(v)$ and find all edges e' with $k_{t-1}(e') \geq \pi(a)$; all these edges will belong to $\mathcal{H}_e$. Thus the total time required is $O(|\mathcal{H}_e| \log \Delta)$. Note that this is possible since $N(v)$ and $N(u)$ are BSTs indexed by $k_{t-1}(.)$ of the elements in them but comes at the cost of an extra $O(\log \Delta)$ factor as these BSTs can have size up to Δ. □

Combining all these claims, we can prove Lemma 8.3.

Lemma 8.3 (restated). *There is an algorithm to update* $\mathsf{GMM}(G, \pi)$ *and the data structures required for it after insertion or deletion of any edge* $f = (a, b)$ *in*

$$O\left(|\mathcal{F}| \min\left\{\Delta, \frac{\log n}{\pi(f)}\right\} \log \Delta\right)$$

time, w.h.p.

Proof. **Correctness:** We first show that when set S becomes empty, for any edge e we have $k(e) = k_t(e)$, and $m(e) = m_t(e)$ and for any vertex v we have $k(v) = k_t(v)$. To do so, we will use proof by induction and show that Invariants 8.6 and 8.7 hold throughout the algorithm. This proves our claim since by Observation 8.12 if both invariants hold in the last iteration of the algorithm, then when set S becomes empty the data structures $k(.)$ and $m(.)$ are updated for all edges and vertices. By Observation 8.5, for any edge e' with $\pi(e') < \pi(f)$ we have $m(e') = m_t(e')$ and $k(e') = k_t(e')$ which means that Invariant 8.6 holds in the first iteration. Further, Invariant 8.7 holds since $m(.)$ of none of the edges has changed yet. This gives us the base case of the induction. Moreover, the induction step is a direct result of Claim 7.25 which states that if both invariants hold in an arbitrary iteration they hold in the next iteration given that $\mathcal{S}$ is nonempty.

To complete the prove of correctness, we need to show that when the algorithm terminates, for any vertex v, we have $N(v) = N_t(v)$. Subroutine UPDATEADJACENCYLISTS() first modifies the adjacency lists of vertices a and b by adding e to them if e is to be added or deleting it otherwise. Note that as we showed, when the algorithm runs this subrutine, for any edge e we already have $k(e) = k_{t-1}(e)$, thus set $\mathcal{A}$ is also updated. Therefore, Algorithm 12, correctly updates the adjacency lists by iterating over edges in $\mathcal{A}$ and updating their index in the adjacency lists of their end-points.

Running Time: First, note that by Claim 8.8, with probability at least $1 - n^{-c}$, the size of $\mathcal{H}_e$ for any edge e is $O(\min\{\Delta, \frac{\log n}{\pi(f)}\})$ and constructing that takes time $O(\log \Delta \cdot \min\{\frac{\log n}{\pi(f)}, \Delta\})$. Moreover, by Observation 8.12, we know that each edge is the lowest-rank edge in set $\mathcal{S}$ in at most one iteration. Putting these facts together gives us that the number of iterations of the algorithm is $O(|\mathcal{F}| \cdot \min\{\frac{\log n}{\pi(f)}, \Delta\})$ with probability at least $1 - n^{-c}$. We also know by Claim 8.11 that subroutine UPDATEDATASTRUCTURES(e) takes $O(1)$ time. Therefore, the total running time of the algorithm until set $\mathcal{S}$ becomes empty is $O(|\mathcal{F}| \cdot \log \Delta \cdot \min\{\frac{\log n}{\pi(f)}, \Delta\})$ with probability at least $1 - n^{-c}$. Further, subroutine UPDATEADJACENCYLISTS() takes $O(|\mathcal{A}| \log \Delta)$ time which by Observation 8.9 is bounded by $O(\log \Delta \cdot \min\{\frac{\log n}{\pi(f)}, \Delta\})$ with probability at least $1 - n^{-c}$. □

Claim 8.10 (restated)**.** *Let G and G' be two graphs that differ in only one edge and let π be a random ranking on their edges. Then, w.h.p., there are at most $O(\log n)$ edges that have different MM-statuses in* $\mathsf{GMM}(G, \pi)$ *and* $\mathsf{GMM}(G', \pi)$*.*

Proof. Assume without loss of generality that G' is obtained by removing an edge e from G and let $\mathcal{F}$ be the set of edges with different MM-statuses in $\mathsf{GMM}(G, \pi)$ and $\mathsf{GMM}(G', \pi)$. We first show that: (1) Each edge $e \in \mathcal{F}$ with $e \neq f$, has a lower-rank neighboring edge in $\mathcal{F}$. (2) The edges in $\mathcal{F}$ form either a single path or a single cycle.

Proof of (1) directly follows from Observation 8.5 part 2. For (2), observe that each edge in $\mathcal{F}$ is in at least one of the two matchings $\mathsf{GMM}(G, \pi)$ and $\mathsf{GMM}(G', \pi)$. Therefore, each vertex has at most two incident edges in $\mathcal{F}$; meaning that each connected component in $\mathcal{F}$ is indeed either a cycle or a path. To see why there cannot be more than one such connected component, observe that in this case, at least one connected component does not include f. Let g be the minimum-rank edge in this component. For g, (1) cannot hold which is a contradiction.

Now, we show that $|\mathcal{F}| = O(\log n)$. To do this, we provide a reduction to the parallel round complexity of RGMM.

GMM can be parallelized, just like GMIS as described in Section 7.4.3, in the following way: In each round, all edges that hold the locally minimum rank among their neighbors join MM, then we remove them and their neighboring edges. It is known from [89] that if ranking π over the edges is chosen randomly, then it takes $O(\log n)$ rounds until we find a maximal matching, with probability $1 - n^{-c}$ for any constant $c > 1$.

We prove that the parallel round-complexity of RGMM is at least $\Omega(|\mathcal{F}|)$, implying that w.h.p. $|\mathcal{F}| = O(\log n)$ as desired. To do this, observe that by properties (1) and (2) above, there should be a monotone path $P = (e_1, \ldots, e_k)$ in $\mathcal{F}$ where $\pi(e_i) < \pi(e_{i+1})$ for any $i \in [k-1]$ and where $k = \Omega(|\mathcal{F}|)$. (Just take the longest path in $\mathcal{F} \setminus \{f\}$, by (1) it has size $\Omega(|\mathcal{F}|)$ and by (2) it is monotone.) Furthermore, since each edge in P is in $\mathcal{F}$ and the edges in $\mathcal{F}$ belong to exactly one of $\mathsf{GMM}(G, \pi)$ and $\mathsf{GMM}(G', \pi)$, the edges in P have to alternate between the two matchings. Suppose w.l.o.g. that the odd ones belong to $\mathsf{GMM}(G, \pi)$. Now, take edge w_{2i+1} for any i. We show that it takes at least i parallel rounds until this edge joins $\mathsf{GMM}(G, \pi)$. For w_{2i+1} to join the matching, its lower rank neighbor w_{2i} should be removed so that w_{2i+1} becomes the local minimum edge. This does not happen until w_{2i-1} joins the matching since w_{2i-1} and w_{2i+1} are the only incident edges to w_{2i} that are in $\mathsf{GMM}(G, \pi)$. Now, a simple induction implies that it takes at least i rounds until w_{2i+1} joins the matching, and thus the parallel round complexity is at least $\Omega(k) = \Omega(|\mathcal{F}|)$, which as described, implies $|\mathcal{F}| = O(\log n)$ w.h.p. $\square$

Chapter 9

Fully Dynamic Approximate Matching

The problem of maintaining a large matching in the dynamic setting has received significant attention over the last two decades (see [135, 22, 132, 102, 58, 55, 147, 57, 56, 65, 10, 53] and the references therein). As we saw in Chapter 8, a maximal matching can be maintained in polylogarithmic time or even a constant time if we allow amortization [147]. This immediately gives a 2-approximation of *maximum* matching. In a sharp contrast, however, we have little understanding of the update-time-complexity once we go below 2 approximation. A famous open question of the area, asked first[1] in the pioneering paper of Onak and Rubinfeld [135] from 2010 is:

> "*Can the approximation constant be made smaller than 2 for maximum matching* [*while having polylogarithmic update-time*]?" [135]

A decade later, we are still far from achieving a polylogarithmic update-time algorithm. Prior to the algorithm of this chapter, the fastest result for maintaining a matching with a better-than-2 approximation factor was presented by Bernstein and Stein [52].[2] Their algorithm handles updates in $O(m^{1/4})$ time where m denotes the number of edges in the graph. However, a notable follow-up result of Bhattacharya, Henzinger, and Nanongkai [55] hinted that we may be able to achieve a faster algorithm. They showed that in n-vertex bipartite graphs, for any constant $\varepsilon > 0$, there is a deterministic algorithm with amortized update-time $O(n^\varepsilon)$ that maintains a $2 - \Omega(1)$ approximation of the size of the maximum matching (the algorithm maintains a fractional matching, but not an integral one).

In light of the result of Bhattacharya *et al.* [55], two main questions remained open: First, *is it possible to maintain the matching in addition to its size?* Second, *can the result be extended from bipartite graphs to general graphs?* We resolve both questions in the affirmative:

[1]See also [54, Section 4], [52, Section 7] or [65, Section 1].

[2]The algorithm of Bernstein and Stein remarkably achieves an (almost) 1.5 approximation.

Theorem 9.1. *For any constant $\varepsilon \in (0, 1)$, there is a randomized fully-dynamic algorithm that with high probability maintains a $2 - \Omega_\varepsilon(1)$ approximate maximum matching in worst-case update-time $O(\Delta^\varepsilon) + \operatorname{polylog} n$ under the standard oblivious adversary assumption. Here, Δ denotes the maximum degree in the graph. Also the precise approximation factor constant depends on ε.*

Compared to the algorithm of Bhattacharya *et al.* [55], our algorithm, at the expense of using randomization, maintains the matching itself, handles general graphs, and also improves the update-time from $O(n^\varepsilon)$ *amortized* to $\widetilde{O}(\Delta^\varepsilon)$ *worst-case*. In addition, our algorithm is arguably simpler.

Similar to other randomized algorithms of the literature, we require the standard oblivious adversary assumption. The adversary here is all powerful and knows the algorithm, but his/her updates should be independent of the random bits used by the algorithm. Equivalently, one can assume that the sequence of updates is fixed adversarially *before* the algorithm starts to operate.

9.1 Our Techniques

In this section, we provide an informal overview of the ideas used in our algorithm for Theorem 9.1 and the challenges that arise along the way.

A main intuition behind the efficient (randomized) 2-approximate algorithms of the literature is essentially "hiding" the matching from the adversary through the use of randomization. For instance if we pick the edges in the matching randomly from the dense regions of the graph where we have a lot of choices, it would then take the adversary (who recall is unaware of our random bits) a lot of trials to remove a matching edge. A natural algorithm having such behavior is *random greedy maximal matching* (RGMM) which processes the edges in a random order and greedily adds them to the matching if possible. Indeed we showed in Chapter 8 that it takes $\operatorname{poly}(\log n)$ time per edge update to maintain a RGMM.

Unfortunately, exactly the feature of RGMM (or of previous algorithms based on the same intuition) that it matches the dense regions first prevents it from obtaining a better-than-2 approximation. A simple bad example is a perfect matching whose one side induces a clique (formally, a graph on vertices $v_1, \ldots, v_{2n}$, whose edge set consists of a clique induced on $v_1, \ldots, v_n$ and edges $v_i v_{i+n}$ for $1 \leq i \leq n$). The RGMM algorithm, for instance, would pick almost all of its edges from the clique, and thus matches roughly half of the vertices while the graph has a perfect matching.

To break this 2 approximation barrier, our starting point is a variant of a streaming algorithm of Konrad, Magniez, and Mathieu [116]. The algorithm starts by constructing a RGMM M_0 of the input graph G. Unless M_0 is significantly larger than half of the size of the maximum matching $\textsc{opt}$ of G, then nearly all edges of M_0 can be shown to belong to length-3 augmenting paths in $M_0 \oplus \textsc{opt}$. Therefore to break 2 approximation, it suffices to pick a constant fraction of the edges in M_0, and discover a collection of vertex disjoint length-3 paths augmenting them. Konrad et al. [116] showed that this can be done by finding another RGMM, this time on a subgraph G' of G whose edges have one endpoint that is matched in M_0 and one endpoint that is unmatched (though for these edges to augment M_0 well, it is crucial that not all such edges are included in G').

The algorithm outlined above shows how to obtain a $2 - \Omega(1)$ approximate maximum matching by merely running two instances of RGMM. Given that we know how to maintain a RGMM in polylogarithmic update time from Chapter 8, one may wonder whether we immediately get a similar update-time for this algorithm. Unfortunately, there is a rather serious drawback. Note that the second stage graph G' is *adaptively* determined based on matching M_0. Particularly, a single edge update that changes matching M_0 may lead to deletion/insertion of a *vertex* (along with its edges) in the second stage graph G'. While we can handle edge updates in polylogarithmic time, the update-time for vertex updates is still polynomial (in the degree of the vertex being updated). Therefore, the algorithm, as stated, requires an update-time of up to $\widetilde{O}(\Delta)$.

To get around the issue above, our first insight is a parametrized analysis of the update-time depending on the structure of the edges in M_0. Suppose that matching M_0 is constructed by drawing a random *rank* $\pi_0(e) \in [0, 1]$ independently on each edge e and then iterating over the edges in the increasing order of their ranks. We show that the whole update-time (i.e. that of both the first and the second stage matchings) can be bounded by $\widetilde{O}(\rho)$, where

$$\alpha = \max_{e \in M_0} \pi_0(e), \qquad \beta = \min_{e \in M_0} \pi_0(e), \qquad \text{and} \qquad \rho = \frac{\alpha}{\beta}.$$

The reason is as follows. For an edge update e, the probability that it causes an update to matching M_0, is upper bounded by α. (If rank of e is larger than the highest rank ever in M_0, then $e \notin M_0$ and thus its insertion/deletion causes no update to M_0.) On the other hand, in case of an update to M_0, the cost of a vertex update to the second stage graph can be bounded by its degree, which we show can be bounded by $\widetilde{O}(1/\beta)$ using the sparsification property of RGMM (Section 3.4) applied to the first stage matching M_0. Thus, the overall update time is indeed $\widetilde{O}(\alpha \cdot \beta^{-1}) = \widetilde{O}(\rho)$.

The analysis highlighted above, shows that as the rank of edges in the first stage matching M_0 get closer to each other, our update-time gets improved. In general, this ratio can be as large as $O(\Delta)$. A natural idea, however, is to partition M_0 into subsets $S_1, \ldots, S_{1/\varepsilon}$ such that the edges in each subset, more or less, have the same ranks (i.e. a max over min rank ratio of roughly Δ^{ε}). We can then individually construct a second-stage graph G_i for each S_i, find a RGMM M_i of it and use it to augment S_i (and thus M_0). Since there are only $1/\varepsilon$ groups, there will be one that includes at least $\varepsilon = \Omega(1)$ fraction of edges of M_0. Therefore, augmenting a constant fraction of edges in this set alone would be enough to break 2 approximation.

However, another technical complication arises here. Once we choose to augment only a subset S_i of the edges in M_0, with say $\frac{\max_{e \in S_i} \pi_0(e)}{\min_{e \in S_i} \pi_0(e)} \leq \Delta^{\varepsilon}$, we cannot bound the update-time by $\widetilde{O}(\Delta^{\varepsilon})$ anymore. (The argument described before only works if we consider all the edges in M_0.) The reason is that, normally, in the second stage graph G_i we would like to have edges that have one endpoint in S_i and one endpoint that is unmatched in M_0 so that if both endpoints of an edge $e \in S_i$ are matched in the matching of G_i, then we get a length-3 augmenting path of M_0. This makes this second stage graph G_i very sensitive to the precise set of vertices matched/unmatched in the whole matching M_0 (as opposed to only those matched in S_i) and this would prevent us from using the same argument to bound the update-time by $\widetilde{O}(\Delta^{\varepsilon})$.

To resolve this issue, on a high level, we also consider any vertex that is matched in M_0, but its matching edge has rank higher than those in S_i, as "unmatched" while constructing graph G_i (see Algorithm 13). This will allow us to argue that the update-time is $\widetilde{O}(\Delta^{\varepsilon})$. The downside is that not all found length 3 paths will be actual augmenting paths of M_0. Fortunately, though, we are still able to argue that the algorithm finds sufficiently many *actual* augmenting paths for M_0 and thus achieves a $2 - \Omega(1)$ approximation (see Section 9.3).

9.2 A Static Algorithm

In this section we describe a static algorithm for finding an approximate maximum matching. We show in Section 9.3 that the algorithm provides a $2 - \Omega(1)$ approximation and show in Section 9.4 that it can be maintained in update-time $\widetilde{O}(\Delta^{\varepsilon})$.

Intuitive explanation of the algorithm. We start with a RGMM M_0. After that, we partition the edge set of M_0 into $1/\varepsilon$ partitions $S_1, \ldots, S_{1/\varepsilon}$ such that roughly the maximum rank over the minimum rank in each partition is at most Δ^{ε}. Then, focusing on each partition S_i, we try augmenting the edges of S_i by finding two random greedy matchings of subgraphs $G_i'^A$ and $G_i'^B$ that are determined based on set S_i. Roughly, each edge in $G_i'^A$ (and similarly in $G_i'^B$) has one endpoint that is matched in S_i—this is the edge to be augmented—and one endpoint that is either unmatched or matched after the edges in S_i are processed in the greedy construction of M_0. Therefore if for an edge $uv \in S_i$, its endpoint v is matched via an edge vx in the matching of $G_i'^A$ and u is matched via an edge uy in the matching of $G_i'^B$, and in addition x and y are unmatched in M_0 then uy, uv, vx will be a length 3 augmenting path of M_0.

Algorithm 13: An algorithm for $(2-\Omega(1))$-approximate maximum matching.

```
1  Input: Graph $G = (V, E)$ and a parameter $\varepsilon \in (0, 1)$.
2  $M_0 \leftarrow \mathsf{GMM}(G, \pi_0)$ where $\pi_0$ is a random permutation of $E$.
3  for any $i \in \{1, 2, \ldots, 1/\varepsilon\}$ do
4      if $i < 1/\varepsilon$ then
5          $S_i \leftarrow \big\{e \mid e \in M_0$ and $\pi_0(e) \in \big(\Delta^{-i\varepsilon}, \Delta^{-(i-1)\varepsilon}\big]\big\}$.    // We call $S_i$ "partition $i$".
6      else
7          $S_{1/\varepsilon} \leftarrow \big\{e \mid e \in M_0$ and $\pi_0(e) \in \big[0, \Delta^{-1+\varepsilon}\big]\big\}$.
8      $U_i \leftarrow \{u \mid$ for any edge $uv$ either $uv \notin M_0$ or $uv \in \cup_{j=1}^{i-1} S_j\}$.
       // $U_i$ includes nodes that are unmatched in $M_0$ or matched by edges in $S_1, \ldots, S_{i-1}$.
9      For each edge $uv \in S_i$ put its endpoint with lower ID in set $V_i^A$ and the other in $V_i^B$.
       // The use of IDs is just to simplify the statements. Any arbitrary way of putting one endpoint of
          the edge in $V_i^A$ and the other in $V_i^B$ would work.
10     Partition $U_i$ into $U_i^A$ and $U_i^B$ by each node picking its partition independently and
        u.a.r.
11     Sample each edge in $S_i$ independently with probability $p := 0.03$.
12     $V_i'^A \leftarrow$ vertices in $V_i^A$ whose edge in $S_i$ is sampled.
13     $V_i'^B \leftarrow$ vertices in $V_i^B$ whose edge in $S_i$ is sampled.
14     $G_i'^A \leftarrow G[V_i'^A \times U_i^A]$.
15     $G_i'^B \leftarrow G[V_i'^B \times U_i^B]$.
16     $M_i \leftarrow \mathsf{GMM}(G_i'^A, \pi_i) \cup \mathsf{GMM}(G_i'^B, \pi_i)$ where $\pi_i$ is a fresh random permutation of $E$.
17 Return the maximum matching of graph $(M_0 \cup M_1 \cup \ldots \cup M_{1/\varepsilon})$.
```

9.3 Approximation Factor of Algorithm 13

In this section, we prove that the approximation factor of Algorithm 13 is at most $2 - \Omega(1)$ given that ε is a constant.

We fix one arbitrary maximum matching of graph G and denote it by $\textsc{opt}$; recall that $|\textsc{opt}| = \mu(G)$. Having this matching $\textsc{opt}$, we now call an edge $e \in M_0$ *3-augmentable* if it is in a length 3 augmenting path in $\textsc{opt} \oplus M_0$. Observe that since $\textsc{opt}$ cannot be augmented, this augmenting path should start and end with edges in $\textsc{opt}$ and thus edge e has to be in the middle.

The following lemma is crucial in the analysis of the approximation factor. Basically, it says that for any partition S_i where most of edges in S_i are 3-augmentable (which in fact should be the case for most of the partitions if $|M_0|$ is close to $0.5\mu(G)$), roughly $p/4$ fraction of the edges in S_i are in length 3 augmenting paths in $S_i \oplus M_i$. We emphasize that this does not directly prove the bound on the approx factor as these length 3 augmenting paths in $S_i \oplus M_i$ may not necessarily be augmenting paths in $M_0 \oplus M_i$.

Lemma 9.2. *For any $i \in [1/\varepsilon]$ and any parameter $\delta \in (0, 1)$, if $(1 - \delta)$ fraction of the edges in S_i are 3-augmentable, then in expectation, there are at least $(\frac{(1-\delta)p}{4} - 4p^2)|S_i|$ edges in S_i where both of their endpoints are matched in M_i.*

In order to prove this lemma, in Lemma 9.3 we recall a property of the greedy maximal matching algorithm under vertex samplings originally due to [116, 115]. We note that the property that we need is slightly stronger than the one proved in [115, Theorem 3] but follows from a similar argument. Roughly, we need a lower bound on the number of vertices in a specific vertex subset that are matched, while the previous statement only lower bounded the overall matching size. We provide the complete proof in Appendix 9.5.

Lemma 9.3. *Let $G(V, U, E)$ be a bipartite graph, π be an arbitrary permutation over E, and M be an arbitrary matching of G. Fix any parameter $p \in (0, 1)$ and let W be a subsample of V including each vertex independently with probability p. Define X to be the number of edges in M whose endpoint in V is matched in $\mathsf{GMM}(G[W \cup U], \pi)$; then*

$$\mathop{\mathbf{E}}_{W}[X] \geq p(|M| - 2p|V|).$$

Equipped with Lemma 9.3, we are ready to prove Lemma 9.2.

