MATCHING IN MASSIVE GRAPHS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF MANAGEMENT SCIENCE AND ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Mohammad Roghani
November 2025

This dissertation is online at: https://purl.stanford.edu/sx221nr8556

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Aviad Rubinstein, Primary Advisor

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Amin Saberi, Primary Advisor

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Moses Charikar

Approved for the Stanford University Committee on Graduate Studies.

Stacey F. Bent, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format.

Abstract

Over the past few decades, advances in computing power have been dramatic, yet they have not kept pace with the exponential growth of data. Traditional algorithm design has long regarded linear-time algorithms as the benchmark of efficiency. However, as data sizes continue to grow, even linear-time algorithms can become inadequate. This motivates the study of sublinear-time algorithms, which seek to extract meaningful information while inspecting only a small portion of the input.

This thesis investigates the power and limitations of sublinear-time algorithms in the context of the maximum matching problem, a cornerstone of combinatorial optimization. Maximum matching has historically played a central role in algorithm design, providing tools and insights that extend far beyond the problem itself. Here, we establish new algorithmic techniques and prove fundamental lower bounds for matching in the sublinear model, thereby resolving several long-standing questions.

In addition, we examine how sublinear matching algorithms along with the techniques developed for them, can be incorporated in other computational models and problems in theoretical computer science. Examples include classical optimization problems such as the Traveling Salesman Problem (TSP), Steiner Tree, Steiner Forest, and Earth Mover's Distance, as well as the maximum matching problem in the dynamic model, where inputs evolve over time. Together, these results advance our understanding of what can be achieved in sublinear time and what the fundamental limits are.

Content

This thesis is based on the author's research conducted during his Ph.D. studies [40, 13, 41, 43, 80, 42, 94, 147, 164, 148, 44, 161, 26, 146, 6, 25, 27, 81] and focuses on the study of graph algorithms in the sublinear time model [40, 41, 43, 42, 147, 148, 44, 26, 146, 27]. In particular, it investigates the problem of estimating the size of a maximum matching in sublinear time. The thesis is organized into three main parts, each addressing a different aspect of the problem.

Chapter 3 studies the problem from an algorithmic perspective, presenting several sublinear time algorithms and exploring the trade-offs between approximation ratio and running time. Chapter 4 focuses on the limitations of these algorithms, establishing lower bounds that characterize the inherent trade-offs in the sublinear model. Finally, Chapter 5 explores applications of sublinear time matching algorithms to other fundamental graph problems and discusses the resulting implications and extensions.

The material presented in Chapter 3 is adapted from the following publications:

- *"Beating Greedy Matching in Sublinear Time"* by Soheil Behnezhad, Mohammad Roghani, Aviad Rubinstein, and Amin Saberi [43]. In the proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms (SODA 2023).
- *"A 0.51-Approximation of Maximum Matching in Sublinear $n^{0.51}$ Time"* by Sepideh Mahabadi, Mohammad Roghani, and Jakub Tarnawski [146]. In the proceedings of the 52^{nd} International Colloquium on Automata, Languages, and Programming (ICALP 2025).
- *"Sublinear Time Algorithms and Complexity of Approximate Maximum Matching"* by Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein [42]. In the proceedings of the 55^{th} Annual ACM Symposium on Theory of Computing (STOC 2023).

The material presented in Chapter 4 is adapted from the following publications:

- *"Sublinear Time Algorithms and Complexity of Approximate Maximum Matching"* by Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein [42]. In the proceedings of the 55^{th} Annual ACM Symposium on Theory of Computing (STOC 2023).
- *"Local Computation Algorithms for Maximum Matching: New Lower Bounds"* by Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein [41]. In the proceedings of the 64^{th} IEEE Annual Symposium on Foundations of Computer Science (FOCS 2023).

- *"Approximating Maximum Matching Requires Almost Quadratic Time"* by Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein [40]. In the proceedings of the 56^{th} Annual ACM Symposium on Theory of Computing (STOC 2024).
- *"Tight Pair Query Lower Bounds for Matching and Earth Mover's Distance"* by Amir Azarmehr, Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein [27]. In the proceedings of the 66^{th} IEEE Annual Symposium on Foundations of Computer Science (FOCS 2025).