Proof of Lemma 9.2. Fix a partition S_i which includes $(1 - \delta)|S_i|$ 3-augmentable edges and denote by T_i the subset of edges in S_i that are 3-augmentable; implying that

$$|T_i| \geq (1 - \delta)|S_i|. \tag{9.1}$$

Recall that each edge $e \in T_i$ is the middle edge in a length 3 augmenting path in $\textsc{opt} \oplus M_0$ where $\textsc{opt}$ is a fixed maximum matching of G. Define set $\textsc{opt}^A$ (resp. $\textsc{opt}^B$) to be the subset of edges uv in $\textsc{opt}$ where one of their endpoints, say, v is in $V(T_i) \cap V_i^A$ (resp. $V(T_i) \cap V_i^B$) and the other endpoint u is in set U_i^A (resp. U_i^B).

We say an edge $ab \in T_i$ is *good* if a is matched in $\textsc{opt}^A$ and also b is matched in $\textsc{opt}^B$; and use Y to denote the subset of edges in T_i that are good. We first claim that

$$\mathop{\mathbf{E}}_{U_i^A, U_i^B}[|Y|] \geq \frac{1}{4}|T_i|, \tag{9.2}$$

where observe that here the expectation is only taken over the randomization in partitioning U_i into U_i^A and U_i^B. To see this, fix an edge $ab \in T_i$ and let au and bv be the edges in $\textsc{opt}$ that along with ab form a length 3 augmenting path. Observe that edge $au \in \textsc{opt}^A$ if $u \in U_i^A$ and $bv \in \textsc{opt}^B$ if $v \in U_i^B$. Since the partition of v and u is chosen independently and u.a.r., there is a probability $\frac{1}{4}$ that both these events occur, implying $ab \in Y$. Linearity of expectation over every edge in T_i proves (9.2).

Now, consider graphs $G_i^A := G[V_i^A \times U_i^A]$ and $G_i^B := G[V_i^B \times U_i^B]$ and observe that $\textsc{opt}^A$ is a matching of G_i^A and $\textsc{opt}^B$ is a matching of G_i^B. One can confirm that $V_i'^A$ is a random subsample of V_i^A where for each $v \in V_i^A$, $\mathbf{Pr}[v \in V_i'^A] = p$. More importantly, whether for a vertex v the event $v \in V_i'^A$ holds is independent of which other vertices are in $V_i'^A$. (Though we note that $v \in V_i'^A$ is not independent of those vertices in $V_i'^B$.) Similarly, $V_i'^B$ can be regarded as a random subsample of V_i^B wherein the vertices appear independently from each other. As a result, graph $G_i'^A$ (resp. $G_i'^B$) is essentially obtained by retaining a random subsample of the vertices in the V_i^A (resp. V_i^B) partition of graph G_i^A (resp. G_i^B). We can thus use Lemma 9.3 while fixing matching $\textsc{opt}^A$ to get

$$\mathbf{E}\left[\#\text{ of vertices in } V(\textsc{opt}^A) \cap V_i'^A \text{ matched in } \mathsf{GMM}(G_i'^A, \pi_i)\right] \geq p(|\textsc{opt}^A| - 2p|V_i^A|). \tag{9.3}$$

Similarly,

$$\mathbf{E}\left[\#\text{ of vertices in } V(\textsc{opt}^B) \cap V_i'^B \text{ matched in } \mathsf{GMM}(G_i'^B, \pi_i)\right] \geq p(|\textsc{opt}^B| - 2p|V_i^B|). \tag{9.4}$$

Observe that $\mathbf{E}[|V(\textsc{opt}^A) \cap V_i'^A|] = p|\textsc{opt}^A|$ since each edge in $\textsc{opt}^A$ has one endpoint in V_i^A which is sampled to $V_i'^A$ with probability p. Combined with (9.3) this means that

$$\begin{aligned} &\mathbf{E}\left[\#\text{ of vertices in } V(\textsc{opt}^A) \cap V_i'^A \ \underline{\text{not}} \text{ matched in } \mathsf{GMM}(G_i'^A, \pi_i)\right] \\ &\leq p|\textsc{opt}^A| - p(|\textsc{opt}^A| - 2p|V_i^A|) \leq 2p^2|V_i^A| = 2p^2|S_i|. \end{aligned} \tag{9.5}$$

Similarly by (9.4),

$$\begin{aligned} &\mathbf{E}\left[\ \#\text{ of vertices in } V(\textsc{opt}^B) \cap V_i'^B \ \underline{\text{not}} \text{ matched in } \mathsf{GMM}(G_i'^B, \pi_i)\ \right] \\ &\leq p|\textsc{opt}^B| - p(|\textsc{opt}^B| - 2p|V_i^B|) \leq 2p^2|V_i^B| = 2p^2|S_i|. \end{aligned} \tag{9.6}$$

By (9.2) we have $\frac{1}{4}|T_i|$ expected good edges. Out of these, each edge $ab \in Y$ is sampled, i.e., $a \in V_i'^A$ and $b \in V_i'^B$ with probability p. Therefore, in expectation, there are a total of $\frac{p}{4}|T_i|$ sampled good edges. Say a sampled good edge ab is wasted if a is unmatched in $\mathsf{GMM}(G_i'^A, \pi_i)$ or b is unmatched in $\mathsf{GMM}(G_i'^B, \pi_i)$. Combined with (9.5) and (9.6) there are at most $2p^2|S_i| + 2p^2|S_i| \leq 4p^2|S_i|$ wasted edges. This means that the expected number of sampled good edges that are not wasted is at least

$$\frac{p}{4}|T_i| - 4p^2|S_i|.$$

Moreover, by (9.1), $|T_i| \geq (1-\delta)|S_i|$. Replacing this into the equation above, we get that there are, in expectation, at least $\frac{p}{4}(1-\delta)|S_i| - 4p^2|S_i| = (\frac{(1-\delta)p}{4} - 4p^2)|S_i|$ good edges that are not wasted, i.e., both of their endpoints are matched in $M_i = \mathsf{GMM}(G_i'^A, \pi_i) \cup \mathsf{GMM}(G_i'^B, \pi_i)$ as claimed in the lemma. □

The following claim shows that there is a subset $S_{i^\star}$ that is "large enough" compared to the size of matching M_0 and is much larger than the total number of edges in previous subsets $S_1, \ldots, S_{i^\star-1}$.

Claim 9.4. *There exists an integer $i^\star \in [1/\varepsilon]$ such that*

$$|S_{i^\star}| \geq \frac{1}{2^{13/\varepsilon}}|M_0| \qquad \textit{and} \qquad |S_{i^\star}| > 2^{11}\sum_{i=1}^{i^\star-1}|S_i|.$$

Proof. Let $i^\star$ be the smallest integer in $[1/\varepsilon]$ for which

$$|S_{i^\star}| \geq 2^{12i^\star - \frac{13}{\varepsilon}}|M_0|, \tag{9.7}$$

we show that both conditions should hold for $i^\star$. First, we have to prove that there is a choice of $i^\star \in [1/\varepsilon]$ satisfying (9.7). Suppose for the sake of contradiction that this is not the case; then:

$$\sum_{i=1}^{1/\varepsilon}|S_i| < \sum_{i=1}^{1/\varepsilon} 2^{12i-\frac{13}{\varepsilon}}|M_0| = 2^{\frac{-13}{\varepsilon}}|M_0|\sum_{i=1}^{1/\varepsilon}2^{12i} \ll 2^{\frac{-13}{\varepsilon}}|M_0|(2 \times 2^{12/\varepsilon}) < |M_0|.$$

Observe that subsets $S_1, \ldots, S_{1/\varepsilon}$ partition the edges in M_0 and thus it should hold that $\sum_{i=1}^{1/\varepsilon}|S_i| = |M_0|$; implying that the equation above is indeed a contradiction, proving existence of $i^\star$.

The first inequality of the claim is automatically satisfied for $i^\star$ due to (9.7) since

$$|S_{i^\star}| \geq 2^{12i^\star - \frac{13}{\varepsilon}}|M_0| > 2^{-\frac{13}{\varepsilon}}|M_0|.$$

It thus only remains to prove the second inequality. For that, observe that since $i^\star$ is the smallest integer satisfying (9.7), then for any $i < i^\star$ we have $|S_i| < 2^{12i-\frac{13}{\varepsilon}}|M_0|$. This means that

$$\sum_{i=1}^{i^\star-1}|S_i| < \sum_{i=1}^{i^\star-1} 2^{12i-\frac{13}{\varepsilon}}|M_0| = 2^{\frac{-13}{\varepsilon}}|M_0|\sum_{i=1}^{i^\star-1}2^{12i} \ll 2^{\frac{-13}{\varepsilon}}|M_0|(2\times 2^{12(i^\star-1)}) = 2^{12i^\star-11-\frac{13}{\varepsilon}}|M_0|.$$

Combining this with (9.7) we get

$$\frac{|S_{i^\star}|}{\sum_{i=1}^{i^\star-1}|S_i|} > \frac{2^{12i^\star-\frac{13}{\varepsilon}}|M_0|}{2^{12i^\star-11-\frac{13}{\varepsilon}}|M_0|} = 2^{11},$$

implying the second inequality of the claim as well. □

Let us recall a folklore property that if maximal matching M_0 is not already large enough, then most of the edges in it are 3-augmentable.

Observation 9.5 (folklore)**.** *If $|M_0| < (\frac{1}{2}+\delta)\mu(G)$, then at least $(\frac{1}{2}-3\delta)\mu(G)$ edges in M_0 are 3-augmentable.*

Proof. See e.g. [115, Lemma 1] for a simple argument. □

We are now ready to analyze the approximation factor. We prove that for $\delta = \frac{1}{1000\times 2^{13/\varepsilon}}$, the matching returned by Algorithm 13 has, in expectation, size at least $(\frac{1}{2}+\delta)\mu(G)$. We first assume that $|M_0| < (\frac{1}{2}+\delta)\mu(G)$ as otherwise matching M_0 already achieves the desired approximation factor. By Observation 9.5, this means that at least $(\frac{1}{2}-3\delta)\mu(G)$ edges of M_0 are 3-augmentable; meaning that the number of edges in M_0 that are not 3-augmentable is at most

$$|M_0| - \left(\frac{1}{2}-3\delta\right)\mu(G) \le \left(\frac{1}{2}+\delta\right)\mu(G) - \left(\frac{1}{2}-3\delta\right)\mu(G) = 4\delta\mu(G) \le 8\delta|M_0|. \tag{9.8}$$

Let $i^\star \in [1/\varepsilon]$ be the integer satisfying Claim 9.4. By (9.8) there are at most $8\delta|M_0|$ edges in M_0 and thus in $S_{i^\star}$ that are not 3-augmentable. Therefore,

$$\begin{aligned}
\text{\# of 3-augmentable edges in } S_{i^\star} &\ge |S_{i^\star}| - 8\delta|M_0| \\
&= |S_{i^\star}| - 8\frac{1}{1000\times 2^{13/\varepsilon}}|M_0| \\
&> |S_{i^\star}| - \frac{1}{100}\times\frac{|M_0|}{2^{13/\varepsilon}} \\
&\ge |S_{i^\star}| - \frac{1}{100}\times\frac{2^{13/\varepsilon}|S_{i^\star}|}{2^{13/\varepsilon}} \qquad \text{First inequality of Claim 9.4.} \\
&\ge 0.99|S_{i^\star}|.
\end{aligned}$$

Since 0.99 fraction of the edges in $S_{i^\star}$ are 3-augmentable, by Lemma 9.2, there are at least

$$\left(\frac{0.99p}{4} - 4p^2\right)|S_{i^\star}| \overset{p=0.03}{=} 0.003825|S_{i^\star}| > 0.003|S_{i^\star}|$$

edges in $S_{i^\star}$ whose both endpoints are matched in $M_{i^\star}$. We would like to argue that these form length 3 augmenting paths but note that an edge $e \in M_{i^\star}$ may have an endpoint that is already matched in subsets $S_1, \ldots, S_{i^\star-1}$. However, the crucial observation here is that since by the second inequality of Claim 9.4, we have $|S_{i^\star}| \geq 2^{11} \sum_{i=1}^{i^\star-1} |S_i|$ and each edge in $S_1 \cup \ldots \cup S_{i^\star-1}$ can be connected to two edges in $M_{i^\star}$ the number of these length 3 augmenting paths that are also augmenting paths in $\textsc{opt} \oplus M_0$ is at least

$$0.003|S_{i^\star}| - 2 \times \sum_{i=1}^{i^\star-1} |S_i| \geq 0.003|S_{i^\star}| - 2 \times \frac{|S_{i^\star}|}{2^{11}} > 0.002|S_{i^\star}|.$$

Each of these augmenting paths can be used to increase size of M_0 by one, therefore the final matching has size at least

$$|M_0| + 0.002|S_{i^\star}| \geq |M_0| + \frac{0.002}{2^{13/\varepsilon}}|M_0| = \left(1 + \frac{1}{500 \times 2^{13/\varepsilon}}\right)|M_0| \geq \left(1 + \frac{1}{500 \times 2^{13/\varepsilon}}\right)\frac{1}{2}\mu(G)$$
$$= \left(\frac{1}{2} + \frac{1}{1000 \times 2^{13/\varepsilon}}\right)\mu(G),$$

which proves the approximation factor is $2 - \Omega(1)$ so long as $\varepsilon > 0$ is a constant.

9.4 Dynamic Implementation of Algorithm 13

In this section, we describe how we can maintain Algorithm 13 in update time $O(\Delta^\varepsilon + \operatorname{polylog} n)$.

9.4.1 Tools

We borrow two black-box tools from the previous works. The first one is a simple corollary of the algorithm of Gupta and Peng [102], see also [51] for a proof of this corollary.

Lemma 9.6 ([102]). *Let Δ' be a fixed upper bound on the maximum degree of a graph at all times. Then we can maintain a $(1+\varepsilon)$ approximate matching deterministically under edge insertions and deletions in worst-case update time $O(\Delta'/\varepsilon^2)$ per update.*

We use Lemma 9.6 only for the last step of Algorithm 13 in which we need to maintain a maximum matching of $M_0 \cup M_1 \cup \ldots \cup M_{1/\varepsilon}$ which is a graph with maximum degree $O(1/\varepsilon)$. We also use, as black-box, the following result that appeared first in [40] and we proved it in Chapter 8.

Lemma 9.7. *Let π be a random ranking where $\pi(e) \in [0,1]$ for each edge e is drawn uniformly at random upon its arrival. Then maximal matching $\mathsf{GMM}(G,\pi)$ can be maintained under edge insertions and deletions in expected time $O(\log^2 \Delta \times \log^2 n)$ per update (without amortization). Furthermore, for each update, the adjustment-complexity is in expectation $O(1)$ and w.h.p. $O(\log n)$.*

9.4.2 Data Structures & Setup

Algorithm 13 computes two types of matchings: (1) Matching $M_0 = \mathsf{GMM}(G, \pi_0)$ which is a standard random greedy maximal matching of the whole graph G. (2) Matchings $M_1, \ldots, M_{1/\varepsilon}$ which are computed on specific subgraphs of G. Observe in Algorithm 13 that each matching M_i for $i \in \{1, \ldots, 1/\varepsilon\}$ is the union of two random greedy matchings $\mathsf{GMM}(G_i'^A, \pi_i)$ and $\mathsf{GMM}(G_i'^B, \pi_i)$. A crucial observation here is that these two graphs $G_i'^A$ and $G_i'^B$ by definition are vertex disjoint. Therefore defining graph G_i to be the union of these two graphs, M_i would be equivalent to $\mathsf{GMM}(G_i, \pi_i)$.

Now we have $\frac{1}{\varepsilon} + 1$ graphs $G, G_1, \ldots, G_{1/\varepsilon}$ and $\frac{1}{\varepsilon} + 1$ independently drawn rankings $\pi_0, \pi_1, \ldots, \pi_{1/\varepsilon}$. Therefore, if a priori these graphs were fixed and remained unchanged after each edge insertion/deletion, we could use Lemma 9.7 to update each one of them in expected time $\operatorname{polylog} n$ requiring only a total update-time of $O(\frac{1}{\varepsilon}) \cdot \operatorname{polylog} n$. However, as highlighted in Section 9.1 the challenge is that the vertex sets of graphs $G_1, \ldots, G_{1/\varepsilon}$ are *adaptively* determined based on matching M_0. That is, a single edge update that changes matching M_0 may lead to many *vertex* insertions/deletions to graphs $G_1, \ldots, G_{1/\varepsilon}$ that are generally much harder to handle than edge updates. Therefore, we need to be careful about what to maintain and how to do it to ensure these vertex updates can be determined and handled efficiently.

Fixing the randomizations. To maintain the matching of Algorithm 13, we fix all the randomizations required. There are two types of randomizations involved: (1) Randomizations on the edges, such as the random rankings and edge samplings; as in Lines 2,16, and 11. (2) Randomizations on the vertices; as in Line 10. We reveal the randomizations on the vertex set in the preprocessing step as it is static. But we reveal the randomizations on the edges upon their arrival. For completeness, we mention the precise random bits drawn below.

For each edge e, we draw the following upon its arrival:

$\text{is\_sampled}_i(e) \in \{0,1\}$: This is drawn for any $i \in [1/\varepsilon]$ independently. It is 1 with probability p and 0 otherwise. It determines the outcome of edge-sampling in Line 11 of the algorithm.

$\pi_i(e) \in [0,1]$: The rank of e in ranking π_i. This is drawn for any $i \in \{0, \ldots, 1/\varepsilon\}$.

And for each vertex v, in the pre-processing step, we draw:

$\text{partition}_i(v) \in \{A, B\}$: This is drawn for any $i \in [1/\varepsilon]$ independently. It is A with probability 0.5 and B otherwise. The value determines whether v would join U_i^A or U_i^B if it is partitioned in Line 10 of the algorithm.

Data structures. Let us for simplicity define $G_0 = G$. For any vertex v and any $i \in \{0, \ldots, \frac{1}{\varepsilon}\}$ we maintain the following data structures:

$k_i(v)$: If v is not part of graph G_i or if it is unmatched in $\mathsf{GMM}(G_i, \pi_i)$ then $k_i(v) = 1$. Otherwise, if e is the edge incident to v that is in matching $\mathsf{GMM}(G_i, \pi_i)$ then $k_i(v) = \pi_i(e)$.

$N_i(v)$: The set of neighbors of v in graph G_i. This set is stored as a self-balancing binary search tree in which each neighbor u of v is indexed by $\pi_i(\text{elim}_{G_i,\pi_i}(uv))$. (If v is not in the vertex-set of G_i then simply $N_i(v) = \emptyset$.)

9.4.3 The Update Algorithm

We run $1 + 1/\varepsilon$ instances of Lemma 9.7 for maintaining greedy matchings of $G_0, \ldots, G_{1/\varepsilon}$. Moreover, we run a single instance of Lemma 9.6 on the edges in union of matchings $M_0 \cup \ldots \cup M_{1/\varepsilon}$.

As mentioned previously, a single edge update to graph G_0 may change the structure of graphs $G_1, \ldots, G_{1/\varepsilon}$ and in particular may lead to vertex insertions or deletions in them. Therefore, our main focus in this section is to show how we can detect these vertices that join/leave graphs $G_1, \ldots, G_{1/\varepsilon}$ and their incident edges in these graphs efficiently. Before that, we need the following lemma. The proof is a simple consequence of Lemma 8.4 and thus we defer it to Section 9.6.

Lemma 9.8. *Suppose that an edge e is inserted to or deleted from a graph G_i for some*

$i \in \{0, \ldots, 1/\varepsilon\}$. *After updating matching* $\mathsf{GMM}(G_i, \pi_i)$ *(e.g. by Lemma 9.7) and getting the list L of edges that joined or left the matching, we can update $k_i(\cdot)$ and $N_i(\cdot)$ accordingly in expected time* $\operatorname{polylog} n$.

Consider insertion or deletion of an edge f. We use the following procedure to maintain our data structures and finally the matching returned by Algorithm 13.

Step 1: Updating M_0. We first update matching M_0. This is done by Lemma 9.7 in $\operatorname{polylog} n$ expected time. After that, we also update data structures $k_0(v)$ and $N_0(v)$ where necessary using Lemma 9.8. There are two cases. If matching M_0 changes after the update, then we may have to update the vertex sets of graphs $G_1, \ldots, G_{1/\varepsilon}$. This is the operation that is costly and we handle it in the next steps. If M_0 does not change, the only remaining update is to see if f itself is part of a graph G_i and reflect that. This only takes polylogarithmic time using Lemmas 9.7 and 9.8.

Step 2: Updating vertex-sets of $G_1, \ldots, G_{1/\varepsilon}$. The vertex-set of each graph G_i is composed of four disjoint subsets $V_i'^A, U_i^A, V_i'^B$, and U_i^B. One can confirm from Algorithm 13 that whether a vertex v belongs to one of these sets (and which one if so) can be uniquely determined by knowing the edge incident to v that is in matching M_0 or knowing that no such edge exists. Therefore:

Observation 9.9. *If after the update, a vertex v leaves or is added to the vertex-set of a graph G_i, then there must exist an edge connected to v that either joined or left matching M_0.*

By Observation 9.9, to update the vertex-sets, it suffices to only iterate over vertices whose matching edge in M_0 has changed and determine which graph G_i they should belong to. The procedure is a simple consequence of the way Algorithm 13 constructs these graphs and also the randomizations fixed previously. We provide the details in Algorithm 14 for completeness.

It has to be noted that we are only updating the vertex-sets in this step. In particular, for a vertex v that e.g. joins graph G_i, we do not construct its adjacency list $N_i(v)$ yet. This is postponed to the next step after all the vertex sets are completely updated.

Algorithm 14: Updating vertex-sets of $G_1, \dots, G_{1/\varepsilon}$.

```
1  for any vertex $v$ whose match-status in $M_0$ has changed after the update do
2      if $v$ is now unmatched then
3          $\ell_v \leftarrow 0$.
4      else
5          Let $e$ be the edge incident to $v$ that is now in matching $M_0$.
6          Let $S_j$ be the partition to which $e$ will be assigned in Algorithm 13 based on
           $\pi_0(e)$.
7          $\ell_v \leftarrow j$.
8      for any $i \in [1/\varepsilon]$ do
9          if $\ell_v < i$ then
10             If $\mathsf{partition}_i(v) = A$, then $v \in U_i^A$. Otherwise $\mathsf{partition}_i(v) = B$, thus $v \in U_i^B$.
11         if $\ell_v = i$ then
               // At this state, $v$ should be matched in $M_0$ through its incident edge $e$.
12             if $\mathsf{is\_sampled}_i(e) = 0$ then
13                 Vertex $v$ is not in the vertex-set of graph $G_i$.
14             else
15                 If $v$ is the lower-ID endpoint of $e$, then $v \in V_i'^A$. Otherwise, $v \in V_i'^B$.
16         if $\ell_v > i$ then
17             Vertex $v$ is not in the vertex-set of graph $G_i$.
```

Step 3: Updating adjacency lists of $G_1, \dots, G_{1/\varepsilon}$ and their matchings. The previous step updated the vertex-sets. Here, we update the adjacency lists and the matchings $M_1, \dots, M_{1/\varepsilon}$. Precisely, we update data structure $N_i(v)$ for each vertex v and each $i \in [1/\varepsilon]$ where necessary. Note that for any vertex v, both $k_0(v)$ and adjacency list $N_0(v)$ were already updated in Step 1.

First, for any vertex v that leaves a graph G_i, we immediately remove its incident edges from the graph one by one. Each one of these should be regarded as edge deletions and thus we can use Lemma 9.7 to update M_i. We then update k_i and N_i data structures accordingly using Lemma 9.8.

Next, for any vertex v that is added to the vertex set of a graph G_i, we have to determine the set of its neighbors in this graph. To do so, we take the steps formalized as Algorithm 15. A crucial observation to note before reading the description of Algorithm 15 is stated below. The proof is a direct consequence of the greedy structure of RGMM, thus we defer it to Appendix 9.6.

Claim 9.10. *Suppose that the edge f that is being inserted to/deleted from G is part of* *matching M_0 (if deleted before deletion and if inserted after insertion). Note that if this was not the case, then updating f would not change the vertex sets of $G_1, \ldots, G_k$. Also assume that f belongs to partition S_j of matching M_0 in Algorithm 13. Then this update may only affect vertex sets of graphs $G_1, \ldots, G_j$. In particular, any graph G_k with $k > j$ remains unchanged after insertion or deletion of f.*

Claim 9.10 is algorithmically useful in the following way. Suppose that $f \in S_j$ and let α be the minimum rank considered to be in S_j in Algorithm 13. Then we can remove all edges whose eliminator ranks are less than α from G, and the remaining graph will include all edges that we have to consider for graphs $G_1, \ldots, G_j$. This, by Lemma 8.4 prunes the degrees to $\widetilde{O}(\alpha^{-1})$ and helps reducing the running time. See Algorithm 15 and Lemma 9.11 for the details.

Algorithm 15: Updating adjacency lists of $G_1, \ldots, G_{1/\varepsilon}$.

```
1  Let $f$ be the original edge that was inserted/deleted from $G$ and suppose that it
     changed matching $M_0$ (otherwise, vertex-sets of $G_1, \ldots, G_{1/\varepsilon}$ will remain the same.)
2  Suppose that $f \in S_j$ (if $f$ was deleted, $f \in S_j$ before deletion, and if inserted, $f \in S_j$
     after it).
   // Note that $f$ has to be in $M_0$ to change it once updated. Thus it should belong to a set $S_j$.
3  If $j < 1/\varepsilon$ then let $\alpha \leftarrow \Delta^{-i\varepsilon}$, otherwise if $j = 1/\varepsilon$ let $\alpha \leftarrow 0$.
   // $\alpha$ is the lower bound on edge ranks that get partitioned to $S_j$ according to Algorithm 13.
4  for any vertex $v$ and any $i \in [1/\varepsilon]$ such that $v$ joins the vertex set of $G_i$ do
       // We can detect these vertices efficiently by only going through the changes found in Step 2
          without exhaustively checking all vertices in the graph.
5      $L_v \leftarrow \{u \in N_0(v) \mid \pi_0(\text{elim}_{G_0,\pi_0}(uv)) \geq \alpha\}$
       // Set $L_v$ has size $\min\{\Delta, O(\alpha^{-1} \log n)\}$ by Lemma 8.4 and can be constructed in time $\widetilde{O}(|L_v|)$
          since all edges in $N_0(v)$ are already indexed by their eliminator ranks.
6      for any neighbor $u \in L_v$ of $v$ do
7          if ($v \in V_i'^A$ and $u \in U_i^A$) or ($v \in V_i'^B$ and $u \in U_i^B$) or ($u \in V_i'^A$ and $v \in U_i^A$) or
             ($u \in V_i'^B$ and $v \in U_i^B$) then
8              Add $u$ to $N_i(v)$ and $v$ to $N_i(u)$.
9              Update matching $M_i$ using Lemma 9.7 according to this edge insertion.
10             Update $k_i(\cdot)$ and $N_i(\cdot)$ as necessary by this edge insertion using Lemma 9.8.
```

Step 4: Updating the final matching. Finally, recall that we run multiple instances of Lemma 9.6 to maintain a $(1+\varepsilon)$ approximate maximum matching of graph $M_0 \cup \ldots \cup M_{1/\varepsilon}$ which will include our final matching. Throughout the updates above, we keep track of all edges that leave/join these matchings and for each one of them we update this final matching via Lemma 9.6.

9.4.4 Correctness & Running Time of Update Algorithm

In this section, as the title describes, we prove the correctness of the update algorithm above and analyze its running time. Namely, we prove the following lemma.

Lemma 9.11. *The update algorithm of previous section correctly updates all data structures and the matching and its expected running time per update without amortization is* $O(\Delta^{\varepsilon} \operatorname{polylog} n)$.

Before that, let us show how we can actually turn this update-time to $O(\Delta^{\varepsilon} + \operatorname{polylog} n)$ as claimed by Theorem 9.1. To do so, given ε, we consider a smaller value for ε, say $\varepsilon/2$. Then the update-time would be $O(\Delta^{\varepsilon/2} \operatorname{polylog} n)$. Now if $\Delta^{\varepsilon/2} \gg \operatorname{polylog} n$, then we already have $\Delta^{\varepsilon/2} \operatorname{polylog} n \ll \Delta^{\varepsilon}$. Otherwise, Δ is polylogarithmic and the whole update-time is also polylogarithmic.

As another note, in Theorem 9.1 we state that the update-time is worst-case but Lemma 9.11 bounds the expected update-time. To turn this into a worst-case bound, we use the reduction of Bernstein *et al.* [53]. For the reduction to work, the crucial property is that the update-time bound should hold in expectation but without any amortization, as is the case here.

Proof of Lemma 9.11. It is easy to verify correctness of Steps 1, 2, and 4 which are actually quite fast and take only $\operatorname{polylog} n$ time in total. We do provide the necessary details for these steps at the end of this proof. However, the main component of the update-algorithm is Step 3 which takes $O(\Delta^{\varepsilon} \operatorname{polylog} n)$ time. We thus first focus on this step and analyze its running time and correctness.

As before, assume that edge f is updated. If matching M_0 does not change as a result of this update, then the vertex sets of all graphs $G_1, \ldots, G_{1/\varepsilon}$ will remain unchanged. However, if updating f changes M_0, then f should be in M_0 once in the graph. As in Algorithm 15, we assume $f \in S_j$ and let α be the minimum possible rank in S_j. By Claim 9.10, graphs $G_{j+1}, \ldots, G_{1/\varepsilon}$ remain unchanged. Also, one can confirm from Algorithm 13 that all edges in graphs $G_1, \ldots, G_j$ have eliminator rank of at least α in M_0. Thus, for any vertex v added to a graph G_i, the set L_v indeed includes all edges incident to v that may belong to G_i. Moreover, by Lemma 8.4 this set L_v has size at most $\min\{\Delta, O(\alpha^{-1} \log n)\}$ and that can be found in time $\widetilde{O}(|L_v|)$ since the neighbors of v in $N_0(v)$ are indexed by their eliminator-rank.

The overall update-time required for Step 3 is thus

$$(\# \text{ of edges updated in } M_0) \times \min\left\{\Delta, O\left(\frac{\log n}{\alpha}\right)\right\}.$$

By Lemma 9.7, the number of edges that are updated in M_0 is w.h.p. bounded by $O(\log n)$. It remains to determine the expected value of the second factor in the running time above. Let us use $I_1, \ldots, I_{1/\varepsilon}$ to denote the interval of ranks considered by Algorithm 13. That is, $I_{1/\varepsilon} = [0, \Delta^{-1+\varepsilon}]$ and for any $i < 1/\varepsilon$, $I_i = (\Delta^{-i\varepsilon}, \Delta^{-(i-1)\varepsilon}]$. For edge f that is to be updated, probability that $\pi_0(f)$ is in the ith interval is upper bounded by $\Delta^{-(i-1)\varepsilon}$. Moreover, given that $\pi_0(f)$ is in the ith interval, then $\min\{\Delta, \alpha^{-1}\log n\}$ would be at most $\Delta^{i\varepsilon}\log n$. Thus:

$$\begin{aligned}\mathbf{E}\left[\min\left\{\Delta, O\left(\frac{\log n}{\alpha}\right)\right\}\right] &\leq \sum_{i=1}^{1/\varepsilon} \mathbf{Pr}[\pi_0(f) \in I_i] \times \mathbf{E}\left[\min\left\{\Delta, O\left(\frac{\log n}{\alpha}\right)\right\} \,\middle|\, \pi_0(f) \in I_i\right]\\ &\leq \sum_{i=1}^{1/\varepsilon} \Delta^{-(i-1)\varepsilon} \times \Delta^{i\varepsilon}\log n \leq \Delta^{\varepsilon}\log n.\end{aligned}$$

This means that the overall update-time required for Step 3 is $O(\Delta^{\varepsilon} \operatorname{polylog} n)$.

Now, we focus on the other steps.

In Step 1, only matching M_0 as well as the data structures related to it are updated. These are correct and only take $\operatorname{polylog} n$ expected time by Lemmas 9.7 and 9.8.

In Step 2, we detect the updates to the vertex sets of $G_1, \ldots, G_{1/\varepsilon}$. This is done in Algorithm 14 by iterating over all vertices whose match-status in M_0 is changed and checking the conditions of Algorithm 13. By Observation 9.9, indeed any vertex who gets added/deleted from any graph G_i should have an incident edge in M_0 whose match-status in M_0 has changed. Therefore, we do discover all the updates. Moreover, for each vertex encountered we only spend $O(1)$ time in Algorithm 14 to detect which graph G_i it belongs to and there are w.h.p. only $O(\log n)$ such vertices who have an edge with an updated match-status in M_0 by Lemma 9.7. Thus, the total running time for Step 2 is $O(\log n)$.

Finally, in Step 4, we update the final matching. The graph here is composed of $O(1/\varepsilon)$ matchings and thus has maximum degree $O(1/\varepsilon)$. Therefore, each update by Lemma 9.6 takes $O(1/\varepsilon) = O(1)$ time. Moreover, an edge update to graph M_0 w.h.p. affects at most $O(\log n)$ edges by Lemma 9.7, each of these updated edges may lead to vertex insertions/deletions in each graph G_i. But these propagated vertex insertions/deletions also affect at most $O(\log n)$ edges in each of the graphs. Thus, the total number of edges that leave/join graph $M_1 \cup \ldots \cup M_{1/\varepsilon}$ is at most $O(\log^2 n)$, which is also the upper bound on the running time of Step 4. □

9.5 Greedy Matching Size under Vertex Sampling

In this section, we prove Lemma 9.3 by extending the ideas presented in [115].

Proof. Consider the following equivalent process of constructing $\mathsf{GMM}(G[W \cup U], \pi)$ that gradually reveals the subsample W of V: We initialize matching $M' \leftarrow \emptyset$, initially mark each vertex as *alive*, and then iterate over the edges in E in the order of π. Upon visiting an edge vu with $v \in V$ and $u \in U$, if either v or u is *dead* (i.e., not alive) we discard vu. Otherwise, we call vu a *potential-match* and then reveal whether v belongs to W by drawing an p-Bernoulli random variable. If $v \notin W$, no edge connected to v can be added to M', thus we mark v as dead and discard vu. If $v \in W$, we add vu to M' and then mark both v and u as dead as they cannot be matched anymore. One can confirm that at the end of this process, M' is precisely equivalent to matching $\mathsf{GMM}(G[W \cup U], \pi)$.

Let us fix an infinite tape of independent p-Bernoulli random variables $\vec{x} = (x_1, x_2, x_3, \ldots)$ and use it in the following way: Once we encounter the i'th potential-match edge vu whose vertex v is matched in matching M (this is the matching in the statement of lemma, not to be confused with the greedy matching M' we are constructing), we use the value of x_i as the indicator of the event $v \in W$. Observe that if $x_i = 1$, then this edge uv will be added to M'. Moreover, define $X := \sum_{i=1}^{D} x_i$ to be the number of 1's in the tape that we encounter by the end of process. (Here D is the upper bound on the number of times that we reveal a random variable from the tape.) One can confirm that X is precisely the number of vertices in the V-side of M that are matched in M': The precise quantity that lemma requires the lower bound for.

Since each variable x_i is 1 independently with probability p, we expect X to be pD. However, note that the value of D itself is a random variable depending on the randomizations revealed. Nonetheless, since D is a stopping time for the process, we can use Wald's equation [130] to argue that the expected value of X is indeed at least $p\,\mathbf{E}[D]$. Therefore, it suffices to show that $\mathbf{E}[D] \geq (|M| - 2p|V|)$ to prove $\mathbf{E}[X] \geq p(|M| - 2p|V|)$ as required by the lemma.

To see why $\mathbf{E}[D]$ is this large, observe that for any vertex $v \in V$ that is matched in M say via edge vu, we will encounter a potential-match edge unless u is matched to another vertex. However, since each vertex in V is sampled into W with probability p, there are in expectation at most $p|V|$ vertices in W. Each such vertex can destroy at most two edges in M. For the rest of $|M| - 2p|V|$ edges, we encounter at least a potential-match edge for which we reveal a random variable of the tape. Thus $\mathbf{E}[D] \geq |M| - p|V|$ and $\mathbf{E}[X] \geq p(|M| - 2p|V|)$

as desired. $\square$

9.6 Missing Proofs

Proof of Lemma 9.8. Updating k_i is easy. If for a vertex v, $k_i(v)$ has to be updated, then an edge incident to it must be in L. On the other hand, there are at most two edges connected to each vertex in L: At most one edge incident to it can join the matching and at most one was in it to leave. Therefore, by simply iterating over the edges in L, we can update $k_i(v)$ of any vertex necessary. This takes $O(|L|)$ time, which by Lemma 9.7 is w.h.p. bounded by $O(\log n)$.

Updating N_i is more tricky. If an edge joins the matching, it can now become the eliminator of many other edges and if the eliminator of an edge uv changes, we have to re-index u and v in each other's adjacency list $N_i(v)$. The crucial observation is that since the matching is constructed greedily, the matching on edges with rank in $[0, \pi_i(e))$ remains unchanged after inserting/deleting e. As a result, if the eliminator of an edge had rank in $[0, \pi_i(e))$, it remains to be its eliminator. Therefore, all the changes occur in the subgraph including edges with eliminator rank before the update was larger than $\pi_i(e)$. By Lemma 8.4 this graph has maximum degree $\min\{\Delta, O(\frac{\log n}{\pi_i(e)})\}$ w.h.p. Moreover, we can iterate over all neighbors g of any edge $f \in L$ with $\text{elim}_{G_i,\pi_i}(g) > \pi_i(e)$ in time $\min\{\Delta, \frac{1}{\pi_i(e)}\}\operatorname{polylog} n$. Re-indexing each also takes at most time $O(\log \Delta)$. Thus, the overall time required is, w.h.p.,

$$|L| \times \mathbf{E}\left[\min\left\{\Delta, \frac{1}{\pi_i(e)}\right\}\operatorname{polylog} n\right] \le \mathop{\mathbf{E}}_{\pi_i(e)\sim[0,1]}\left[\min\left\{\Delta, \frac{1}{\pi_i(e)}\right\}\right]\operatorname{polylog} n \qquad \text{(W.h.p. } |L| = O(\log n).\text{)}$$

$$= O(\log \Delta) \times \operatorname{polylog} n = \operatorname{polylog} n,$$

completing the proof. $\square$

Proof of Claim 9.10. Fix a graph G_k with $k > j$. We first prove no edge is removed from G_k after updating f. Take an edge uv that is in G_k. By Algorithm 13, one endpoint of this edge, say v w.l.o.g., should be matched in M_0 via an edge that belongs to S_k. The other endpoint u, is either unmatched in M_0 or matched via an edge ux where $ux \in S_\ell$ for some $\ell < k$. After the update, v remains to be matched to the same vertex u in M_0 for the following reason. Since $f \in S_j$, $uv \in S_k$, and $k > j$, then $\pi_0(f) > \pi_0(uv)$. As a result, since matching M_0 is constructed greedily by processing the edges in the increasing order of ranks, all the edges processed before f that are in the matching will remain in the matching no matter if f is in the graph or not, meaning that uv will remain in M_0. Moreover, even though edge ux may leave matching M_0, the edge to which u will be matched after the update (if any) will have rank at least $\pi_0(f)$. Thus u will remain in set U_k and as a result, the edge vu will remain in G_k.

A similar argument shows that any edge uv in G_k after the update, should have been in G_k before the update too. More precisely, if v is the part of edge uv that is matched in S_k after the update, then its matching edge in M_0 should have been in M_0 before the update too for precisely the same reason mentioned above. Moreover, no matter the update, if the other endpoint u is in set U_k after the update, it should have been in U_k before the update too. Implying that uv should have also been in G_k before the update.

Combination of the arguments of the two paragraphs above implies that graph G_k remains exactly the same after and before the update. $\square$

Part IV

Streaming Algorithms

Chapter 10

Random-Order Streaming Matching

In this chapter, we study the maximum matching problem in the *semi-streaming* model of computation [88] defined as follows.

Definition 10.1. *Given a graph $G = (V, E)$ with n vertices $V = \{1, \ldots, n\}$ and m edges in E presented in a stream $S = \langle e_1, \ldots, e_m \rangle$, a semi-streaming algorithm makes a single pass over the stream of edges S and uses $O(n \cdot \text{polylog}\,(n))$ space, measured in words of size $\Theta(\log n)$ bits, and at the end outputs an approximate maximum matching of G.*

The greedy algorithm for maximal matching gives a simple $1/2$-approximation algorithm to this problem in $O(n)$ space. When the stream of edges is adversarially ordered, this is simply the best result known for this problem, while it is also known that a better than $\frac{1}{1+\ln 2} \sim 0.59$-approximation is not possible [106] (see also [105, 96]). Closing the gap between these upper and lower bounds is among the most longstanding open problems in the graph streaming literature.

Going beyond this "doubly worst case" scenario, namely, an adversarially-chosen graph and an adversarially-ordered stream, there has been an extensive interest in recent years in studying this problem on **random order streams**. This line of work was pioneered in [116] who showed that the $1/2$-approximation of greedy can be broken in this case and obtained an algorithm with approximation ratio $(1/2 + 0.003)$ for this problem. Since [116], there has been two main lines of attack on this problem. Firstly, [114, 90, 85] followed up on the approach of [116] and improved the approximation ratio all the way to $6/11$ [85]. In parallel, [15] built on the sparsification approach of [51, 52] in dynamic graphs to achieve an (almost) $2/3$-approximation but at the cost of $\widetilde{O}(n^{1.5})$ space, which is no longer semi-streaming. A beautiful work of [50] then obtained a semi-streaming (almost) $2/3$-approximation by showing how a generalization of the sparsification approach in [15] can be found in $\widetilde{O}(n)$ space.

The $2/3$-approximation ratio of the algorithm of [50] is the best possible among all prior techniques for this problem: the first line of attack in [116, 114, 90, 85] is based on finding length-3 augmenting paths and even finding *all* these paths does not lead to a better-than-$2/3$-approximation[1]. The second line in [15, 50] is based on finding an *edge-degree constrained subgraph* (EDCS) which hits the same exact barrier as there are graphs whose EDCS does not provide a better than $2/3$-approximation (see [51]). Finally, even for an algorithmically easier variant of this problem, the one-way communication problem, which roughly corresponds to only measuring the space of the algorithm when crossing the midpoint of the stream, the best known approximation ratio is still $2/3$ which is known to be tight for adversarial orders/partitions [96].

Given this state-of-affairs, the $2/3$-approximation ratio for random-order streaming matching has emerged as natural barrier [114, 50]. In particular, [50] posed obtaining a $(2/3 + \Omega(1))$-approximation to this problem as an important open question. We resolve this question in the affirmative in our work.

Our Contributions

Our main result is a semi-streaming algorithm for maximum matching in random-order streams with approximation ratio strictly-better-than-$2/3$.

Theorem 10.2 (Main Result). *Let G be an n-vertex graph whose edges arrive in a random-order stream. For an absolute constant $\varepsilon_0 > 0$, there is a single-pass streaming algorithm that obtains a $(\frac{2}{3} + \varepsilon_0)$-approximate maximum matching of G using $O(n \log n)$ space with high probability.*

Theorem 10.2 breaks the $2/3$-barrier of all prior work in [116, 114, 90, 15, 50, 85]. Moreover, even though the improvement over $2/3$ is minuscule in this theorem (while we did not optimize for constants, the bound on ε_0 is only $\sim 10^{-14}$ at this point), it still proves that $(2/3)$-approximation is not the "right" answer to this problem. This is in contrast to some other problems of similar flavor such as one-way communication complexity of matching (on adversarial partitions) [96, 13] or the fault-tolerant matching problem [13] which are both solved using similar techniques (see the unifying framework of [13] based on EDCS) and for both $2/3$-approximation is provably best possible.

[1]The work of [85] also considers length-5 augmenting paths. However, these paths are used *instead of* length-3 paths "missed" by the algorithm *not in addition to* them and thus the same shortcoming persists.

10.1 Overview of Techniques

Prior work: As stated earlier, there has been two main lines of attack on the streaming matching problem in random-order streams. The first approach aims to find a *large* matching of the graph G early on in the stream, and then spends the rest of the stream *augmenting* this matching. For instance, [116] showed that in order for the greedy algorithm to fail to find a better-than-$1/2$-approximation, the algorithm should necessarily pick many "wrong" edges early on in the stream. As such, in instances where greedy is not beating the $1/2$-approximation itself, we already have an almost $1/2$-approximation by the *middle* of the stream, and we can thus focus on augmenting this matching in the remainder half to beat $1/2$-approximation. The work of [114] then improved this result further by showing that a modified greedy algorithm, when unsuccessful in obtaining a large matching itself, finds an almost $1/2$-approximation when only $o(1)$-fraction of the stream has passed (as opposed to middle), which gives us more room for augmentation. Finally, [85] built on this approach and further improved the augmentation phase.