The material presented in Chapter 4 is adapted from the following publications:

- *"Fully Dynamic Matching:*$(2-\sqrt{2})$*-Approximation in Polylog Update Time"* by Amir Azarmehr, Soheil Behnezhad, and Mohammad Roghani [26]. In the proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms (SODA 2024).
- *"Sublinear Algorithms for TSP via Path Covers"* by Soheil Behnezhad, Mohammad Roghani, Aviad Rubinstein, and Amin Saberi [44]. In the proceedings of the 51^{st} International Colloquium on Automata, Languages, and Programming (ICALP 2024).
- *"Sublinear Metric Steiner Tree via Improved Bounds for Set Cover"* by Sepideh Mahabadi, Mohammad Roghani, Jakub Tarnawski, and Ali Vakilian [148]. In the proceedings of the the 17^{th} Innovations in Theoretical Computer Science (ITCS 2025).
- *"Sublinear Metric Steiner Forest via Maximal Independent Set"* by Sepideh Mahabadi, Mohammad Roghani, Jakub Tarnawski, and Ali Vakilian [147]. In the proceedings of the 2026 ACM-SIAM Symposium on Discrete Algorithms (SODA 2026).

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my advisors, Aviad Rubinstein and Amin Saberi, for their invaluable guidance, support, and mentorship throughout my PhD journey. I feel truly fortunate to have worked with and learned from both of them. Their insight, creativity, and high standards of research have profoundly shaped the way I think about problems and approach science. I have greatly benefited from their patience, encouragement, and unwavering belief in me, even during challenging times. My PhD has been an immensely enjoyable and rewarding experience, and they are the main reason behind that. Beyond research, I have learned a great deal from them about integrity, perseverance, and balance in life and academia. It has been a privilege to be their student, and I will remain deeply grateful for their mentorship and the opportunities they have provided me.

I would like to sincerely thank the members of my thesis and oral examination committees for their time, guidance, and valuable feedback. In particular, I am grateful to Moses Charikar for serving on both my reading and oral examination committees; his insightful questions and thoughtful suggestions have greatly strengthened my research. I also thank Itai Ashlagi and Jan Vondrak for serving on my oral examination committee. Their perspectives, constructive feedback, and encouragement have been extremely helpful in refining my work and broadening my understanding of the field. I deeply appreciate their contributions to this thesis and to my growth as a researcher.

I am grateful to Soheil Behnezhad for his guidance and mentorship throughout many of my projects. Working with him on multiple papers has been both inspiring and educational. His thoughtful advice, rigorous approach to research, and willingness to discuss ideas in depth have taught me a great deal and shaped my development as a researcher. Beyond technical guidance, I have also learned from him about collaboration, problem-solving, and approaching research with curiosity and rigor. I am thankful for his support, patience, and the opportunity to learn from him over the course of my PhD. I would also like to thank David Wajc for mentoring me at the beginning of my PhD, providing guidance and encouragement that helped me build a strong foundation for my research and set me on the path to where I am today. I am also grateful to my internship mentors, Sepideh Mahabadi and Jakub Tarnawski at Microsoft Research, Mina Dalirrooyfard at Morgan Stanley, and Hossein Safavi at Uber, for their guidance, support, and the valuable lessons I learned from working with them.

Research would not have been as rewarding without the opportunity to work with my brilliant co-authors. I am grateful to Aviad Rubinstein, Amin Saberi, Soheil Behnezhad, Amir Azarmehr, Sepideh Mahabadi, Jakub Tarnawski, Itai Ashlagi, Ali Vakilian, Jiale Chen, Tao Yu, Mohammad Saneian, Mahsa Derakhshan, Tristan Pollner, Ruiquan Gao, Persi Diaconis, Yeganeh Alimohammadi, and David Wajc for their insights,

creativity, and dedication. Collaborating with them has not only advanced this research but also made the process truly enjoyable and inspiring.

My PhD experience has been made all the more memorable and enjoyable thanks to the wonderful friends I have had at Stanford, to whom I am deeply grateful: Bak, Hosein, Pegah, Fatol, Soheil, Raman, Nicole, Sahand, both Alis, Samira, Yasi, Siavash, Dada, Farshid, Nazli, Hasti, Ashkan, Peyman, Setareh, Kiana, Dorsa, Shayan, Alma, Mobin, Mahsa, Shima, both Sinas, Vahid, Rosa, Misagh, Ronak, Kyriakos, Abbas, Sadegh, and Sara.

I am deeply grateful to my parents, Aliasghar and Firoozeh, for their unconditional love, support, and encouragement throughout my life and especially during my PhD. Their belief in me, guidance, and sacrifices have made this journey possible, and I am constantly inspired by their example. I am forever thankful for their patience and care, and for always being my foundation no matter the distance.