The second approach to this problem was based on obtaining an EDCS, a subgraph defined by [51, 52] and studied further in [13], that acts as a "matching sparsifier". On a high level, an EDCS is a sparse subgraph satisfying the following two constraints: (i) edge-degree of edges in the EDCS cannot be "high", while (ii) edge-degree of missing edges cannot be "low". These constraints ensure that an EDCS always contains an almost $2/3$-approximate matching of the graph and has additional robustness properties [51, 52, 15, 13, 50]. For instance, [15] proved that union of several EDCS computed on different parts of a random stream, is itself an EDCS for the entire stream. This allowed them to compute an EDCS of the input in $\widetilde{O}(n^{1.5})$ space and directly obtain their almost $2/3$-approximation. Finally, [50] gave an elegant proof that weakening the requirement of EDCS allows one to still preserve the almost $2/3$-approximation but now recover this subgraph in only $O(n \log n)$ space. More specifically, the algorithm of [50] first finds a subgraph only satisfying property (i) of the EDCS in the first $o(1)$ fraction of the stream, and then picks *all* (potentially) necessary edges for satisfying property (ii) in the remainder; the proof then shows that this set of potentially necessary edges is of size only $O(n \log n)$.

Our work: Our approach can be seen as a natural combination of these two mostly disjoint lines of work. The first part comes from a better understanding of EDCS. We present a rough characterization of when an EDCS cannot beat the $2/3$-approximation, which shows that in these instances, we can effectively ignore the second constraint of EDCS. As a result, we obtain that the only way for the algorithm of [50] to fail to achieve a better-than-$2/3$-approximation, is if it already picks an almost $2/3$-approximation in the first $o(1)$ fraction of the stream. Note that this is conceptually similar to the first line of work on random-order streaming matching, but the techniques are entirely disjoint. In particular, our proof is a deterministic property of EDCS not a randomized property of a greedy algorithm on a particular ordering.

We are now in the familiar territory of having a large matching very early on in the stream, and we can spend the remainder of the stream augmenting it. The main difference however is that starting from an almost $2/3$-approximation matching, there is essentially no length-3 paths for us to augment and we instead need to handle length-5 augmenting paths. The key challenge is to find the middle edge of these length-5 augmenting paths. Indeed, we note that the $2/3$-approximation lower bound of [96] for *adversarial* order streams gives away a $2/3$-approximate matching early on for free, yet it is provably impossible to augment it in the remainder of the stream using a semi-streaming algorithm. To get around this, we crucially use the random arrival assumption again. Particularly, we regard any length-5 augmenting path whose middle edge arrives after its two endpoint edges as a "discoverable" path and then find a constant fraction of such paths. Since the edges arrive in a random order, a constant fraction of length-5 augmenting will be discoverable and thus we are able to beat $2/3$-approximation in our setting.

10.2 Background and Definitions

In this chapter, we will follow the general notation defined in Chapter 2. Here we mention some additional notation that we use only in this chapter.

For integer $k \geq 1$ and $p \in [0, 1]$, we use $\mathcal{B}(k, p)$ to denote the *binomial distribution* with parameters k and p. More formally, $\mathcal{B}(k, p)$ is the discrete probability distribution of the number of successful experiments out of k experiments each with an independent probability p of success.

Random-order streams. We consider the random-order streaming setting where the edges of G arrive one by one in an order chosen uniformly at random from all possible orderings. Let e_i be the i-th edge that arrives in the stream. For any two parameters a, b satisfying $1 \leq a < b \leq m$ we use $G[a, b]$ to denote the subgraph of G on vertex-set V and edge-set

$\{e_a, \ldots, e_b\}$. We may also use $G_{<a}$ and $G_{\geq a}$ respectively as shorthands for $G[1, a-1]$ and $G[a, m]$.

For the input graph G defined by the stream, we can assume w.l.o.g. that $\mu(G) \geq c \log n$ for any desirably large constant c. The reason is that any graph can be easily shown to have at most $2n \cdot \mu(G)$ edges and if $\mu(G) = O(\log n)$ then we can store the whole input in the memory and report an optimal solution using $O(n \log n)$ space. We further assume throughout the chapter that the number of edges m is known by the algorithm in advance. This is a common assumption in the literature and can be removed via standard techniques by guessing m in geometrically increasing values at the expense of multiplying the space by an $O(\log n)$ factor.

Edge degrees: For any edge $e = (u, v) \in E$, we define the *edge-degree* of e in G as $\deg(u) + \deg(v)$.

Hall's witness set: We will use the following standard extension of the Hall's marriage theorem for characterizing maximum matching size in bipartite graphs.

Fact 10.3 (Extended Hall's Theorem; cf. [103]). *Let $G = (L, R, E)$ be a bipartite graph and $|L| = |R| = n$. Then,*

$$\max \Big(|A| - |N(A)| \Big) = n - \mu(G),$$

where A ranges over L or R, separately. We refer to such set A as a ***witness set****.*

Fact 10.3 follows from Tutte-Berge formula for matching size in general graphs [150, 49] or a simple extension of the proof of Hall's marriage theorem itself.[2]

10.2.1 Bernstein's Algorithm

We briefly review the parameters and guarantees of the algorithm of Bernstein [50] that we use in our result. In the following, we slightly increase the constants in the parameters which is needed for our results.

Definition 10.4 (Parameters). *For some small $\varepsilon \in (0, \frac{1}{2})$ to be determined later, let*

$$\lambda := \frac{\varepsilon}{128}, \qquad \beta_+ := 64 \cdot \lambda^{-2} \log(1/\lambda), \qquad \beta_- = (1 - \lambda) \cdot \beta_+.$$

The algorithm of [50] proceeds in the following way:

[2]Simply add $n - \mu(G)$ vertices to each side of the graph and connect them to all the original vertices; then apply original's Hall's theorem for perfect matching to this graph as this graph now has one.

Algorithm 3. The structure of Bernstein's algorithm [50].

The algorithm of [50] proceeds in two phases as follows:

- Phase I terminates within the first εm edges of the stream. At the end of Phase I, the algorithm constructs a subgraph $H \subseteq G_{<\varepsilon m}$ such that (in addition to some other properties) for all $(u, v) \in H$:

$$\deg_H(u) + \deg_H(v) \leq \beta_+.$$

  Moreover, let U be the set of *all* edges in $G_{\geq \varepsilon m}$ such that

$$\deg_H(u) + \deg_H(v) < \beta_-.$$

- In Phase II, the algorithm simply stores U in the memory and at the end of the stream returns a maximum matching of $H \cup U$.

The following lemma is all we need from [50].

Lemma 10.5 (Lemma 4.1 of [50])**.** *There is a way of constructing the subgraph H of $G_{<\varepsilon m}$ such that with probability at least $1 - n^{-3}$, $|H \cup U| = O(n \log(n) \cdot \mathrm{poly}(1/\varepsilon))$.*

10.3 Finding an Almost (2/3)-Approximation Early On

We start by characterizing the tight instances of the algorithm of [50] (Algorithm 3). Roughly speaking, we show that the only way for Algorithm 3 to end up with a (2/3)-approximation is if in its Phase I it computes a subgraph H that already has an almost (2/3)-approximate matching. This will then be used by our algorithm in the next section to obtain a strictly better-than-(2/3)-approximation by augmenting this already-large matching.

We start by presenting and proving this result for bipartite graphs which is the main part of the proof; we then extend the result to general graphs (with no considerable loss of parameters for our purpose) using the probabilistic method approach of [13] for the original EDCS.

10.3.1 Bipartite Graphs

In this section we prove the following structural result:

Theorem 10.6. *Let $\lambda \in (0, 1/2)$ and $\beta_- \leq \beta_+$ be such that $\beta_+ \geq \frac{10}{\lambda}$ and $\beta_- \geq (1-\lambda)\beta_+$. Suppose $G = (L, R, E)$ is any bipartite graph and:*

(i) H is a subgraph of G where for all $(u, v) \in H$: $\deg_H(u) + \deg_H(v) \leq \beta_+$; and

(ii) U is the set of all edges (u,v) in $G \setminus H$ such that $\deg_H(u) + \deg_H(v) < \beta_-$.

Then, for any parameter $\delta \in (0,1)$, either:

$$\mu(H) \geq (1-4\lambda) \cdot (\frac{2}{3} - \delta) \cdot \mu(G) \quad \text{or} \quad \mu(H \cup U) \geq (1-2\lambda) \cdot \left(\frac{2}{3} + \frac{\delta^2}{18}\right) \cdot \mu(G).$$

Let us define the following (see Figure 10.1 for an illustration):

- Let M^* be a maximum matching of G and define $M^*_U := M^* \cap U$ and $M^*_{\bar{U}} := M^* \setminus U$.
- A is Hall's theorem witness set in $H \cup M^*_U$ (as in Fact 10.3) and $B := N_{H \cup M^*_U}(A)$. Without loss of generality we assume $A \subseteq L$ and define $\bar{A} := L \setminus A$ and $\bar{B} := R \setminus B$.

We start with the following simple claim that follows easily from Fact 10.3.

Claim 10.7. *For the witness set A:*

(i) $|\bar{A}| + |B| \leq \mu(H \cup U)$.

(ii) There is a matching $\bar{M} \subseteq M^*_{\bar{U}}$ between A and $\bar{B}$ in G with size $|\bar{M}| = \mu(G) - \mu(H \cup M^*_U)$.

Proof. For part (i), note that $|\bar{A}| + |B| = n - (|A| - |B|) = n - (n - \mu(H \cup M^*_U)) \leq \mu(H \cup U)$ where the second to last equation is since A is a witness set in $H \cup M^*_U$, and the last equation is because M^*_U is a subset of U.

For part (ii), consider the graph consisting of only M^*. Given that for the set A in this new graph, we have $|A| - |N_{M^*}(A)| \leq n - \mu(G)$ by Fact 10.3, we get that $|N_{M^*}(A)| - |B| \geq \mu(G) - \mu(H \cup M^*_U)$. Moreover, since M^* is a matching, these new neighbors of A are only formed via a matching. Finally, as these edges are missing from $H \cup M^*_U$, this matching from A to $\bar{B}$ should entirely belong to $M^*_{\bar{U}}$. □

Consider any edge $(u,v) \in \bar{M}$ defined in Claim 10.7. As $\bar{M} \subseteq M^*_{\bar{U}}$, by property (ii) of Theorem 10.6 statement, we have, $\deg_H(u) + \deg_H(v) \geq \beta_-$. We arbitrarily remove the edges on u and v until the above inequality becomes tight for every edge (since $\bar{M}$ is a matching, this is possible indeed). We let F be the remaining edges. Note that any edge in F is incident on exactly one vertex of $\bar{M}$ as there are no edges in $H \cup M^*_U$ between the endpoints of $\bar{M}$. We record these properties as follows:

$$\forall (u,v) \in \bar{M}: \quad \deg_F(u) + \deg_F(v) = \beta_- \quad \text{and} \quad |F| = |\bar{M}| \cdot \beta_-. \tag{10.1}$$

In the following, we first give some illustrating examples that highlight the ideas for proving Theorem 10.6, and then proceed to the formal proof.

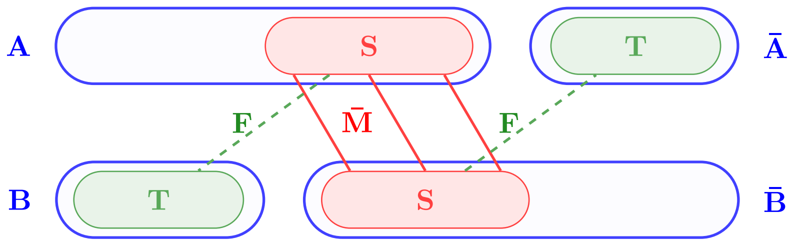


Figure 10.1: An illustration of the Hall's witness set and our notation in the proof of Theorem 10.6. Note that in particular, there are no edges between A and $\bar{B}$ in $H \cup M^*_U$, and the matching $\bar{M}$ belongs entirely to $M^*_{\bar{U}}$.

Illustrating Examples and The High Level Idea

By Claim 10.7, $\mu(H \cup U) \geq \mu(G) - |\bar{M}|$; thus, if $\bar{M}$ is sufficiently smaller than $\mu(G)/3$, we already satisfy the second condition of Theorem 10.6 and we would be done. As such, in this informal discussion, we are simply going to assume that $|\bar{M}| = \mu(G)/3$. Moreover, we define the endpoints of $\bar{M}$ as S, and their neighborset of S in H as the set T. See Figure 10.1 for an illustration. Let us now consider two *extreme* cases:

When degrees of edges in $\bar{M}$ are "highly balanced". That is, both endpoints of edges in $\bar{M}$, namely, vertices in S, have degree $\beta_-/2$ (recall that by Eq (10.1), edge-degree of every edge in $\bar{M}$ is β_-). We claim that in this case, there is a large matching in H already that satisfies condition one of Theorem 10.6.

Firstly, note that the degrees of vertices in T needs to be at most $\beta_+ - \beta_-/2 \leq (1 + \lambda)\beta_+/2$ to satisfy property (i) of Theorem 10.6 for edges of H between S and T. As such, the subgraph between S and T has degree $\beta_-/2$ on the S-side and degree at most $\beta_+/2$ on the T-side. By putting a mass of $\frac{2}{(1+\lambda)\beta_+}$ on every edge of this subgraph, we can create a feasible fractional matching of value $|S| \cdot (\beta_-/2) \cdot (2/((1+\lambda) \cdot \beta_+)) \geq (1-\Theta(\lambda))|S|$ in this subgraph (and thus H). Considering the integrality gap of the matching polytope in bipartite graphs is one, this means there is a matching of size $(1-\Theta(\lambda))|S| = (1-\Theta(\lambda)) \cdot 2|\bar{M}| = (1-\Theta(\lambda)) \cdot 2\mu(G)/3$ in H. Thus, in this case, H already has a large matching that satisfies the first condition of Theorem 10.6.

It is worth mentioning that the tight 2/3-approximation example of [51] for EDCS can be used here to prove that in this case, the subgraph $H \cup U$ may *not* have a matching of size larger than $2\mu(G)/3$, i.e., the second condition of Theorem 10.6 may indeed not hold here.

When degrees of edges in $\bar{M}$ are "mostly unbalanced". Let us for our informal discussion assume that for every edge in $\bar{M}$ its endpoint in L has degree $\beta_-/3$ while its endpoint in R has degree $2\beta_-/3$ (again recall that sum of these degrees should add up to β_- by Eq (10.1)). We claim that in this case, $H \cup U$ has a large matching that satisfies condition two of Theorem 10.6.

In this case, to satisfy property (i) of Theorem 10.6 for edges of H between S and T, we need that vertices in $T \cap L$ should have degree at most $\beta_+ - 2\beta_-/3 \leq (1+\lambda)\beta_+/3$. Given the bound of $2\beta_-/3$ on the degrees of vertices in $S \cap R$, we have that,

$$|T \cap L| \geq (1 - \Theta(\lambda)) \cdot 2 \cdot |S \cap R|.$$

A similar argument also proves that

$$|T \cap R| \geq (1 - \Theta(\lambda)) \cdot \frac{1}{2} \cdot |S \cap L|.$$

Now note that by Claim 10.7, $|S \cap R| = |S \cap L| = |\bar{M}| = \mu(G) - \mu(H \cup M^*_U) \geq \mu(G) - \mu(H \cup U)$, while $|T \cap L| + |T \cap R| = |T| \leq |\bar{A}| + |B| \leq \mu(H \cup U)$. Combining these with the above two bounds, we get that,

$$\mu(H \cup U) \geq (1 - \Theta(\lambda)) \cdot \frac{5}{7} \cdot \mu(G).$$

Thus, in this case, $H \cup U$ has a matching which is a (much) better than $2/3$ approximation.

It is worth mentioning that in this case, the subgraph H may *not* have a matching larger than $3/2 \cdot |\bar{M}| = \mu(G)/2$, which means the first condition of Theorem 10.6 may indeed not hold here.

The above extreme examples suggest that when edge-degrees of $\bar{M}$ are more toward being balanced, the subgraph H has a close to $2/3$-approximate matching, while when edge-degrees are more unbalanced, the matching of $H \cup U$ is strictly better than $2/3$-approximation. This will be the general strategy underlying our proof of Theorem 10.6 in the next subsection. The proof can then be seen more or less as a "smooth interpolation" between these two extreme cases.

The Formal Proof

In the following lemma, we prove a lower bound on $\mu(H)$. This lemma can then be used as follows: if degree of most edges in $\bar{M}$ are "balanced", i.e., both endpoints have degree $\approx \beta_-/2$, then $\mu(H)$ will already be of size $2 \cdot |\bar{M}|$ which will be sufficient for the first condition of Theorem 10.6.

Lemma 10.8 (matching of H is large). *We have* $\mu(H) \geq \frac{\beta_-}{1+4\lambda} \cdot \sum_{(u,v)\in\bar{M}} \frac{1}{\max\{\deg_F(u)\,,\deg_F(v)\}}$.

Proof. For every edge $(u,v) \in \bar{M}$, define $F(u,v)$ as set of edges in F that are incident on u or v. We define the following fractional matching $x \in \mathbb{R}^F$ on edges of F:

- for any edge $e \in F(u,v)$: set $x_e := \frac{1}{1+4\lambda} \cdot \frac{1}{\max\{\deg_F(u)\,,\deg_F(v)\}}$.

Let us now prove that this is indeed a valid fractional matching. For any vertex w matched by $\bar{M}$,

$$x_w := \sum_{e \ni w} x_e \leq \deg_F(w) \cdot \frac{1}{1+4\lambda} \cdot \frac{1}{\deg_F(w)} < 1,$$

thus satisfying the fractional matching constraint.

Now fix a vertex w not matched by $\bar{M}$. Let $u_1, \ldots, u_{\deg_F(w)}$ denote the neighbors of w in F. By definition, all these vertices are matched by $\bar{M}$. Let $v_1, \ldots, v_{\deg_F(w)}$ be the matched pairs of these vertices. We need the following simple claim.

Claim 10.9. *For every* $i \in [\deg_F(w)]$, $\deg_F(w) \leq (1+4\lambda) \cdot \max\{\deg_F(u_i)\,, \deg_F(v_i)\}$.

Proof. We first have the following two equations:

$$\deg_F(w) + \deg_F(u_i) \leq \beta_+, \qquad \text{(by the property } (i) \text{ of Theorem 10.6 statement)}$$
$$\deg_F(u_i) + \deg_F(v_i) = \beta_-. \qquad \text{(by Eq (10.1))}$$

As such,

$$\deg_F(w) - \deg_F(v_i) \leq \beta_+ - \beta_- \leq 2\lambda\beta_- \qquad (\text{as } \lambda \leq 1/2, \text{ and } \beta_- \geq (1-\lambda)\beta_+)$$

Noting that $\max\{\deg_F(u_i)\,, \deg_F(v_i)\} \geq \beta_-/2$ by Eq (10.1), concludes the proof. □

To finalize Lemma 10.8, for any vertex w not matched by $\bar{M}$, we have,

$$x_w := \sum_{e=(w,u_i)} x_e = \sum_{u_i} \frac{1}{1+4\lambda} \cdot \frac{1}{\max\{\deg_F(u_i)\,, \deg_F(v_i)\}} \underset{\text{Claim 10.9}}{\leq} \sum_{u_i} \frac{1}{\deg_F(w)} = 1,$$

thus satisfying the fractional matching constraint. This implies that x is a valid fractional matching.

Finally, the value of this fractional matching is:

$$\sum_{e\in F} x_e = \sum_{(u,v)\in N} \sum_{e\in F(u,v)} x_e = \sum_{(u,v)\in N} \frac{\deg_F(u) + \deg_F(v)}{(1+4\lambda)\cdot\max\{\deg_F(u)\,, \deg_F(v)\}}$$
$$= \frac{\beta_-}{1+4\lambda} \cdot \sum_{(u,v)\in N} \frac{1}{\max\{\deg_F(u)\,, \deg_F(v)\}},$$

where the last equation is by Eq (10.1). As the integrality gap of matching polytope on bipartite graphs is one, we obtain that the desired lower bound on $\mu(H)$. □

We now prove that if on the other hand most edges of $\bar{M}$ are "unbalanced", then $\mu(H \cup U)$ should be sufficiently large. To continue, we need a quick definition. Let S denote the endpoints of the matching $\bar{M}$ and T be the neighborset of these vertices in F. Recall that by Eq (10.1), S and T are disjoint (see Figure 10.1).

Lemma 10.10 (matching of $\mu(H \cup U)$ is large). *We have* $\mu(H \cup U) \geq \frac{|\bar{M}|^2 \cdot \beta_-{}^2}{|\bar{M}| \cdot \beta_- \cdot \beta_+ - \sum_{s \in S}(\deg_F(s))^2}$.

Proof. Since $F \subseteq H$, by property (i) of Theorem 10.6, we have that

$$|F| \cdot \beta_+ \geq \sum_{(u,v) \in F} \deg_F(u) + \deg_F(v) = \sum_{s \in S} (\deg_F(s))^2 + \sum_{t \in T} (\deg_F(t))^2. \qquad (10.2)$$

We can lower bound the second term of the RHS as follows. Recall that sum of quadratics is minimized over all-equal terms. As $\sum_{t \in T} \deg_F(t) = |F|$, this implies that,

$$\sum_{t \in T} (\deg_F(t))^2 \geq \sum_{t \in T} (\frac{|F|}{|T|})^2 = |T| \cdot (\frac{|F|}{|T|})^2 = \frac{|F|^2}{|T|}.$$

By plugging in this bound in Eq (10.2) and moving the terms around, we have that

$$|T| \geq \frac{|F|^2}{|F| \cdot \beta_+ - \sum_s (\deg_F(s))^2} = \frac{|\bar{M}|^2 \cdot \beta_-{}^2}{|\bar{M}| \cdot \beta_- \cdot \beta_+ - \sum_s (\deg_F(s))^2}.$$

(as $|F| = |\bar{M}| \cdot \beta_-$ by Eq (10.1))

Finally, $T \subseteq \bar{A} \cup B$ (as there are no edges between A and $\bar{B}$) and thus by Claim 10.7, $|T| \leq \mu(H \cup U)$ which finalizes the proof. □

Lemma 10.10 can be used as follows: when degree of most edges in $\bar{M}$ are "balanced", the quantity $\sum_s (\deg_F(s))^2$ will be close to $|\bar{M}| \cdot (\beta_-)^2/2$ which implies that $\mu(H \cup U)$ will be almost $2 \cdot |\bar{M}|$; however, when degrees of edges in $\bar{M}$ are "unbalanced", the quantity $\sum_s (\deg_F(s))^2$ *cannot* decrease all the way to $|\bar{M}| \cdot (\beta_-)^2/2$ and thus we can get a higher lower bound on the value of $\mu(H \cup U)$ which breaks the $(2/3)$-approximation.