I want to give my deepest thanks to my wife, Helia. Your love, patience, and unwavering support at every step have been the cornerstone of my journey. You have celebrated my successes, comforted me through setbacks, and believed in me even during moments when I doubted myself. Being with you has made me feel at home, even when we were far from our families, and your presence has turned challenges into opportunities for growth and moments of joy. You have shared in every part of my PhD, the late nights, the stressful deadlines, the breakthroughs, and the small victories along the way, making these years not only productive but truly unforgettable. Your encouragement, wisdom, and kindness have shaped not only my work but also the person I have become during this journey. I am endlessly grateful for your love, your patience, and the countless ways you have supported me, and I could not have completed this PhD without you by my side.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The continuous advancement of technology has led to a dramatic increase in the complexity and scale of real-world datasets, making them increasingly difficult for traditional algorithms to handle efficiently. A major challenge stems from the massive volume of data, which demands the design of algorithms that are far more efficient than before. Although computing power has improved at an impressive rate, it has been far outpaced by the exponential expansion of data. Consequently, many modern algorithmic problems now involve inputs so large that even reading them in full once is inadequate.

Massive graphs arise naturally in many domains, spanning social networks, knowledge bases, and online platforms. For example, Instagram, with over 3 billion monthly active users, forms a dynamic graph of friendships, followers, and interactions [177]. The English Wikipedia contains more than 7 million articles interconnected through hyperlinks, creating a complex knowledge graph that represents vast human knowledge [174]. Similarly, Facebook's social graph comprises over one trillion edges, capturing the enormous web of relationships among its users [71]. These examples illustrate the immense scale of real-world graphs, highlighting why it is often infeasible to process them entirely and motivating the development of algorithmic models that can operate efficiently with limited access to the data.

To cope with this reality, researchers have developed new models of computation that explicitly account for the massive size of data. In this thesis, we study several fundamental graph problems within these emerging models, with a particular emphasis on the *sublinear time model*—a setting in which algorithms are allowed to read only a small portion of the input and must still produce accurate approximations or estimates. In the sublinear time model, the input is accessed through oracle queries rather than being read in its entirety. The goal is to design algorithms whose running time is sublinear in the input size. Such algorithms can only sample or query limited information about the input—for example, by querying the adjacency list or adjacency matrix of the graph—and must use this partial information to infer global properties of the graph, such as connectivity, matchings, or distances.

The focus of this thesis is on the *maximum matching* problem. Matching is a fundamental problem in computer science with a wide range of applications, including ride-sharing, resource allocation, labor markets, kidney exchange, and online advertising. It has been a central topic in theoretical computer science for over half a century, studied extensively through the lens of algorithm design by pioneering theoreticians. This long-standing focus has led to the development of innovative algorithms and frameworks that address both practical challenges and deep theoretical questions. Many foundational ideas in theoretical computer science

have been shaped by research on maximum matching. Notably, the concept of *polynomial-time solvability* was first rigorously introduced by Jack Edmonds [82] in the context of maximum matching, laying the groundwork for much of modern algorithm design. Although the primary focus is on the maximum matching problem, this thesis also investigates other graph problems, such as *Steiner Forest*, *Steiner Tree*, the *Traveling Salesman Problem*, and the *Earth Mover's Distance*, within the sublinear model. These problems are explored as applications of maximum matching in this model by leveraging the algorithms and techniques developed for maximum matching.

We begin by providing an overview of the computational models considered in this thesis. Although the primary focus is on the sublinear time model, we also explore related frameworks, including the *streaming*, *dynamic*, *local computation algorithm (LCA)*, and *LOCAL* models.

1.1 Computational Models

Sublinear Time Algorithms: Traditionally, linear-time algorithms have been regarded as the benchmark for efficiency in algorithm design. However, as datasets continue to grow in size, even algorithms with linear running time become impractical. This challenge has motivated the study of *sublinear-time algorithms*, which aim to produce approximate solutions while examining only a small portion of the input. Since such algorithms cannot process the entire input, it is essential to precisely define how the algorithm is allowed to access the data. In the context of graph problems, for instance, the input may be represented either through adjacency lists or adjacency matrices, and the algorithm's query capabilities are defined accordingly.

Streaming Algorithms: In the streaming model, the input—such as the edges of a graph G—arrives sequentially in a stream, often in an arbitrary order. The algorithm has limited memory and cannot store the entire input, so it must decide on the fly which information to retain in order to compute the desired property of the graph once the stream ends. A common scenario in large-scale computation is that the input is much larger than the main (random-access) memory but fits in external storage. In such cases, a space-efficient streaming algorithm allows processing the data line by line directly from the external memory, avoiding the need to load the entire input at once.