To finalize the proof of Theorem 10.6, we need the following claim for lower bounding $\sum_{s \in S} (\deg_F(s))^2$ in the RHS of Lemma 10.10, in the cases where RHS of Lemma 10.8 is small.

Claim 10.11. *Suppose* $\sum_{(u,v) \in \bar{M}} \frac{\beta_-}{\max\{\deg_F(u), \deg_F(v)\}} = (2-\gamma) \cdot |\bar{M}|$ *for some* $\gamma \in [0,1)$*; then* $\sum_s (\deg_F(s))^2 \geq |\bar{M}| \cdot \left(\frac{(2+\gamma^2-2\gamma) \cdot \beta_-{}^2}{4+\gamma^2-4\gamma} \right)$.

Proof. The intuition behind the proof is that $\sum_s (\deg_F(s))^2$ term is a quadratic sum and is thus minimized in the most "balanced" case possible under the given constraints. Formally, we define the following vector of vertex degrees $d \in \mathbb{R}^S$ (recall that S is the endpoints of matching $\bar{M}$):

- For any edge $(u, v) \in \bar{M}$, let $d_u := \frac{\beta_-}{2-\gamma}$ and $d_v := \beta_- - d_u$.

Notice that these vertex degrees satisfy the first constraint of Eq (10.1) and that

$$\sum_{(u,v)\in\bar{M}} \frac{\beta_-}{\max\{d_u\,,\,d_v\}} = (2-\gamma)\cdot|\bar{M}|,$$

thus satisfying the assumption of the lemma as well. We now prove that these degrees minimize the quadratic sum, namely,

$$\sum_{s\in S}(\deg_F(s))^2 \geq \sum_{s\in S} d_s^2. \tag{10.3}$$

Suppose there is an edge (u_1, v_1) where $\deg_F(u_1) > d_{u_1}$ and thus $\deg_F(v_1) < d_{v_1}$ (as both pairs satisfy Eq (10.1)). This also implies that there is another edge (u_2, v_2) where $\deg_F(u_2) < d_{u_2}$ and $\deg_F(v_2) > d_{v_2}$ so that the sum of all degrees satisfies the condition of Eq (10.1).

Now consider a sufficiently small parameter $\theta_1 \in (0, 1)$ and the new "more balanced" degrees

$$\hat{d}_{u_1} := \deg_F(u_1) - \theta_1 \quad , \quad \hat{d}_{v_1} := \deg_F(v_1) + \theta_1,$$
$$\hat{d}_{u_2} := \deg_F(u_2) + \theta_2 \quad , \quad \hat{d}_{v_2} := \deg_F(v_2) - \theta_2,$$

where θ_2 is defined using the following equation:

$$\frac{1}{\deg_F(u_1)} + \frac{1}{\deg_F(u_2)} = \frac{1}{\deg_F(u_1) - \theta_1} + \frac{1}{\deg_F(u_2) + \theta_2} = \frac{1}{\hat{d}_{u_1}} + \frac{1}{\hat{d}_{u_2}}.$$

Considering $\deg_F(u_1) > \deg_F(u_2)$, we have that $\theta_1 > \theta_2$. Note that these new degrees (assuming we keep the degrees of all other vertices unchanged) satisfy all the constraints as before. We have,

$$\begin{aligned}
\sum_{s\in\{u_1,v_1,u_2,v_2\}} \deg_F(s)^2 &= (\hat{d}_{u_1}+\theta_1)^2 + (\hat{d}_{v_1}-\theta_1)^2 + (\hat{d}_{u_2}-\theta_2)^2 + (\hat{d}_{v_2}+\theta_2)^2 \\
&\geq 2\theta_1\cdot(\hat{d}_{u_1}-\hat{d}_{v_1}) - 2\theta_2\cdot(\hat{d}_{u_2}-\hat{d}_{v_2}) + \hat{d}_{u_1}^2 + \hat{d}_{v_1}^2 + \hat{d}_{u_2}^2 + \hat{d}_{v_2}^2 \\
&\qquad\qquad \text{(by ignoring the postive } \theta_1^2, \theta_2^2 \text{ terms)} \\
&> \hat{d}_{u_1}^2 + \hat{d}_{v_1}^2 + \hat{d}_{u_2}^2 + \hat{d}_{v_2}^2 \qquad \text{(as } \hat{d}_{u_1}-\hat{d}_{v_1} > \hat{d}_{u_2}-\hat{d}_{v_2} \text{ and } \theta_1 > \theta_2\text{)}
\end{aligned}$$

Thus, this change reduces the value of $\sum_{s\in S}\deg_F(s)^2$ term as expected. We can now repeatedly continue this until we converge to the degree distribution $\{d_s\}_{s\in S}$ defined earlier. This proves Eq (10.3). By plugging in the bounds for $\{d_s\}_{s\in S}$ in the RHS of Eq (10.3), we have that,

$$\sum_{s\in S}(\deg_F(s))^2 \geq \sum_{s\in S}(\deg_F(s))^2 = \sum_{(u,v)\in\bar{M}} d_u^2 + d_v^2 = |\bar{M}|\cdot\left(\frac{{\beta_-}^2}{(2-\gamma)^2} + (\beta_- - \frac{\beta_-}{(2-\gamma)})^2\right)$$
$$= |\bar{M}|\cdot\left(\frac{(2+\gamma^2-2\gamma)\cdot{\beta_-}^2}{4+\gamma^2-4\gamma}\right),$$

as desired. □

Proof of Theorem 10.6. Let us pick $\gamma\in[0,1)$ such that $\sum_{(u,v)\in\bar{M}}\frac{\beta_-}{\max\{\deg_F(u),\deg_F(v)\}} = (2-\gamma)\cdot|\bar{M}|$ (as the max-term is at least $\beta_-/2$, such a γ always exist). By plugging in the bound of Claim 10.11 in Lemma 10.10, we have that,

$$\mu(H\cup U) \geq \frac{|\bar{M}|^2\cdot{\beta_-}^2}{|\bar{M}|\cdot\beta_-\cdot\beta_+ - |\bar{M}|\cdot\left(\frac{(2+\gamma^2-2\gamma)\cdot{\beta_-}^2}{4+\gamma^2-4\gamma}\right)}$$
$$\geq (1-2\lambda)\cdot|\bar{M}|\cdot\frac{1}{1-\left(\frac{(2+\gamma^2-2\gamma)}{4+\gamma^2-4\gamma}\right)} \qquad (\text{as } \beta_-\geq(1-\lambda)\beta_+)$$
$$= (1-2\lambda)\cdot|\bar{M}|\cdot\frac{4+\gamma^2-4\gamma}{2-2\gamma} = (1-2\lambda)\cdot|\bar{M}|\cdot(2+\frac{\gamma^2}{2-2\gamma}).$$

Considering $|\bar{M}|\geq\mu(G)-\mu(H\cup U)$ by Claim 10.7, we obtain that

$$\mu(H\cup U)\geq(1-2\lambda)\cdot\mu(G)\cdot\left(\frac{2}{3}+\frac{\gamma^2}{18-18\gamma+3\gamma^2}\right)\geq(1-2\lambda)\cdot\mu(G)\cdot\left(\frac{2}{3}+\frac{\gamma^2}{18}\right).$$

Now if for the parameter δ in Theorem 10.6, we already have $\gamma\geq\delta$, we will obtain the second condition. Further, without loss of generality, we can assume that $|\bar{M}|\geq(\frac{1}{3}-\frac{\delta}{3})\cdot\mu(G)$ as otherwise $\mu(H\cup M_U^*)\geq(\frac{2}{3}+\delta)\cdot\mu(G)$ by Claim 10.7 which is stronger than the second condition of Theorem 10.6.

Suppose $\gamma<\delta$ and $|\bar{M}|\geq(\frac{1}{3}-\frac{\delta}{3})\cdot\mu(G)$ then. In this case, by the definition of γ and Lemma 10.8,

$$\mu(H)\geq\frac{1}{1+4\lambda}\cdot(2-\gamma)\cdot|\bar{M}|\geq\frac{1}{1+4\lambda}\cdot(2-\delta)\cdot(\frac{1}{3}-\frac{\delta}{3})\cdot\mu(G)\geq(1-4\lambda)\cdot\left(\frac{2}{3}-\delta\right)\cdot\mu(G),$$

thus satisfying the first condition. This concludes the proof. □

10.3.2 General Graphs

We now extend the results of Theorem 10.6 to general (non-bipartite) graphs following the probabilistic method technique of [13] for the original EDCS.

Corollary 10.12. *Let $\lambda \in (0, 1/2)$ and $\beta_- \leq \beta_+$ be such that $\beta_+ \geq \frac{64}{\lambda^2} \cdot \log(1/\lambda)$ and $\beta_- \geq (1-\lambda)\beta_+$. Suppose $G = (V, E)$ is any graph (not necessarily bipartite) and:*

(i) H is a subgraph of G where for all $(u, v) \in H$: $\deg_H(u) + \deg_H(v) \leq \beta_+$; and

(ii) U is the set of all edges (u, v) in $G \setminus H$ such that $\deg_H(u) + \deg_H(v) < \beta_-$.

Then, for any parameter $\delta \in (0, 1)$, either:

$$\mu(H) \geq (1-8\lambda) \cdot (\frac{2}{3} - \delta) \cdot \mu(G) \quad \textit{or} \quad \mu(H \cup U) \geq (1-4\lambda) \cdot \left(\frac{2}{3} + \frac{\delta^2}{18}\right) \cdot \mu(G).$$

Proof. The proof is based on the probabilistic method and Lovász Local Lemma. Let M^* be a maximum matching of G. Consider the following randomly chosen bipartite subgraph $\tilde{G} = (L, R, \tilde{E})$ of G with respect to M^*, where $L \cup R = V$:

- For any edge $(u, v) \in M^*$, with probability $1/2$, u belongs to L and v belongs to R, and with probability $1/2$, the opposite (the choices between different edges of M^* are independent).

- For any vertex $w \in V$ not matched by M^*, we assign w to L or R uniformly at random (again, the choices are independent across vertices).

- The set of edges in $\tilde{E}$ are all edges in E with one end point in L and the other one in R.

Note that by the definition of $\tilde{G}$, every edge of M^* belongs to $\tilde{G}$ as well and thus $\mu(\tilde{G}) = \mu(G)$. Define $\tilde{H} := H \cap \tilde{G}$ and $\tilde{U} := U \cap \tilde{G}$. We prove that with non-zero probability:

(i) For all $(u, v) \in \tilde{H}$: $\deg_{\tilde{H}}(u) + \deg_{\tilde{H}}(v) \leq (1+\lambda) \cdot \beta_+/2$;

(ii) $\tilde{U}$ is the set of all edges (u, v) in $\tilde{G} \setminus \tilde{H}$ where $\deg_{\tilde{H}}(u) + \deg_{\tilde{H}}(v) < (1-\lambda)\beta_-/2$;

Before proving these parts, let us mention how they imply Corollary 10.12. Consider the subgraph $\tilde{G}$ of G and the sets $\tilde{H}$ and $\tilde{U}$. Since $\tilde{G}$ is bipartite and $\tilde{H}$ and $\tilde{U}$ satisfy the requirements of Theorem 10.6 for parameters $\tilde{\beta_+} = (1+\lambda) \cdot \beta_+/2$, $\tilde{\beta_-} = (1-\lambda)\beta_-/2$, and $\tilde{\lambda} = \lambda/2$, we get either

$$\mu(\tilde{H}) \geq (1-8\lambda) \cdot (\frac{2}{3} - \delta) \cdot \mu(\tilde{G}) \quad \text{or} \quad \mu(\tilde{H} \cup \tilde{U}) \geq (1-4\lambda) \cdot \left(\frac{2}{3} + \frac{\delta^2}{18}\right) \cdot \mu(\tilde{G}).$$

As $\tilde{H} \subseteq H$, $\tilde{U} \subseteq U$, and $\mu(\tilde{G}) = \mu(G)$, we obtain the final result (notice that for this argument, we only need existence of $\tilde{H}$ and $\tilde{U}$ and not a way of finding them; as such, the non-zero probability guarantee completely suffices for us).

To prove either property, we need the following auxiliary claim.

Claim 10.13. *With non-zero probability, for every vertex $v \in V$, $|\deg_{\tilde{H}}(v) - \deg_H(v)/2| < \frac{\lambda}{4} \cdot \beta_-$.*

Proof. Fix any vertex $v \in V$ and let $N_H(v) := \left\{u_1, \ldots, u_{\deg_H(v)}\right\}$ be the neighbors of v in H. Let us assume v is assigned to L in $\tilde{G}$ (the other case is symmetric). Hence, degree of v in $\tilde{H}$ is exactly equal to the number of vertices in $N_H(v)$ that are chosen in R. By construction of $\tilde{G}$,

$$\mathbf{E}\left[\deg_{\tilde{H}}(v)\right] = \begin{cases} (\deg_H(v)+1)/2 & \text{if } v \text{ is incident on } M^* \cap H \\ \deg_H(v)/2 & \text{otherwise} \end{cases}.$$

Also, if two vertices u_i, u_j in $N_H(v)$ are matched by M^*, then exactly one of them will be a neighbor to v in $\tilde{H}$; otherwise the choices are independent. Thus, by Chernoff bound (Proposition 2.1),

$$\mathbf{Pr}\left(|\deg_{\tilde{H}}(v) - \deg_H(v)/2| \geq \frac{\lambda}{4} \cdot \beta_-\right) \leq 2\exp\left(-\frac{\lambda^2 \cdot \beta_-^2}{8\beta_-}\right) \leq 2\exp\left(-4\log\beta_+\right) \leq \frac{2}{\beta_+^4}.$$

(as $\beta_+ \geq 64\lambda^{-2}\log(1/\lambda)$ and $\beta_- \geq (1-\lambda)\beta_+$, we have $\beta_- \geq 32\lambda^{-2} \cdot \log\beta_+$)

For every vertex $v \in V$, define:

- event $\mathcal{E}_v$: the event that $|\deg_{\tilde{H}}(v) - d_v/2| \geq \frac{\lambda}{4} \cdot \beta_-$.

The event $\mathcal{E}_v$ depends only on the choice of vertices in $N_H(v)$ and hence can depend on at most β_+^2 other events $\mathcal{E}_u$ for vertices u which are neighbors to $N_H(v)$. As such, we can apply Lovasz Local Lemma (Proposition 2.5) to argue that with a non-zero probability, $\cap_{v \in V}\overline{\mathcal{E}_v}$ happens, which concludes the proof. ☐

In the following, we condition on the non-zero probability event of Claim 10.13.

Proof of property (*i*)**.** For any edge $(u,v) \in \tilde{H}$, we have,

$$\deg_{\tilde{H}}(u) + \deg_{\tilde{H}}(v) \leq \frac{1}{2} \cdot (\deg_H(u) + \deg_H(v)) + \frac{\lambda}{2} \cdot \beta_- \leq \beta_+/2 + \frac{\lambda}{2} \cdot \beta_- \leq (1+\lambda) \cdot \beta_+/2,$$

where the second to last inequality is because $(u,v) \in H$. As such all edge $(u,v) \in \tilde{H}$ have the desired bound on edge-degree.

Proof of property *(ii)*. For any edge $(u,v) \in \tilde{G} \setminus \tilde{H}$ with $\deg_{\tilde{H}}(u)+\deg_{\tilde{H}}(v) < (1-\lambda)\cdot\beta_-/2$,

$$\deg_H(u) + \deg_H(v) \leq 2 \cdot \left(\deg_{\tilde{H}}(u) + \deg_{\tilde{H}}(v)\right) + \frac{\lambda}{2}\cdot\beta_- < (1-\lambda)\cdot\beta_- + \frac{\lambda}{2}\cdot\beta_- < \beta_-.$$

This implies that this edge belongs to U and thus since $\tilde{U} := \tilde{G} \cap U$, it also belongs to $\tilde{U}$. As a result, any edge with "low" edge-degree belongs to U.

This concludes the proof. □

10.4 An Improved Algorithm via Augmentation

In this section, we show that the maximum matching of the subgraph H constructed in the early part of the stream of Algorithm 3 can be augmented well via the remaining edges. Combined with our Corollary 10.12 of Section 10.3, we complete in this section the proof of Theorem 10.2. Namely, we show that for some parameter $\varepsilon_0 > 0$, there is a single-pass random-order streaming algorithm (formalized as Line 16) that obtains a $(\frac{2}{3}+\varepsilon_0)$-approximate maximum matching of G using $O(n \log n)$ space with high probability of $1 - 1/\operatorname{poly}(n)$.

10.4.1 The Algorithm

Our starting point is Algorithm 3. Recall that this algorithm stores two subgraphs H and U of G of size $O(n \log n)$. Subgraph H is constructed early on, after merely observing εm edges of the stream. In addition to H and U, here we store an additional subset of edges that we use to augment a matching of H with. Particularly, let M_H be an arbitrary maximum matching of H. Having matching M_H early on, in our algorithm we augment M_H using the edges that arrive in the rest of the stream (i.e., Phase II) in parallel to storing U. The augmenting paths that we find may be of size up to *five*. This is crucial since we may not have enough augmenting paths of length smaller than five to go beyond $(2/3)$-approximation. Now by plugging our bound of Corollary 10.12, it can be shown that either $H \cup U$ includes our desired approximation of strictly better that $2/3$, or M_H is almost a $(2/3)$-approximate matching which coupled with the augmenting paths that we find for it in Phase II leads to our better-than-$(2/3)$-approximation.

To find these augmenting paths, we divide the $(1-\varepsilon)m$ edges of Phase II into Phase II.A and Phase II.B. To do this, we first draw a random variable $\tau \sim \mathcal{B}((1-\varepsilon)m, \gamma)$. Phase II.A will then proceed on the edges that arrive up to the τ-th edge of Phase II and Phase II.B proceeds on the rest of the edges. Drawing random variable τ (instead of having a fixed threshold) is particularly useful in the analysis: Conditioned on the edges that are to arrive in Phase II (but not their ordering), each edge now belongs to Phase II.A *independently* with probability γ and to Phase II.B otherwise. Note that with a fixed threshold, we do not get this independence.

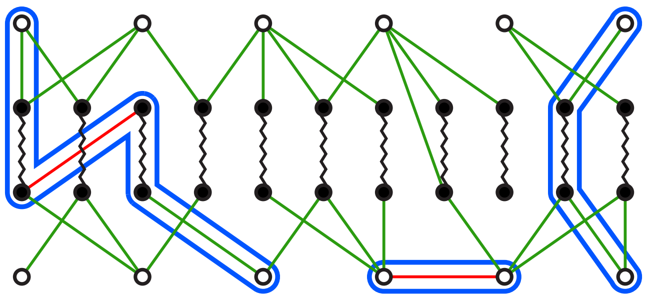


Figure 10.2: An example of an execution of Line 16. Here the black zig-zagged edges are those in matching M_H which is fixed by the end of Phase I and we would like to augment it. The black nodes are those matched by M_H and the white ones are those left unmatched by M_H. The edges between white and black nodes (colored green) are the edges in T. Each black node has at most two edges in T and the green nodes can have up to b. The red edges are those that arrive in Phase II.B. Three augmenting paths of length one, three, and five that are discoverable by the algorithm are also highlighted in the figure.

For Phase II.A, let us define G_H to be the subgraph of G whose edges arrive in Phase II.A and have exactly one endpoint matched by M_H. Note that G_H is bipartite (even though G may not be) with one partition corresponding to vertices $V(M_H)$ and another to $V \setminus V(M_H)$. In Phase II.A, we only consider the edges of G_H and greedily construct a maximal $(2, b)$-matching T of G_H (for some constant $b \geq 2$). It is the vertices in partition $V(M_H)$ of G_H that have maximum degree 2 in T and those in the other partition can have degree up to b. In our analysis, we show that the edges of T can be used as the two endpoint edges of many augmenting paths of length three or five for M_H (see Figure 10.2).

In Phase II.B, we first let $M \leftarrow M_H$ and upon arrival of each edge e, we iteratively augment M via length-up-to-five augmenting paths using the edges in $T \cup \{e\}$ until no such path is left. In our analysis, we use the edges of Phase II.B either as the middle edge of length-five augmenting paths or as the single edge of the length-one augmenting paths the algorithm may find (see Figure 10.2).

At the end of the stream, we return a maximum matching of $M \cup H \cup U$. The algorithm outlined above is formalized as Algorithm 16.

Algorithm 16: Our random-order streaming approximate matching algorithm.

1 **Parameters:** $\gamma = 2/3$, $b = 500$, and a sufficiently small constant $\varepsilon < 0.01$ to be fixed later.

(1) In Phase I of the algorithm, which consists of the first εm edges of the stream, we construct a subgraph H of G as in Phase I of Algorithm 3. At the end of Phase I, we fix an arbitrary maximum matching M_H of H.

(2) In Phase II, which includes all the edges that arrive after Phase II, we store subgraph U using Phase II of Algorithm 3. In addition, we store another subset of edges that we use to augment M_H. These edges are constructed in two sub-phases Phase II.A and Phase II.B.

(3) Draw random variable τ from the Binomial distribution $\mathcal{B}((1-\varepsilon)m, \gamma)$. Note that this can be done in $O(m)$ time and $O(1)$ space as we only need a counter to count the successes.

(4) Phase II.A starts after Phase I and ends upon arrival of the τ'th edge of Phase II.

  (a) Let $G_H(V_H, U_H, E_H)$ be a bipartite subgraph of G where $V_H := V(M_H)$ is the set of vertices matched in M_H, $U_H := V \setminus V(M_H)$ is the set of vertices left unmatched in M_H, and E_H is the edges of G between V_H and U_H that arrive in Phase II.A.

  (b) We initialize $T \leftarrow \emptyset$ and upon arrival of an edge $e = (u, v)$ of G_H with $u \in U_H$ and $v \in V_H$, if $\deg_T(v) < 2$ and $\deg_T(u) < b$ we add e to T. That is, T is a maximal $(2, b)$-matching of G_H which requires $O(nb)$ space to store.

(5) Phase II.B starts after Phase II.A and continues to the end of the stream:

  (a) $M \leftarrow M_H$. Upon arrival of each edge e in Phase II.B, we iteratively take an arbitrary augmenting path P for M of $\underline{\text{length up to five}}$ using the edges in $M \cup T \cup \{e\}$ and let $M \leftarrow M \oplus P$. We repeat this process until no more augmenting paths of length up to five exist in $M \cup T \cup \{e\}$; we then continue to the next edge of the stream in Phase II.B.

(6) Finally, we return a maximum matching of $M \cup H \cup U$.

Space Complexity

We know already from Lemma 10.5 that $|H \cup U| = O(n\log(n) \cdot \text{poly}(1/\varepsilon)) = O(n \log n)$ for constant ε with high probability. In addition, subgraph T that we store in the memory has maximum degree $b = O(1)$ and thus requires $O(n)$ space to store. Other than these, we only store a matching M and augment it only using the edges stored in memory. Hence, overall, the space complexity of the algorithm is $O(n \log n)$ with high probability.

Analysis of Approximation Ratio

Let $M^\star$ be an arbitrary maximum matching of $G_{\geq \varepsilon m}$. Fixing an arbitrary maximum matching of G, each of its edges appears in $G_{\geq \varepsilon m}$ with probability $(1-\varepsilon)$, thus $\mathbf{E}\,|M^\star| \geq (1-\varepsilon)\mu(G)$. Now so long as $\mu(G) \geq 20\log(n)\varepsilon^{-2}$ and $\varepsilon < 1/2$ (which we can assume to hold as discussed in Section 10.2), we can prove a high probability lower bound on the size of $M^\star$ via a Chernoff bound on negatively associated random variables. See, e.g., [50, Lemma 2.2] for the proof of the following:

Observation 10.14. *If $\mu(G) \geq 20\log(n)\varepsilon^{-2}$ and $\varepsilon < 1/2$, then $\mathbf{Pr}[|M^\star| \geq (1-2\varepsilon)\mu(G)] \geq 1 - n^{-5}$.*

From now on, we condition on $G_{<\varepsilon m}$ which fixes subgraph H and matching $M^\star$. We only assume that $G_{<\varepsilon m}$ is chosen such that the high probability event of Observation 10.14 holds.

Assumption 10.15. $|M^\star| \geq (1-2\varepsilon)\mu(G)$.

Other than Assumption 10.15, we do not need any other assumption on how $G_{<\varepsilon m}$ is chosen for the rest of the analysis of the approximation ratio.[3] By conditioning on the outcome of Phase I, the only randomization that will be left, is the order with which the edges of $G_{\geq \varepsilon m}$ arrive in the stream. For brevity, we do not explicitly write the conditioning on $G_{<\varepsilon m}$ for the rest of the section, but it should be noted that **all random statements are conditioned on the outcome of Phase I**.

Let $\mathcal{P}$ be the set of all augmenting paths of M_H in $S := M^\star \Delta M_H$ with length at most five. Note that since we regard H (and thus M_H) as given, the set $\mathcal{P}$ is deterministic (as it only depends on M_H and $M^\star$ and not on the order of edges in $G_{\geq \varepsilon m}$).

[3]We note, however, that the randomization in $G_{<\varepsilon m}$ is crucial for arguing that the algorithm uses $O(n \log n)$ space. Here, however, we are only analyzing the approximation ratio.

Observation 10.16. *We have* $|\mathcal{P}| \geq |M^\star| - \frac{4}{3} \cdot \mu(H)$.

Proof. Let $\mathcal{P}'$ denote the set of augmenting paths of length larger than 5 in S. Note that there must be at least $|M^\star| - |M_H|$ augmenting paths for M_H in S, hence $|\mathcal{P}| + |\mathcal{P}'| \geq |M^\star| - |M_H|$. Moreover, any augmenting path in $\mathcal{P}'$ must have at least 3 edges of M_H; thus $|\mathcal{P}'| \leq |M_H|/3$. Combination of the two bounds gives $|\mathcal{P}| \geq |M^\star| - |M_H| - \frac{1}{3}|M_H| = |M^\star| - \frac{4}{3}|M_H| = |M^\star| - \frac{4}{3}\mu(H)$. □

We use $G_{\text{II.A}}$ to denote the subgraph of G that arrives in Phase II.A and use $G_{\text{II.B}}$ to denote the subgraph of G that arrives in Phase II.B.

Definition 10.17. *We say an augmenting path* $P \in \mathcal{P}$ *is "lucky" under these conditions:*

1. *If* $P = \langle e_1 \rangle$ *then* $e_1 \in G_{II.B}$.
2. *If* $P = \langle e_1, e_2, e_3 \rangle$ *then* $e_1, e_3 \in G_{II.A}$.
3. *If* $P = \langle e_1, e_2, e_3, e_4, e_5 \rangle$ *then* $e_1, e_5 \in G_{II.A}$ *and* $e_3 \in G_{II.B}$.

We denote the set of lucky augmenting paths in $\mathcal{P}$ *by* $\mathcal{P}_L$.

Note that the subset $\mathcal{P}_L$ of $\mathcal{P}$ is now random since it depends on the order of edges in $G_{\geq \varepsilon m}$. Lemma 10.18 below proves that a relatively large fraction of augmenting paths in $\mathcal{P}$ will turn out to be lucky with high probability. The proof is straightforward and is given in Section 10.4.3.

Lemma 10.18. *It holds that* $\mathbf{Pr}\left(|\mathcal{P}_L| \leq \gamma^2(1-\gamma)|\mathcal{P}| - \sqrt{15\mu(G)\ln n}\right) \leq 2n^{-5}$.

Next, observe that in Phase II.B of Line 16 where we iteratively discover augmenting paths, we do not have the whole subgraph $G_{\text{II.A}}$ and have stored only a subgraph T of $G_{\text{II.A}}$ in the memory. In addition, when finding augmenting paths we use only the current edge e of $G_{\text{II.B}}$ in Line 16. Therefore, not all lucky paths are actually discoverable by Line 16. This motivates our next definition for "discoverable paths".

Definition 10.19. *An augmenting path* P *(not necessarily in* $\mathcal{P}$*) for* M_H *is* discoverable *if* $|P| \leq 5$, *all edges of* P *are in* $M_H \cup T \cup G_{II.B}$, *and* P *has at most one edge in* $G_{II.B}$.

The next lemma proves there are many vertex-disjoint discoverable augmenting paths, by relating them to the number of lucky augmenting paths $|\mathcal{P}_L|$. We provide the proof in Section 10.4.2.

Lemma 10.20. *There exists a set $\mathcal{Q}$ of vertex-disjoint discoverable augmenting paths for M_H with*

$$|\mathcal{Q}| \geq \frac{1}{2b+3}\left(|\mathcal{P}_L| - \frac{4}{b}\cdot \mu(H)\right).$$

Observe that $\mathcal{Q}$ is only a set of vertex-disjoint discoverable augmenting paths. However, since Line 16 applies augmenting paths greedily and in an arbitrary order, the set of applied augmenting paths may be very different from $\mathcal{Q}$. The next claim shows that we can nonetheless relate the number of augmenting paths that Line 16 applies to the size of $\mathcal{Q}$.

Claim 10.21. *Let $\mathcal{Q}$ be as in Lemma 10.20. Line 16 applies at least $|\mathcal{Q}|/6$ augmenting paths in* Phase II.B. *In other words, $|M| \geq \mu(H) + \frac{1}{6}|\mathcal{Q}|$.*

Proof. Take an augmenting path $P \in \mathcal{Q}$. Since P is discoverable, there must be a moment during Phase II.B of Line 16 where all the edges of P are stored in the memory. Note, however, that P is by definition an augmenting path for M_H whereas Line 16 tries to augment matching M (which is the result of iteratively augmenting M_H). The crucial observation, here, is that if P is not an augmenting path for M, then at some point one of the augmenting paths that Line 16 has applied on M must have intersected with P (through a vertex). Now, recall that each augmenting paths that Line 16 applies has length at most five, and thus has at most six vertices. This means that any augmenting path that Line 16 applies can intersect (and thus "destroy") at most six paths in $\mathcal{Q}$ (since recall $\mathcal{Q}$ is a collection of vertex-disjoint paths). Hence Line 16 must apply at least $|\mathcal{Q}|/6$ augmenting paths on M. Since each augmenting path increases the size of M by one and initially $M = M_H$, we have $|M| \geq |M_H| + \frac{1}{6}|\mathcal{Q}| = \mu(H) + \frac{1}{6}|\mathcal{Q}|$. □

Lemma 10.22. *There is an absolute constant $\varepsilon_0' > 0$ such that for any $\varepsilon < 0.01$, if $\mu(H) \leq 0.68\mu(G)$ then with probability $1 - 1/\operatorname{poly}(n)$, we have $|M| \geq \mu(H) + \varepsilon_0' \cdot \mu(G)$.*

Proof. We have

$$|M| \overset{\text{Claim 10.21}}{\geq} \mu(H) + \frac{1}{6}|\mathcal{Q}| \overset{\text{Lemma 10.20}}{\geq} \mu(H) + \frac{|\mathcal{P}_L| - \frac{4}{b}\mu(H)}{6(2b+3)}. \tag{10.4}$$

On the other hand, by Lemma 10.18 we know that with $1 - 1/\operatorname{poly}(n)$ probability,

$$\begin{aligned}
|\mathcal{P}_L| &> \gamma^2(1-\gamma)|\mathcal{P}| - \sqrt{15\mu(G)\ln n} && \text{(By Lemma 10.18)}\\
&= \frac{4}{27}|\mathcal{P}| - \sqrt{15\mu(G)\ln n} && \text{(Since } \gamma = 2/3\text{)}\\
&\geq \frac{4}{27}\left(|M^\star| - \frac{4}{3}\mu(H)\right) - \sqrt{15\mu(G)\ln n} && \text{(By Observation 10.16)}\\
&\geq \frac{4}{27}\left((1-2\varepsilon)\mu(G) - \frac{4}{3}\mu(H)\right) - \sqrt{15\mu(G)\ln n} && \text{(By Assumption 10.15)}\\
&> 0.0108\mu(G) - \sqrt{15\mu(G)\ln n} && (\varepsilon < 0.01 \text{ and } \mu(H) \leq 0.68\mu(G))\\
&> 0.01\mu(G). && \text{(Since } \mu(G) > c\log n \text{ for any desirably large constant } c.\text{)}
\end{aligned}$$

Replacing this high probability lower bound for $|\mathcal{P}_L|$ into (10.4) we get that w.h.p.,

$$\begin{aligned}
|M| &\geq \mu(H) + \frac{0.01\mu(G) - \frac{4}{b}\mu(H)}{6(2b+3)}\\
&> \mu(H) + 10^{-7}\mu(G). && \text{(Replacing } b = 500 \text{ and noting } \mu(H) \leq 0.68\mu(G).\text{)}
\end{aligned}$$

This completes the proof. □

We are now ready to prove that Line 16, w.h.p., achieves a better-than-(2/3) approximation.

Lemma 10.23. *For some absolute constant $\varepsilon_0 > 0$ the matching returned by Line 16 with probability $1 - 1/\operatorname{poly}(n)$ has size at least $(2/3 + \varepsilon_0) \cdot \mu(G)$.*

Proof. Let M_O be the matching returned by Line 16 which has size at least as large as maximum of $|M|$ and $\mu(H \cup U)$; we thus get $|M_O| \geq \max\{|M|, \mu(H \cup U)\}$. Hence, from the lower bound of Lemma 10.22 for $|M|$, we get that there is a constant $\varepsilon_0' > 0$ such that with probability $1 - 1/\operatorname{poly}(n)$,

$$|M_O| \geq \max\left\{\mu(H) + \varepsilon_0' \cdot \mu(G),\ \mu(H \cup U)\right\}. \tag{10.5}$$

In the next step, we employ Corollary 10.12 to argue that the lower bound above implies that $|M_O| \geq (2/3 + \Omega(1))\mu(G)$. In particular, let us consider subgraph G' of G which includes all the edges in H as well as all the edges in $G_{>\varepsilon m}$. In other words, the only edges of G that do not belong to G' are those that arrive in Phase I and are not included in subgraph H. One can verify that H and U (constructed in Algorithm 16) satisfy the constraints of Corollary 10.12 for graph G' (but not necessarily G since the edges in $G - G'$ may have a small edge-degree). Corollary 10.12 thus implies that for any $\delta \in (0,1)$, either:

$$\mu(H) \geq (1-8\lambda) \cdot (\frac{2}{3} - \delta) \cdot \mu(G') \quad \text{or} \quad \mu(H \cup U) \geq (1-4\lambda) \cdot \left(\frac{2}{3} + \frac{\delta^2}{18}\right) \cdot \mu(G').$$

Recall that $M^\star$ is the maximum matching of $G_{>\varepsilon m}$ which is entirely included in G'. Also recall from Observation 10.14 that w.h.p. $|M^\star| \geq (1-2\varepsilon)\mu(G)$. Hence, w.h.p., $\mu(G') \geq (1-2\varepsilon)\mu(G)$ which combined with $\lambda = \varepsilon/128$ (Definition 10.4) simplifies the equation above to the following:

$$\mu(H) \geq (1-O(\varepsilon)) \cdot (\frac{2}{3} - \delta) \cdot \mu(G) \quad \text{or} \quad \mu(H \cup U) \geq (1-O(\varepsilon)) \cdot \left(\frac{2}{3} + \frac{\delta^2}{18}\right) \cdot \mu(G). \tag{10.6}$$

Plugging (10.6) into (10.5) implies for any $\delta \in (0,1)$ that

$$\begin{aligned} |M_O| &\geq (1-O(\varepsilon)) \cdot \min\left\{\left(\frac{2}{3} - \delta\right)\mu(G) + \varepsilon_0'\mu(G), \left(\frac{2}{3} + \frac{\delta^2}{18}\right)\mu(G)\right\} \\ &\geq (1-O(\varepsilon)) \cdot \min\left\{\left(\frac{2}{3} - \delta + \varepsilon_0'\right), \left(\frac{2}{3} + \frac{\delta^2}{18}\right)\right\} \cdot \mu(G). \end{aligned}$$

(Note that inequality above takes minimum of the two terms whereas (10.5) takes maximum. This is because Corollary 10.12 only guarantees either the lower bound of $\mu(H)$ or that of $\mu(H \cup U)$ and we do not know which one holds for our instance.)

Now letting $\delta = \varepsilon_0'/2$, we get

$$\begin{aligned} |M_O| &\geq (1-O(\varepsilon)) \cdot \min\left\{\left(\frac{2}{3} + \frac{\varepsilon_0'}{2}\right), \left(\frac{2}{3} + \frac{(\varepsilon_0'/2)^2}{18}\right)\right\} \cdot \mu(G) \\ &\geq (1-O(\varepsilon))\left(\frac{2}{3} + \frac{(\varepsilon_0'/2)^2}{18}\right)\mu(G). \end{aligned}$$

Finally, noting that ε can be made arbitrarily small (without affecting ε_0'), combined with the fact that ε_0' is an absolute positive constant, we get that there must be some $\varepsilon_0 > 0$ such that $|M_O| \geq \left(\frac{2}{3} + \varepsilon_0\right)\mu(G)$ with probability $1 - 1/\operatorname{poly}(n)$. □

Theorem 10.2 now follows immediately from this.

10.4.2 Proof of Lemma 10.20

Observe that not all augmenting path $P \in \mathcal{P}_L$ are discoverable. For example, if $P \in \mathcal{P}_L$ is of length five, despite its two endpoints e_1 and e_5 being part of $G_{\text{II.A}}$ by Definition 10.17, it may still be the case that $e_1, e_5 \notin T$ and thus $e_1, e_5 \notin M_H \cup T \cup G_{\text{II.B}}$ implying that P may not be discoverable. To prove Lemma 10.20, however, we show in this section that for most augmenting paths $P \in \mathcal{P}_L$, we can modify P, particularly, by changing its two endpoint edges (if any and if necessary) and turn P into a discoverable augmenting path $\phi(P)$.

Take an augmenting path $P \in \mathcal{P}_L$ and recall from definition that $\mathcal{P}_L \subseteq \mathcal{P}$ and thus $|P| \in \{1, 3, 5\}$. We define $\phi(P)$ as follows depending on the size of P:

- $|P| = 1$: In this case, we simply let $\phi(P) \leftarrow P$.
- $|P| = 3$: Let $\langle e_1, e_2, e_3 \rangle$ be the edges in P and note that $e_2 \in M_H$ since P is an augmenting path for M_H. If edges $e'_1, e'_3 \in T$ exist such that $\langle e'_1, e_2, e'_3 \rangle$ forms a length-three augmenting path for M_H, we let $\phi(P) \leftarrow \langle e'_1, e_2, e'_3 \rangle$. Otherwise, $\phi(P) \leftarrow \emptyset$.
- $|P| = 5$: Let $\langle e_1, e_2, e_3, e_4, e_5 \rangle$ be the edges in P. Note that $e_2, e_4 \in M_H$ since P is an augmenting path for M_H and $e_3 \in G_{II.B}$ since $P \in \mathcal{P}_L$. Now if there are edges $e'_1, e'_5 \in T$ such that $\langle e'_1, e_2, e_3, e_4, e'_5 \rangle$ is an augmenting path for M_H, we let $\phi(P)$ to denote this path. Otherwise, $\phi(P) \leftarrow \emptyset$.

The properties enlisted in Observation 10.24 are immediate consequences of construction above:

Observation 10.24. *Let $P \in \mathcal{P}_L$ and suppose $\phi(P) \neq \emptyset$. It holds that*

1. $|\phi(P)| = |P|$.
2. *If $P = \langle e_1, \ldots, e_k \rangle$ and $\phi(P) = \langle e'_1, \ldots, e'_k \rangle$ then $e_i = e'_i$ for any $2 \leq i \leq k - 1$.*
3. *The endpoint vertices of $\phi(P)$ are unmatched in M_H since it is an augmenting path for M_H.*
4. *If $|\phi(P)| > 1$ then the two endpoint edges of $\phi(P)$ belong to T.*
5. *If $\phi(P) \neq \emptyset$, then $\phi(P)$ is discoverable.*

We let $\Phi := \{\phi(P) \mid P \in \mathcal{P}_L, \phi(P) \neq \emptyset\}$. Although each element in Φ is a discoverable augmenting path for M_H, it has to be noted that these augmenting paths may not necessarily be vertex-disjoint. In the first part of the proof, we show that a large fraction of paths in Φ are vertex-disjoint. In the second part, we show that Φ is itself large. The combination of these two, gives that there is a large number of vertex-disjoint paths in Φ.

A Large Fraction of Paths in Φ are Vertex-Disjoint

We first need an auxiliary claim:

Claim 10.25. *Let $P \in \mathcal{P}_L$ and $P' \in \mathcal{P}_L$ be such that $P \neq P'$, $\phi(P) \neq \emptyset$, and $\phi(P') \neq \emptyset$:*

1. *If $\phi(P)$ and $\phi(P')$ intersect at a vertex v, then v is an endpoint of both $\phi(P)$ and $\phi(P')$.*

2. *If $e \in \phi(P)$ then $e \notin \phi(P')$.*

Proof. Note that P and P' are vertex-disjoint since both belong to $\mathcal{P}_L \subseteq \mathcal{P}$. By Observation 10.24 part 2, only the endpoint edges of $\phi(P)$ and $\phi(P')$ may differ from P and P' respectively. Combination of these two observations implies that any vertex v that belongs to both of $\phi(P)$ and $\phi(P')$ must be an endpoint of at least one of the two paths. Now using Observation 10.24 part 3, we get that v cannot be an intermediate vertex of one path and an endpoint of another since an intermediate vertex must be matched in M_H (as both $\phi(P)$ and $\phi(P')$ are augmenting paths for M_H). Hence, v must be an endpoint of $\phi(P)$ and $\phi(P')$.

To prove the second part, we know from the first part that if e belongs to both $\phi(P)$ and $\phi(P')$, then both of the endpoints of e must be endpoints of paths $\phi(P)$ and $\phi(P')$. This means that we should have $|\phi(P)| = |\phi(P')| = 1$ and $P = P'$ contradicting $P \neq P'$. $\square$

The next claim is that a large fraction of paths in Φ are vertex-disjoint.

Claim 10.26. *There is a subset $\mathcal{Q} \subseteq \Phi$ such that all the augmenting paths in $\mathcal{Q}$ are vertex-disjoint and $|\mathcal{Q}| \geq \frac{1}{2b+3}|\Phi|$ where we recall b is the parameter of Line 16.*

Proof. We greedily construct $\mathcal{Q} \subseteq \Phi$ by iterating over the augmenting paths in Φ in an arbitrary order and including in $\mathcal{Q}$ any encountered augmenting path $\phi \in \Phi$ which does not intersect with augmenting paths already added to $\mathcal{Q}$.

Take an augmenting path $\phi(P) \in \Phi$. We know from Claim 10.25 part 1, that any other path $\phi(P') \in \Phi$ that intersects $\phi(P)$ must do so at an endpoint vertex of $\phi(P)$. Furthermore, by Claim 10.25 part 2, $\phi(P')$ and $\phi(P'')$ for $P' \neq P''$ cannot be connected to an endpoint of $\phi(P)$ via the same edge. Hence, any $\phi(P')$ intersecting $\phi(P)$ must do so via a unique edge to an endpoint of P. Since the two endpoint edges of any path $\phi(P')$ of size larger than one belong to T by Observation 10.24 part 4, and that the maximum degree of T is b, there are at most $2b$ such paths intersecting $\phi(P)$. Moreover, at most one path $\phi(P')$ of length one can intersect each endpoint of $\phi(P)$ since $\phi(P') = P'$ for length-one paths and thus all of them are vertex-disjoint. Therefore, overall, $\phi(P)$ intersects at most $2b+2$ other paths $\phi(P')$.

Now every time that we add a path $\phi(P)$ to $\mathcal{Q}$, let us remove the remaining paths in Φ that intersect $\phi(P)$. By our discussion above, every time we add a path to $\mathcal{Q}$, we remove at most $2b+2$ other paths from Φ. Hence $|\mathcal{Q}| \geq \frac{1}{2b+3}|\Phi|$. $\square$

The Set Φ is Large

The main statement that Φ is large is formally given as Claim 10.29. Before proving it, we need two auxiliary Claims 10.27 and 10.28.

Claim 10.27. *Let $P = \langle e_1, \ldots, e_k \rangle$ be an augmenting path of length three or five in $\mathcal{P}_L$. Let us denote the endpoints of e_1 and e_k respectively by (u_1, v_1) and (v_k, u_k) where v_1 is the vertex connected to e_2 and v_k is the vertex connected to e_{k-1}. If it holds that*

$$(e_1 \in T \text{ or } \deg_T(v_1) \geq 2) \text{ and } (e_k \in T \text{ or } \deg_T(v_k) \geq 2), \tag{10.7}$$

then $\phi(P) \neq \emptyset$.