A particularly well-studied variant is the *semi-streaming model*, where the available space is restricted to $O(n \cdot \operatorname{polylog} n)$ for a graph with n vertices. In this regime, the algorithm typically performs a single pass over the stream, meaning it processes each element only once in the order it arrives. More general versions allow multiple passes, known as the *multi-pass* streaming model, where the algorithm is permitted to scan the stream a few times to improve accuracy or approximation guarantees. In the *random-order* model, the elements of the stream arrive in a uniformly random order, which often enables stronger algorithmic results compared to the adversarial (arbitrary) order setting.

Key parameters that characterize the efficiency of streaming algorithms include the number of passes over the stream, the space complexity (i.e., how much memory the algorithm uses), and the achieved approximation ratio relative to the optimal solution.

Dynamic Algorithms: In many large-scale applications, graphs evolve continuously as edges are inserted or deleted over time. Recomputing the desired property from scratch after every change using the best-known static algorithm is often prohibitively expensive. Dynamic algorithms overcome this limitation by efficiently

maintaining an exact or approximate value of the property as the graph changes, rather than starting from scratch after each update.

In the *fully dynamic model*, we consider a graph on a fixed set of vertices that is subject to both edge insertions and deletions. After each update, the algorithm must maintain or estimate a graph property, such as connectivity, matching size, or the weight of a minimum spanning tree. The goal is to achieve high approximation ratio while minimizing the update time, i.e., the computational cost of processing each modification.

We say the adversary—who determines the sequence of updates—is *oblivious* if she fixes the update sequence in advance, independent of the algorithm's outputs. In contrast, an *adaptive* adversary may choose future updates based on the algorithm's responses, making the design and analysis of efficient dynamic algorithms considerably more challenging.

Local Algorithms: In the distributed LOCAL model, each vertex of the graph hosts a processor, and two processors can exchange an unbounded amount of information in each round if their corresponding vertices are neighbors in the graph. The objective of a LOCAL algorithm is to compute a property of the underlying communication network—such as a matching—within a small number of rounds. The output is typically distributed; for instance, each vertex may indicate the neighbor to which it is matched, if any. The following well-known property of LOCAL algorithms holds: the existence of an r-round LOCAL algorithm for a problem implies that the output of each vertex depends solely on its r-hop neighborhood.

Local Computation Algorithm (LCA): In the local computation algorithm (LCA) model, an algorithm is required to answer queries about the output of a solution without computing the entire solution explicitly. Each query asks for the value of the solution at a specific vertex or edge (for example, whether a vertex is matched in a matching), and the algorithm must respond consistently with some global solution. The goal of an LCA is to answer each query using a small number of local probes into the input graph, ideally much smaller than the total size of the graph.

1.2 Our Contribution

In this thesis, we study several fundamental graph problems within the models discussed above, with a particular focus on the sublinear model and the maximum matching problem. We develop both algorithms and lower bounds for different problems, each improving upon the state of the art in its respective setting. In the following sections, we present these results in detail. For formal definition of each problem, we refer the reader to Section 2.4.

1.2.1 Sublinear Time Algorithms for Maximum Matching

The maximum matching problem in the sublinear setting has been extensively studied in the literature from an algorithmic perspective. However, all previous results suffer from one of the following shortcomings:

- They assume a bound on the maximum degree of the graph, i.e., vertices have constant degree [159, 154, 176, 165, 9, 144, 97]. Under this assumption, the focus is typically on designing algorithms with poly(Δ) running time which is not necessarily $n^{2-\Omega(1)}$ for dense graphs.

- Their approximation ratio is at most 1/2 [34, 132, 68].