Proof. It suffices from our construction of $\phi(P)$ to show there are edges $e'_1, e'_k \in T$ such that $\langle e'_1, e_2, \ldots, e_{k-1}, e'_k \rangle$ is an augmenting path for M_H. We let $e'_1 \leftarrow e_1$ if $e_1 \in T$ and similarly let $e'_k \leftarrow e_k$ if $e_k \in T$. If $e_1 \notin T$ but still (10.7) holds, then $\deg_T(v_1) \geq 2$. Moreover, by construction of T in Line 16, these two edges of v_1 are in U_H, i.e., the vertices left unmatched by M_H. Note that none of these two edges of v_1 are connected to the intermediate vertices of P since P is an augmenting-path for M_H and hence all of its intermediate vertices are matched by M_H (and so do not belong to U_H). However, it could be that one of these edges is connected to the other endpoint of the augmenting path if the graph is non-bipartite. But this can happen for at most one of the edges of v_1 since there are no parallel edges in the graph, which leaves the other edge as a valid option for e'_1. In a similar way, if $e_k \notin T$, we get $\deg_T(v_k) \geq 2$ under (10.7) and can pick one of these two edges of v_k to be e'_k such that $\langle e'_1, e_2, \ldots, e_{k-1}, e'_k \rangle$ forms an augmenting path for M_H. This completes the proof of the claim that condition (10.7) suffices to get $\phi(P) \neq \emptyset$. □

Claim 10.28. *Let $P \in \mathcal{P}_L$, $e_1 = (u_1, v_1)$, and $e_k = (v_k, u_k)$ be as in Claim 10.27. Suppose that condition (10.7) does not hold for P. Then $\deg_T(u_1) \geq b$ or $\deg_T(u_k) \geq b$.*

Proof. We first argue that both e_1 and e_k are part of graph G_H of Phase II.A of Line 16. Toward this, note that since $P \in \mathcal{P}_L$, we get from Definition 10.17 that $e_1, e_k \in G_{\text{II.A}}$. Moreover, since P is by definition an augmenting path for M_H, its endpoints u_1, u_k must be unmatched in M_H (implying $u_1, u_k \in U_H$) and vertices v_1, v_k which are intermediate vertices of P must be matched in M_H (implying $v_1, v_k \in V_H$). Hence, both e_1 and e_k must belong to G_H (refer to Line 16).

Now let us suppose that (10.7) is false since its first clause is false. That is, ($e_1 \notin T$ and $\deg_T(v_1) < 2$). In this case, knowing that $e_1 \in G_H$, the fact that Line 16 does not add

e_1 to T upon processing e_1 implies that either $\deg_T(v_1) \geq 2$ or $\deg_T(u_1) \geq b$ (see description of Algorithm 16). The former cannot hold or otherwise the first clause of (10.7) would not be false. Hence it should be the case that $\deg_T(u_1) \geq b$. The same argument implies that if (10.7) is false for its second clause, then $\deg_T(u_k) \geq b$. The proof is thus complete. □

Claim 10.29. $|\Phi| \geq |\mathcal{P}_L| - \frac{4}{b} \cdot \mu(H)$.

Proof. Let $\mathcal{X} := \{P \in \mathcal{P}_L \mid \phi(P) = \emptyset\}$. By definition, $\Phi = \mathcal{P}_L \setminus \mathcal{X}$, thus

$$|\Phi| = |\mathcal{P}_L| - |\mathcal{X}|. \tag{10.8}$$

It, therefore, suffices to upper bound the size of $\mathcal{X}$. We do so by double counting the number of edges in T.

Recall that for any $P \in \mathcal{P}_L$, $|P| \in \{1,3,5\}$ by definition of $\mathcal{P}_L$. Moreover, if $|P| = 1$, then by construction $\phi(P) = P \neq \emptyset$ and thus $P \notin \mathcal{X}$. Hence for any $P \in \mathcal{X}$ it holds that $|P| \in \{3,5\}$. Now, by Claim 10.27, condition (10.7) should not hold for any $P \in \mathcal{X}$. This further implies from Claim 10.28 that at least one of the endpoints of each $P \in \mathcal{X}$ must have degree at least b edges in T. Since $\mathcal{X} \subseteq \mathcal{P}_L$ and all augmenting paths in $\mathcal{P}_L$ are vertex disjoint, this means that the endpoints of paths in $\mathcal{X}$ collectively have at least $|\mathcal{X}|b$ edges in T. Moreover, all of these vertices must be on the $U_H = V \setminus V(M_H)$ partition of graph G_H since each $P \in \mathcal{X} \subseteq \mathcal{P}_L$ is an augmenting path for M_H by definition of $\mathcal{P}_L$. Now we give an alternative way of counting the edges in T. Note that any vertex in partition $V_H = V(M_H)$ of G_H, has at most 2 edges in T by construction of T in Algorithm 16. Hence, the number of edges in T can be upper bounded by $2 \cdot |V(M_H)| = 2 \cdot 2|M_H| = 4|M_H|$. As such, we get $|\mathcal{X}|b \leq 4|M_H|$ and thus $|\mathcal{X}| \leq 4|M_H|/b$. Plugging this upper bound for $|\mathcal{X}|$ into (10.8) and noting that $|M_H| = \mu(H)$ completes the proof. □

We are finally ready to formally prove Lemma 10.20:

Proof of Lemma 10.20. Let $\mathcal{Q} \subseteq \Phi$ be as in Claim 10.26. All the paths in $\mathcal{Q}$ are vertex-disjoint. Also:

$$|\mathcal{Q}| \overset{\text{Claim 10.26}}{\geq} \frac{|\Phi|}{2b+3} \overset{\text{Claim 10.29}}{\geq} \frac{1}{2b+3}\left(|\mathcal{P}_L| - \frac{4}{b}\mu(H)\right).$$

The proof of Lemma 10.20 is thus complete. □

10.4.3 Proof of Lemma 10.18

We first lower bound $\mathbf{E}\,|\mathcal{P}_L|$ and then prove Lemma 10.18 via a concentration bound.

Claim 10.30. $\mathbf{E}\,|\mathcal{P}_L| \geq \gamma^2(1-\gamma)|\mathcal{P}|$.

Proof. Recall again that we regard $\mathcal{P}$ as fixed as we have conditioned on the outcome of Phase I. Now whether or not an augmenting path $P \in \mathcal{P}$ turns out to be lucky depends on the arrival ordering of the edges in $G_{\geq \varepsilon m}$. We first show that for any $P \in \mathcal{P}$,

$$\mathbf{Pr}[P \in \mathcal{P}_L] \geq \gamma^2(1-\gamma). \tag{10.9}$$

(Where, recall, we hide the condition on Phase I for brevity in our probabilistic statements.)

The key insight is to note that once we condition on $G_{<\varepsilon m}$, an edge e that is to arrive in Phase II belongs to $G_{\text{II.A}}$ independently (than other edges of Phase II) with probability γ and belongs to $G_{\text{II.B}}$ otherwise (i.e., with probability $(1-\gamma)$). As already discussed at the start of Section 10.4, this follows from the fact that we do not fix the size of Phase II.A in Algorithm 16 but rather choose it from distribution $B((1-\varepsilon)m, \gamma)$. Having this independence, we can prove (10.9) as follows:

Proof of Inequality (10.9). Take an augmenting path $P \in \mathcal{P}$. Since $\mathcal{P}$ includes augmenting paths of length up to five, $|P| \in \{1, 3, 5\}$. We prove (10.9) for all three cases one by one.

First, consider the case where P is of length five and let $P = \langle e_1, e_2, e_3, e_4, e_5 \rangle$. By Definition 10.17, P is lucky if $e_1, e_5 \in G_{II.A}$ and $e_3 \in G_{II.B}$. The former two events happen with probability γ each and the latter happens with probability $(1-\gamma)$. Since the three events, as discussed, are independent, we have

$$\mathbf{Pr}[P \in \mathcal{P}_L] = \gamma^2(1-\gamma) \qquad\qquad \forall P = \langle e_1, e_2, e_3, e_4, e_5 \rangle \in \mathcal{P}.$$

For length-three paths, only the two endpoints should appear in Phase II.A, hence

$$\mathbf{Pr}[P \in \mathcal{P}_L] = \gamma^2 \geq \gamma^2(1-\gamma) \qquad\qquad \forall P = \langle e_1, e_2, e_3 \rangle \in \mathcal{P}.$$

For length-one paths, the single edge of the path should appear in Phase II.B, hence:

$$\mathbf{Pr}[P \in \mathcal{P}_L] = (1-\gamma) \geq \gamma^2(1-\gamma) \qquad\qquad \forall P = \langle e_1 \rangle \in \mathcal{P}.$$

The combination of these cases completes the proof of inequality (10.9).

Proof of Lemma 10.18 via inequality (10.9). By linearity of expectation, we have

$$\mathbf{E}\,|\mathcal{P}_L| = \sum_{P\in\mathcal{P}} \mathbf{Pr}[P \in \mathcal{P}_L] \overset{(10.9)}{\geq} \sum_{P\in\mathcal{P}} \gamma^2(1-\gamma) = \gamma^2(1-\gamma)|\mathcal{P}|. \quad \square$$

We are now ready to prove Lemma 10.18 via a simple Chernoff bound.

Proof of Lemma 10.18. Whether or not an augmenting path $P \in \mathcal{P}$ turns out to be lucky depends on how its odd edges belong to $G_{II.A}$ and $G_{II.B}$. Since all the augmenting paths in $\mathcal{P}$ are by definition vertex-disjoint, and since as discussed edges of $G_{\geq\varepsilon m}$ belong to $G_{II.A}$ and $G_{II.B}$ independently from each other, we get that the paths in $\mathcal{P}$ belong to $\mathcal{P}_L$ independently from each other. By a simple Chernoff bound (Proposition 2.1), letting $\delta = \sqrt{\frac{15\ln n}{\mathbf{E}\,|\mathcal{P}_L|}} > 0$, we have

$$\begin{aligned}\mathbf{Pr}\left(|\mathcal{P}_L| \leq (1-\delta)\,\mathbf{E}\,|\mathcal{P}_L| = \mathbf{E}\,|\mathcal{P}_L| - \sqrt{15\,\mathbf{E}\,|\mathcal{P}_L|\ln n}\right) &\leq 2\exp\left(-\frac{\delta^2\cdot\mathbf{E}\,|\mathcal{P}_L|}{3}\right)\\ &\leq 2\exp(-5\ln n) = 2n^{-5}.\end{aligned}$$

Since $\mathbf{E}\,|\mathcal{P}_L| \geq \gamma^2(1-\gamma)|\mathcal{P}|$ by Claim 10.30 and $\mathbf{E}\,|\mathcal{P}_L| \leq |\mathcal{P}| \leq \mu(G)$ this implies that

$$\mathbf{Pr}\left(|\mathcal{P}_L| \leq \gamma^2(1-\gamma)|\mathcal{P}| - \sqrt{15\mu(G)\ln n}\right) \leq 2n^{-5}. \quad \square$$

Part V

Conclusion and Open Problems

Chapter 11

Conclusion and Open Problems

In this thesis, we revisited a number of fundamental problems for massive graphs, where traditional algorithms are no longer applicable. One of the main challenges with such graphs is that they are often, by orders of magnitude, larger than a single machine's memory. This invalidates several assumptions of traditional algorithms such as the assumption to have random-access to various parts of the graph. In this thesis, we considered various forms of large-scale algorithms that allow efficient processing of such massive graphs. Specifically, we focused on (*i*) *massively parallel computation* algorithms where the workload is distributed to several machines each with sublinear space/communication, (*ii*) *sublinear-time* algorithms that process the input while reading a small fraction of it, (*iii*) *streaming* algorithms that take only few passes over the input having access to a sublinear space, and (*iv*) *dynamic* algorithms that address changes to the input in sublinear time. We presented new algorithms for fundamental graph problems including *maximum matching*, *maximal independent set*, *minimum vertex cover*, and *graph connectivity* in these models that substantially improve upon the state-of-the-art.

We conclude this thesis with a number of important open problems that relate to the models and problems that we considered.

11.1 Open Problems for Massively Parallel Computation

11.1.1 Connectivity Problems

Proving or refuting the 1v2-Cycle conjecture (Conjecture 5.1) is arguably the most fundamental open question in the study of MPC algorithms. This conjecture has been the basis of several conditional lower bounds in the model, even for problems that are seemingly unrelated to graph connectivity. Hence, proving or disproving it will have deep consequences.

Concretely, the open problem is the following:

Open Problem 1. *Suppose that the input is promised to be either a cycle on n vertices or two cycles on $n/2$ vertices each. Does there exist a $o(\log n)$ round algorithm for distinguishing the two cases using $n^{1-\Omega(1)}$ space per machine and $\mathrm{poly}(n)$ total space?*

Recall that the 1v2-Cycle conjecture states that the answer to Open Problem 1 is negative: that there is no such algorithm. A challenge in proving this conjecture is that any $\omega(1)$ round unconditional lower bound in the MPC model with $n^{\Omega(1)}$ local space for a polynomial-time solvable problem would separate NC^1 from P [144], which is a notoriously difficult problem in circuit complexity.

11.1.2 Matching

We showed in Chapter 4 that a maximal matching can be found in $O(\log\log\Delta)$-rounds of MPC using $O(n)$ space per machine and an optimal total space of $O(m)$. This leaves two main open problems. First, can we improve the round-complexity further? Namely,

Open Problem 2. *Does there exist an $O(1)$ round MPC algorithm for finding a maximal matching, or in fact any $O(1)$-approximation of maximum matching, using $O(n)$ local space and $\mathrm{poly}(n)$ total space?*

We note that the $O(\log\log\Delta)$-round algorithm remains the fastest known for any $O(1)$-approximate matching and any $o(\log\log n)$ round algorithm would also be very interesting.

The next question is can we reduce the local space to strictly sublinear in n without blowing up the round-complexity? Namely,

Open Problem 3. *Does there exist a $\mathrm{poly}(\log\log n)$ round MPC algorithm for finding a maximal matching, or in fact any $O(1)$-approximation of maximum matching, using $n^{1-\Omega(1)}$ local space and $\mathrm{poly}(n)$ total space?*

We discussed in Chapter 4 that the local space of our algorithm can be made mildly sublinear in n, namely $n/2^{\Omega(\sqrt{\log n})}$ though note that it is still much larger than $n^{1-\Omega(1)}$. We also showed that an algorithm settles Open Problem 3 so long as $\log\lambda = \mathrm{poly}(\log\log n)$ where recall λ is the arboricity of the graph. The fastest current algorithm for finding a maximal matching with strictly sublinear in n local space for general graphs is that of Ghaffari and Uitto [92] which takes $\widetilde{O}(\sqrt{\log\Delta})$ rounds. See also [95] for some evidence that $\Omega(\log\log n)$ rounds might be needed for any $O(1)$-approximate matching with strictly sublinear space.

We note that both Open Problem 2 and Open Problem 3 are also open for the maximal independent set problem.

11.2 Open Problems for Dynamic Algorithms

We considered the maximal independent set, maximal matching, and approximate maximum matching problems in Part III of the thesis where we discussed dynamic graph algorithms. Here we mention some intriguing open questions related to these problems.

We showed in Chapter 9 that a $(\frac{1}{2} + \Omega_\varepsilon(1))$-approximate matching can be maintained in an arbitrary small polynomial update-time of $O(\Delta^\varepsilon) + \text{poly}(\log n)$ where $\varepsilon > 0$ can be any parameter. It is a long-standing open problem to reduce the update-time to $\text{poly}(\log n)$ while maintaining a strictly-better-than-half approximation:

Open Problem 4. *Does there exist a fully dynamic algorithm maintaining a $(\frac{1}{2} + \varepsilon_0)$-approximate matching for some constant $\varepsilon_0 > 0$ with $\text{poly}(\log n)$ update-time?*

We note this problem is open even if we allow an update-time of up to $n^{o(1)}$, allow randomization against an oblivious adversary, and assume that the graph is bipartite!

Next, we highlight an intriguing open problem regarding deterministic fully dynamic algorithms. Particularly we saw algorithms in Chapters 7 and 8 that maintain a maximal independent set and a maximal matching in $\text{poly}(\log n)$ update-time. However, these algorithms use randomization in a crucial way. One downside, of course, is that there is a tiny chance of failure hence motivating the search for deterministic algorithms. There is another drawback also: both of these mentioned algorithms require the oblivious adversary assumption crucially. That is, the adversary should fix the sequence of updates before the algorithm starts to operate. In some applications of dynamic algorithms the output of the algorithm may affect the future updates, hence invalidating the oblivious adversary assumption. Unfortunately, for both maximal matching and maximal independent set there is a huge gap between the best known algorithms that work against oblivious adversaries and those that work against an adaptive adversaries.

Open Problem 5. *Does there exist fully dynamic algorithms for maintaining a maximal independent set or a maximal matching in $\text{poly}(\log n)$ time deterministically?*

Currently all known algorithms for maintaining an MIS or a MM against adaptive adversaries take a large polynomial update-time.

11.3 Open Problems for Streaming Algorithms

We showed in Chapter 10 that there is a single-pass streaming algorithm that uses $O(n \log n)$ space and returns a $(\frac{2}{3}+\Omega(1))$-approximate maximum matching of the graph at the end of the stream, provided that the edges of the graph arrive in a random order. The important message of this result is that the $\frac{2}{3}$-approximation, which was a barrier for all previous techniques, is *not* the right answer for this problem. An immediate next question is the following:

Open Problem 6. *What is the best approximation achievable for the one-pass random-order streaming matching problem using $n \operatorname{poly}(\log n)$ space? Does there exist a $(1-\varepsilon)$-approximation for any fixed $\varepsilon > 0$?*

In a result that we did not cover in this thesis, we showed in [12] that there is no $(1-\varepsilon)$-approximate matching algorithm in the random-order streaming model that uses $(\exp(1/\varepsilon)^{0.99} n \operatorname{poly}(\log n))$ space. In other words, an exponential dependence on $1/\varepsilon$ is necessary. But this does not rule out an $\widetilde{O}(n)$ space algorithm for fixed $\varepsilon > 0$ yet.

Another intriguing and long-standing open question is for adversarial arrivals where the greedy half approximation achieved via a maximal matching remains the best known algorithm.

Open Problem 7. *Does there exist a $(\frac{1}{2}+\Omega(1))$-approximate one-pass streaming algorithm using $n \operatorname{poly}(\log n)$ space under adversarial-order edge arrivals?*

This problem is open even if one allows all the way up to $n^{2-\Omega(1)}$ space!

Bibliography

[1] Kook Jin Ahn and Sudipto Guha. Access to Data and Number of Iterations: Dual Primal Algorithms for Maximum Matching under Resource Constraints. *ACM Trans. Parallel Comput.*, 4(4):17:1–17:40, 2018.

[2] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 459–467. SIAM, 2012.

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

[4] Noga Alon and Joel H. Spencer. *The Probabilistic Method, Third Edition.* Wiley-Interscience series in discrete mathematics and optimization. Wiley, 2008. ISBN 978-0-470-17020-5.

[5] Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-Efficient Local Computation Algorithms. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 1132–1139.

[6] Noga Alon, László Babai, and Alon Itai. A Fast and Simple Randomized Parallel Algorithm for the Maximal Independent Set Problem. *J. Algorithms*, 7(4):567–583, 1986.

[7] Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. Space-efficient local computation algorithms. In Rabani [141], pages 1132–1139. ISBN 978-1-61197-210-8.

[8] Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 574–583. ACM, 2014.

[9] Alexandr Andoni, Zhao Song, Clifford Stein, Zhengyu Wang, and Peilin Zhong. Parallel Graph Connectivity in Log Diameter Rounds. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 674–685. IEEE Computer Society, 2018.

[10] Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. Dynamic Matching: Reducing Integral Algorithms to Approximately-Maximal Fractional Algorithms. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 7:1–7:16, 2018.

[11] Sepehr Assadi. Simple Round Compression for Parallel Vertex Cover. *CoRR*, abs/1709.04599, 2017.

[12] Sepehr Assadi and Soheil Behnezhad. Beating Two-Thirds For Random-Order Streaming Matching. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021, July 12-16, 2021, Glasgow, Scotland (Virtual Conference)*, volume 198 of *LIPIcs*, pages 19:1–19:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[13] Sepehr Assadi and Aaron Bernstein. Towards a Unified Theory of Sparsification for Matching Problems. In *2nd Symposium on Simplicity in Algorithms, SOSA@SODA 2019, January 8-9, 2019 - San Diego, CA, USA*, pages 11:1–11:20, 2019.

[14] Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully Dynamic Maximal Independent Set with Sublinear Update Time. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 815–826, 2018.

[15] Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab S. Mirrokni, and Cliff Stein. 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, 2019.

[16] Sepehr Assadi, Yu Chen, and Sanjeev Khanna. Sublinear Algorithms for $(\Delta+1)$ Vertex Coloring. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 767–786. SIAM, 2019.

[17] Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully Dynamic Maximal Independent Set with Sublinear in n Update Time. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1919–1936, 2019.

[18] Sepehr Assadi, Xiaorui Sun, and Omri Weinstein. Massively Parallel Algorithms for Finding Well-Connected Components in Sparse Graphs. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 461–470. ACM, 2019.

[19] Frederico AC Azevedo, Ludmila RB Carvalho, Lea T Grinberg, José Marcelo Farfel, Renata EL Ferretti, Renata EP Leite, Wilson Jacob Filho, Roberto Lent, and Suzana Herculano-Houzel. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *Journal of Comparative Neurology*, 513 (5):532–541, 2009.

[20] Leonid Barenboim, Michael Elkin, Seth Pettie, and Johannes Schneider. The Locality of Distributed Symmetry Breaking. In *53rd Annual IEEE Symposium on Foundations of Computer Science, FOCS 2012, New Brunswick, NJ, USA, October 20-23, 2012*, pages 321–330. IEEE Computer Society, 2012.

[21] Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully Dynamic Maximal Matching in $O(\log n)$ Update Time. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 383–392, 2011.

[22] Surender Baswana, Manoj Gupta, and Sandeep Sen. Fully Dynamic Maximal Matching in $O(\log n)$ Update Time (Corrected Version). *SIAM J. Comput.*, 47(3):617–650, 2018.

[23] MohammadHossein Bateni, Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Raimondas Kiveris, Silvio Lattanzi, and Vahab S. Mirrokni. Affinity Clustering: Hierarchical Clustering at Scale. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6864–6874, 2017.

[24] MohammadHossein Bateni, Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, and Vahab S. Mirrokni. Brief Announcement: MapReduce Algorithms for Massive Trees. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 162:1–162:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[25] Paul Beame, Paraschos Koutris, and Dan Suciu. Communication Steps for Parallel Query Processing. *J. ACM*, 64(6):40:1–40:58, 2017.

[26] Soheil Behnezhad. Time-Optimal Sublinear Algorithms for Matching and Vertex Cover. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, to appear*, 2021.

[27] Soheil Behnezhad and Mahsa Derakhshan. Stochastic Weighted Matching: $(1 - \varepsilon)$ Approximation. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1392–1403. IEEE, 2020.

[28] Soheil Behnezhad and Nima Reyhani. Almost Optimal Stochastic Weighted Matching with Few Queries. In *Proceedings of the 2018 ACM Conference on Economics and Computation, Ithaca, NY, USA, June 18-22, 2018*, pages 235–249. ACM, 2018.

[29] Soheil Behnezhad, Sina Dehghani, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, and Saeed Seddighin. Faster and Simpler Algorithm for Optimal Strategies of Blotto Game. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, pages 369–375. AAAI Press, 2017.

[30] Soheil Behnezhad, Mahsa Derakhshan, Hossein Esfandiari, Elif Tan, and Hadi Yami. Brief Announcement: Graph Matching in Massive Datasets. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2017, Washington DC, USA, July 24-26, 2017*, pages 133–136. ACM, 2017.

[31] Soheil Behnezhad, Mahsa Derakhshan, Mohammad Taghi Hajiaghayi, and Aleksandrs Slivkins. A Polynomial Time Algorithm for Spatio-Temporal Security Games. In *Proceedings of the 2017 ACM Conference on Economics and Computation, EC '17, Cambridge, MA, USA, June 26-30, 2017*, pages 697–714. ACM, 2017.

[32] Soheil Behnezhad, Avrim Blum, Mahsa Derakhshan, Mohammad Taghi Hajiaghayi, Mohammad Mahdian, Christos H. Papadimitriou, Ronald L. Rivest, Saeed Seddighin, and Philip B. Stark. From Battlefields to Elections: Winning Strategies of Blotto and

Auditing Games. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 2291–2310. SIAM, 2018.

[33] Soheil Behnezhad, Mahsa Derakhshan, Mohammad Taghi Hajiaghayi, and Saeed Seddighin. Spatio-Temporal Games Beyond One Dimension. In *Proceedings of the 2018 ACM Conference on Economics and Computation, Ithaca, NY, USA, June 18-22, 2018*, pages 411–428. ACM, 2018.

[34] Soheil Behnezhad, Mahsa Derakhshan, and MohammadTaghi Hajiaghayi. Brief Announcement: Semi-MapReduce Meets Congested Clique. *CoRR*, abs/1802.10297, 2018.

[35] Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, and Richard M. Karp. Massively Parallel Symmetry Breaking on Sparse Graphs: MIS and Maximal Matching. *CoRR*, abs/1807.06701, 2018.

[36] Soheil Behnezhad, Avrim Blum, Mahsa Derakhshan, Mohammad Taghi Hajiaghayi, Christos H. Papadimitriou, and Saeed Seddighin. Optimal Strategies of Blotto Games: Beyond Convexity. In *Proceedings of the 2019 ACM Conference on Economics and Computation, EC 2019, Phoenix, AZ, USA, June 24-28, 2019*, pages 597–616. ACM, 2019.

[37] Soheil Behnezhad, Sebastian Brandt, Mahsa Derakhshan, Manuela Fischer, MohammadTaghi Hajiaghayi, Richard M. Karp, and Jara Uitto. Massively Parallel Computation of Matching and MIS in Sparse Graphs. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 481–490. ACM, 2019.

[38] Soheil Behnezhad, Mahsa Derakhshan, Alireza Farhadi, MohammadTaghi Hajiaghayi, and Nima Reyhani. Stochastic Matching on Uniformly Sparse Graphs. In *Algorithmic Game Theory - 12th International Symposium, SAGT 2019, Athens, Greece, September 30 - October 3, 2019, Proceedings*, volume 11801 of *Lecture Notes in Computer Science*, pages 357–373. Springer, 2019.

[39] Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Marina Knittel, and Hamed Saleh. Streaming and Massively Parallel Algorithms for Edge Coloring. In *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 15:1–15:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[40] Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Cliff Stein, and Madhu Sudan. Fully Dynamic Maximal Independent Set with Polylogarithmic Update Time. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 382–405. IEEE Computer Society, 2019.

[41] Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Lacki, and Vahab S. Mirrokni. Near-Optimal Massively Parallel Graph Connectivity. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1615–1636. IEEE Computer Society, 2019.

[42] Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Lacki, Vahab S. Mirrokni, and Warren Schudy. Massively Parallel Computation via Remote Memory Access. In *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 59–68. ACM, 2019.

[43] Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Lacki, Vahab S. Mirrokni, and Warren Schudy. Massively Parallel Computation via Remote Memory Access. In *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 59–68. ACM, 2019.

[44] Soheil Behnezhad, Alireza Farhadi, MohammadTaghi Hajiaghayi, and Nima Reyhani. Stochastic Matching with Few Queries: New Algorithms and Tools. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2855–2874. SIAM, 2019.

[45] Soheil Behnezhad, MohammadTaghi Hajiaghayi, and David G. Harris. Exponentially Faster Massively Parallel Maximal Matching. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1637–1649. IEEE Computer Society, 2019.

[46] Soheil Behnezhad, Mahsa Derakhshan, and MohammadTaghi Hajiaghayi. Stochastic Matching with Few Queries: $(1-\varepsilon)$ Approximation. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1111–1124. ACM, 2020.

[47] Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, Jakub Lacki, Vahab S. Mirrokni, and Warren Schudy. Parallel Graph Algorithms in Constant Adaptive Rounds: Theory meets Practice. *Proc. VLDB Endow.*, 13(13):3588–3602, 2020.

[48] Soheil Behnezhad, Jakub Lacki, and Vahab S. Mirrokni. Fully Dynamic Matching: Beating 2-Approximation in Δ^{ε} Update Time. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2492–2508. SIAM, 2020.

[49] Claude Berge. *The theory of graphs.* Courier Corporation, 1962.

[50] Aaron Bernstein. Improved Bounds for Matching in Random-Order Streams. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, pages 12:1–12:13, 2020.

[51] Aaron Bernstein and Cliff Stein. Fully Dynamic Matching in Bipartite Graphs. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 167–179, 2015.

[52] Aaron Bernstein and Cliff Stein. Faster Fully Dynamic Matchings with Small Approximation Ratios. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 692–711, 2016.

[53] Aaron Bernstein, Sebastian Forster, and Monika Henzinger. A Deamortization Approach for Dynamic Spanner and Dynamic Maximal Matching. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1899–1918, 2019.

[54] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. New Deterministic Approximation Algorithms for Fully Dynamic Matching. *CoRR*, abs/1604.05765, 2016.

[55] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. New deterministic approximation algorithms for fully dynamic matching. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 398–411, 2016.

[56] Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. Deterministic Fully Dynamic Approximate Vertex Cover and Fractional Matching in $O(1)$ Amortized Update Time. In *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, pages 86–98, 2017.

[57] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. Fully Dynamic Approximate Maximum Matching and Minimum Vertex Cover $O(\log^3 n)$ Worst Case Update Time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 470–489, 2017.

[58] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Deterministic Fully Dynamic Data Structures for Vertex Cover and Matching. *SIAM J. Comput.*, 47(3): 859–887, 2018.

[59] Guy E. Blelloch, Jeremy T. Fineman, and Julian Shun. Greedy Sequential Maximal Independent Set and Matching are Parallel on Average. In *24th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '12, Pittsburgh, PA, USA, June 25-27, 2012*, pages 308–317, 2012.

[60] Guy E. Blelloch, Jeremy T. Fineman, and Julian Shun. Greedy sequential maximal independent set and matching are parallel on average. In *24th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '12, Pittsburgh, PA, USA, June 25-27, 2012*, pages 308–317, 2012.

[61] Béla Bollobás and Oliver Riordan. The Diameter of a Scale-Free Random Graph. *Combinatorica*, 24(1):5–34, 2004.

[62] Béla Bollobás and Oliver M Riordan. Mathematical results on scale-free random graphs. *Handbook of graphs and networks: from the genome to the internet*, pages 1–34, 2003.

[63] Karl Bringmann, Fabian Kuhn, Konstantinos Panagiotou, Ueli Peter, and Henning Thomas. Internal DLA: Efficient Simulation of a Physical Growth Model - (Extended Abstract). In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 247–258, 2014.

[64] Keren Censor-Hillel, Elad Haramaty, and Zohar S. Karnin. Optimal Dynamic Distributed MIS. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 217–226, 2016.

[65] Moses Charikar and Shay Solomon. Fully Dynamic Almost-Maximal Matching: Breaking the Polynomial Worst-Case Time Barrier. In *45th International Colloquium on Au-*

tomata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic, pages 33:1–33:14, 2018.

[66] Moses Charikar, Weiyun Ma, and Li-Yang Tan. Unconditional Lower Bounds for Adaptive Massively Parallel Computation. In *SPAA '20: 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, July 15-17, 2020*, pages 141–151. ACM, 2020.

[67] Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the Minimum Spanning Tree Weight in Sublinear Time. *SIAM J. Comput.*, 34(6):1370–1379, 2005.

[68] Shiri Chechik and Tianyi Zhang. Fully Dynamic Maximal Independent Set in Expected Poly-Log Update Time. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2019, to appear*, 2019.

[69] Yu Chen, Sampath Kannan, and Sanjeev Khanna. Sublinear Algorithms and Lower Bounds for Metric TSP Cost Estimation. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, pages 30:1–30:19, 2020.

[70] Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. One Trillion Edges: Graph Processing at Facebook-Scale. *Proc. VLDB Endow.*, 8(12):1804–1815, 2015.

[71] Fan Chung. Graph theory in the information age. *Notices of the AMS*, 57(6):726–732, 2010.

[72] Sam Coy and Artur Czumaj. Deterministic Massively Parallel Connectivity. *CoRR*, abs/2108.04102, 2021.

[73] Pilu Crescenzi, Roberto Grossi, Michel Habib, Leonardo Lanzi, and Andrea Marino. On computing the diameter of real-world undirected graphs. *Theor. Comput. Sci.*, 514: 84–95, 2013.

[74] Michael S. Crouch, Andrew McGregor, and Daniel Stubbs. Dynamic Graphs in the Sliding-Window Model. In *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, volume 8125 of *Lecture Notes in Computer Science*, pages 337–348. Springer, 2013. ISBN 978-3-642-40449-8.

[75] Artur Czumaj and Christian Sohler. Estimating the Weight of Metric Minimum Spanning Trees in Sublinear Time. *SIAM J. Comput.*, 39(3):904–922, 2009.

[76] Artur Czumaj and Christian Sohler. Sublinear-time Algorithms. In *Property Testing - Current Research and Surveys*, volume 6390 of *Lecture Notes in Computer Science*, pages 41–64. Springer, 2010.

[77] Artur Czumaj and Christian Sohler. Sublinear Time Approximation of the Cost of a Metric k-Nearest Neighbor Graph. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2973–2992. SIAM, 2020.

[78] Artur Czumaj, Jakub Lacki, Aleksander Madry, Slobodan Mitrovic, Krzysztof Onak, and Piotr Sankowski. 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, 2018.

[79] Sebastian Daum, Seth Gilbert, Fabian Kuhn, and Calvin C. Newport. Leader election in shared spectrum radio networks. In *ACM Symposium on Principles of Distributed Computing, PODC '12, Funchal, Madeira, Portugal, July 16-18, 2012*, pages 215–224, 2012.

[80] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[81] Yuhao Du and Hengjie Zhang. Improved Algorithms for Fully Dynamic Maximal Independent Set. *CoRR*, abs/1804.08908, 2018.

[82] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

[83] Hossein Esfandiari and Michael Mitzenmacher. Metric Sublinear Algorithms via Linear Sampling. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 11–22. IEEE Computer Society, 2018.

[84] Martin Farach-Colton and Meng-Tsung Tsai. Exact Sublinear Binomial Sampling. *Algorithmica*, 73(4):637–651, 2015.

[85] Alireza Farhadi, Mohammad Taghi Hajiaghayi, Tung Mai, Anup Rao, and Ryan A. Rossi. Approximate Maximum Matching in Random Streams. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1773–1785, 2020.

[86] Illés J Farkas, Imre Derényi, Albert-László Barabási, and Tamas Vicsek. Spectra of "real-world" graphs: Beyond the semicircle law. *Physical Review E*, 64(2):026704, 2001.

[87] Uriel Feige. On Sums of Independent Random Variables with Unbounded Variance and Estimating the Average Degree in a Graph. *SIAM J. Comput.*, 35(4):964–984, 2006.

[88] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3): 207–216, 2005.

[89] Manuela Fischer and Andreas Noever. Tight Analysis of Parallel Randomized Greedy MIS. *ACM Trans. Algorithms*, 16(1):6:1–6:13, 2020.

[90] Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. Weighted Matchings via Unweighted Augmentations. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 491–500, 2019.

[91] Mohsen Ghaffari. Distributed MIS via All-to-All Communication. In *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 141–149, 2017.

[92] Mohsen Ghaffari and Jara Uitto. 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, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1636–1653. SIAM, 2019.

[93] Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrovic, and Ronitt Rubinfeld. Improved Massively Parallel Computation Algorithms for MIS, Matching, and Vertex Cover. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed (PODC)*, pages 129–138, 2018.

[94] Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrovic, and Ronitt Rubinfeld. Improved Massively Parallel Computation Algorithms for MIS, Matching,

and Vertex Cover. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, July 23-27, 2018*, pages 129–138, 2018.

[95] Mohsen Ghaffari, Fabian Kuhn, and Jara Uitto. Conditional Hardness Results for Massively Parallel Computation from Distributed Lower Bounds. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 1650–1663. IEEE Computer Society, 2019.

[96] Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the Communication and Streaming Complexity of Maximum Bipartite Matching. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 468–485. SIAM, 2012.

[97] Oded Goldreich and Dana Ron. Approximating Average Parameters of Graphs. *Random Struct. Algorithms*, 32(4):473–493, 2008.

[98] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, Searching, and Simulation in the MapReduce Framework. In *Proceedings of the 22nd International Symposium on Algorithms and Computation (ISAAC)*, pages 374–383, 2011.

[99] Lei Gu, Hui Lin Huang, and Xiao Dong Zhang. The clustering coefficient and the diameter of small-world networks. *Acta Mathematica Sinica, English Series*, 29(1): 199–208, 2013.

[100] Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 537–550, 2017.

[101] Manoj Gupta and Shahbaz Khan. Simple dynamic algorithms for Maximal Independent Set and other problems. *CoRR*, abs/1804.01823, 2018.

[102] Manoj Gupta and Richard Peng. Fully Dynamic $(1 + \varepsilon)$-Approximate Matchings. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 548–557, 2013.

[103] Philip Hall. On representatives of subsets. *Journal of the London Mathematical Society*, 1(1):26–30, 1935.

[104] Amos Israeli and Alon Itai. A Fast and Simple Randomized Parallel Algorithm for Maximal Matching. *Inf. Process. Lett.*, 22(2):77–80, 1986.

[105] Michael Kapralov. Better bounds for matchings in the streaming model. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1679–1697, 2013.

[106] Michael Kapralov. Space Lower Bounds for Approximating Maximum Matching in the Edge Arrival Model. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, 2021.

[107] Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Approximating matching size from random streams. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 734–751. SIAM, 2014.

[108] Michael Kapralov, Slobodan Mitrovic, Ashkan Norouzi-Fard, and Jakab Tardos. 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, 2020.

[109] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A Model of Computation for MapReduce. In *Proceedings of the 21st annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 938–948, 2010.

[110] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A Model of Computation for MapReduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 938–948. SIAM, 2010.

[111] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within $2 - \varepsilon$. *Journal of Computer and System Sciences*, 74(3):335–349, 2008.

[112] Raimondas Kiveris, Silvio Lattanzi, Vahab S. Mirrokni, Vibhor Rastogi, and Sergei Vassilvitskii. Connected Components in MapReduce and Beyond. In *Proceedings of the ACM Symposium on Cloud Computing, Seattle, WA, USA, November 3-5, 2014*, pages 18:1–18:13. ACM, 2014. ISBN 978-1-4503-3252-1.

[113] Christof Koch. *Biophysics of computation: information processing in single neurons.* Oxford university press, 2004.

[114] Christian Konrad. A Simple Augmentation Method for Matchings with Applications to Streaming Algorithms. In *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, pages 74:1–74:16, 2018.

[115] Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum Matching in Semi-Streaming with Few Passes. *CoRR*, abs/1112.0184, 2011.

[116] Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum Matching in Semi-streaming with Few Passes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 15th International Workshop, APPROX 2012, and 16th International Workshop, RANDOM 2012, Cambridge, MA, USA, August 15-17, 2012. Proceedings*, pages 231–242, 2012.

[117] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. Local Computation: Lower and Upper Bounds. *J. ACM*, 63(2):17:1–17:44, 2016.

[118] Jakub Lacki, Vahab S. Mirrokni, and Michal Wlodarczyk. Connected Components at Scale via Local Contractions. *CoRR*, abs/1807.10727, 2018.

[119] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: a method for solving graph problems in MapReduce. In *Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 85–94. ACM, 2011. ISBN 978-1-4503-0743-7.

[120] Christoph Lenzen. Optimal deterministic routing and sorting on the congested clique. In *ACM Symposium on Principles of Distributed Computing, PODC*, pages 42–50, 2013.

[121] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data, June 2014.

[122] Nathan Linial. Distributive Graph Algorithms-Global Solutions from Local Data. In *Proceedings of the 28th annual Symposium on Foundations of Computer Science (FOCS)*, pages 331–335, 1987.

[123] Sixue Cliff Liu, Robert E. Tarjan, and Peilin Zhong. Connected Components on a PRAM in Log Diameter Time. In *SPAA '20: 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, July 15-17, 2020*, pages 359–369, 2020.

[124] Zvi Lotker, Boaz Patt-Shamir, and Adi Rosén. Distributed Approximate Matching. *SIAM J. Comput.*, 39(2):445–460, 2009.

[125] Linyuan Lu. The diameter of random massive graphs. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 912–921. Society for Industrial and Applied Mathematics, 2001.

[126] Michael Luby. A Simple Parallel Algorithm for the Maximal Independent Set Problem. In *Proceedings of the 17th annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10, 1985.

[127] Alessandro Lulli, Emanuele Carlini, Patrizio Dazzi, Claudio Lucchese, and Laura Ricci. Fast Connected Components Computation in Large Graphs by Vertex Pruning. *IEEE Trans. Parallel Distrib. Syst.*, 28(3):760–773, 2017.

[128] Andrew McGregor. Finding Graph Matchings in Data Streams. In *8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX) and 9th International Workshop on Randomization and Computation (RANDOM)*, pages 170–181, 2005.

[129] Andrew McGregor. Graph stream algorithms: a survey. *SIGMOD Rec.*, 43(1):9–20, 2014.

[130] Michael Mitzenmacher and Eli Upfal. *Probability and computing - randomized algorithms and probabilistic analysis.* Cambridge University Press, 2005. ISBN 978-0-521-83540-4.

[131] Shanmugavelayutham Muthukrishnan. *Data streams: Algorithms and applications.* Now Publishers Inc, 2005.

[132] Ofer Neiman and Shay Solomon. Simple deterministic algorithms for fully dynamic maximal matching. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 745–754, 2013.

[133] Huy N. Nguyen and Krzysztof Onak. Constant-Time Approximation Algorithms via Local Improvements. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 327–336, 2008.

[134] Krzysztof Nowicki and Krzysztof Onak. Dynamic Graph Algorithms with Batch Updates in the Massively Parallel Computation Model. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2939–2958, 2021.

[135] Krzysztof Onak and Ronitt Rubinfeld. Maintaining a large matching and a small vertex cover. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 457–464, 2010.

[136] Krzysztof Onak, Dana Ron, Michal Rosen, and Ronitt Rubinfeld. A Near-Optimal Sublinear-Time Algorithm for Approximating the Minimum Vertex Cover Size. In Rabani [141], pages 1123–1131. ISBN 978-1-61197-210-8.

[137] Krzysztof Onak, Baruch Schieber, Shay Solomon, and Nicole Wein. Fully Dynamic MIS in Uniformly Sparse Graphs. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 92:1–92:14, 2018.

[138] Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theor. Comput. Sci.*, 381(1-3):183–196, 2007.

[139] Michal Parnas and Dana Ron. Approximating the Minimum Vertex Cover in Sublinear Time and a Connection to Distributed Algorithms. *Theor. Comput. Sci.*, 381(1-3): 183–196, 2007.

[140] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach.* Society for Industrial and Applied Mathematics, 2000.

[141] *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, 2012. SIAM. ISBN 978-1-61197-210-8.

[142] Vibhor Rastogi, Ashwin Machanavajjhala, Laukik Chitnis, and Anish Das Sarma. Finding connected components in map-reduce in logarithmic rounds. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 50–61. IEEE Computer Society, 2013. ISBN 978-1-4673-4909-3.

[143] Tim Roughgarden, Sergei Vassilvitskii, and Joshua R. Wang. Shuffles and Circuits: (On Lower Bounds for Modern Parallel Computation). In *Proceedings of the 28th*

ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016, pages 1–12. ACM, 2016. ISBN 978-1-4503-4210-0.

[144] Tim Roughgarden, Sergei Vassilvitskii, and Joshua R. Wang. Shuffles and Circuits: (On Lower Bounds for Modern Parallel Computation). In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016*, pages 1–12, 2016.

[145] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M. Tamer Özsu. The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing. *PVLDB*, 11(4):420–431, 2017.

[146] Yossi Shiloach and Uzi Vishkin. An $O(\log n)$ Parallel Connectivity Algorithm. *J. Algorithms*, 3(1):57–67, 1982.

[147] Shay Solomon. Fully Dynamic Maximal Matching in Constant Update Time. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 325–334, 2016.

[148] J Michael Steele. An Efron-Stein Inequality for Nonsymmetric Statistics. *The Annals of Statistics*, 14(2):753–758, 1986.

[149] Stergios Stergiou, Dipen Rughwani, and Kostas Tsioutsiouliklis. Shortcutting Label Propagation for Distributed Connected Components. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018*, pages 540–546. ACM, 2018.

[150] William T Tutte. The factorization of linear graphs. *Journal of the London Mathematical Society*, 1(2):107–111, 1947.

[151] Uzi Vishkin. An optimal parallel connectivity algorithm. *Discrete Applied Mathematics*, 9(2):197–207, 1984.

[152] Tom White. *Hadoop - The Definitive Guide: Storage and Analysis at Internet Scale (2. ed.)*. O'Reilly, 2011. ISBN 978-1-449-38973-4.

[153] Grigory Yaroslavtsev and Adithya Vadapalli. Massively Parallel Algorithms and Hardness for Single-Linkage Clustering Under ℓ_p-Distances. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm,*

Sweden, July 10-15, 2018, volume 80 of *JMLR Workshop and Conference Proceedings*, pages 5596–5605. JMLR.org, 2018.

[154] Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. An improved constant-time approximation algorithm for maximum matchings. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 225–234. ACM, 2009.

[155] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster Computing with Working Sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2010.