Before the recent series of works on sublinear matching, the state-of-the-art algorithm for dense graphs was proposed by Behnezhad [34]. Their approach, which runs in $\widetilde{O}(n)$ time, is based on implementing the random greedy maximal matching algorithm and estimating the size of the resulting matching when the permutation of edges is drawn uniformly at random. Since the algorithm estimates the size of a maximal matching, it achieves a 1/2-approximation. We aim to address the following question:

Question 1. *Is it possible to* $(\frac{1}{2} + \Omega(1))$*-approximate maximum matching size in* $n^{2-\Omega(1)}$ *time?*

We answer this question affirmatively, presenting several algorithms that achieve different trade-offs, which we discuss later in Chapter 3. The following result is formally proved in Section 3.1

Result 1. *For any constant* $\varepsilon > 0$*, there is a constant* $\delta > 2^{-O(1/\varepsilon)}$ *along with an algorithm that w.h.p. estimates the size of maximum matching up to a:*

(1) *multiplicative factor of* $(\frac{1}{2} + \delta)$ *in the adjacency list model in* $\widetilde{O}(n + \Delta^{1+\varepsilon})$ *time,*

(2) *multiplicative-additive factor of* $(\frac{1}{2} + \delta, o(n))$ *in the adjacency list model in* $\widetilde{O}(\bar{d} \cdot \Delta^{\varepsilon})$ *time,*

(3) *multiplicative-additive factor of* $(\frac{1}{2} + \delta, o(n))$ *in the adjacency matrix model in* $O(n^{1+\varepsilon})$ *time.*

On Beating Greedy Matching in Various Settings: The greedy 1/2-approximation is a prevalent barrier for maximum matching across various settings. As a result, numerous works in the literature study the possibility of beating it — both on the upper bound side as well as the lower bound side. The answer is not always the same. For instance, for the online model under *edge arrivals*, [93] showed that 1/2 is provably the best achievable approximation, which can be trivially matched by the greedy algorithm. There are also settings where the answer remains unknown, despite a significant research effort. For instance, in the single-pass streaming setting, beating the greedy 1/2-approximation in $\widetilde{O}(n)$ (or even subquadratic) space has been open for nearly two decades [88], and is often considered as one of the most fundamental open problems of the area. Finally, there are settings for which the greedy 1/2-approximation has been broken. Various models of the online setting [87, 93], the random-order streaming setting [140], and the stochastic matching setting [20] are examples of this. In many of these settings, the approximation has been improved well beyond 1/2 after the greedy bound was first broken. For instance, in the random-order streaming the current best known bound is slightly above 2/3 [15], and in the stochastic matching setting a $(1 - \varepsilon)$-approximation has been achieved [36]. We hope that our work also inspires future work on going tangibly above 1/2-approximation in the sublinear time model.

Discovering short *augmenting paths* has been a central technique in many of the works discussed above in beating the greedy algorithm. What varies significantly is whether it is possible to find these augmenting paths effeciently in the particular model at hand. In particular, a common approach is to first construct a maximal matching in full, and then augment it via the vertices left unmatched. This "adaptivity" complicates things in our model, making it hard to estimate the size of the solution in subquadratic time. One of our main contributions in this work is to give a less "adaptive" algorithm that interleaves the construction of a maximal matching and the augmentation phase. We believe this technique, which is overviewed in Section 3.1.1, might be of independent interest.

Next, we show that one can surpass the 0.5-approximation factor using a simpler algorithm that runs in strongly sublinear time. The formal proof of this result appears in Section 3.2.

Result 2. *There exists an algorithm that estimates the size of maximum matching in $\widetilde{O}(n\sqrt{n})$ time with*

- *a multiplicative approximation factor of 0.5109 given access to the adjacency list model, and*
- *a multiplicative-additive approximation factor of $(0.5109, o(n))$ given access to the adjacency matrix model.*

It is worth emphasizing that our algorithm is significantly simpler—both conceptually and analytically—than the previous result.

Next, we show that it is possible to achieve an approximation ratio substantially greater than 1/2, in contrast to previous results that improved upon the 1/2-approximation only marginally. The following two results are formally stated in Section 3.3.2.

Result 3. *For any fixed $\varepsilon > 0$, there are algorithms for approximating the maximum matching size of any (general) n-vertex graph that take $n^{2-\Omega_\varepsilon(1)}$ time and obtain*

- *a multiplicative $(2/3 - \varepsilon)$-approximation in the adjacency list model, and*
- *a multiplicative-additive $(2/3 - \varepsilon, o(n))$-approximation in the adjacency matrix model.*

Result 4. *There are algorithms for approximating the maximum matching size of any bipartite n-vertex graph that take $n^{2-\Omega(1)}$ time and obtain*

- *a multiplicative $(2/3 + \Omega(1))$-approximation in the adjacency list model, and*
- *a multiplicative-additive $(2/3 + \Omega(1), o(n))$-approximation in the adjacency matrix model.*

Subsequent Work: Independently and concurrently with these results, Bhattacharya, Kiss, and Saranurak [59] proposed a sublinear-time algorithm that achieves a $(2/3 - \varepsilon)$-approximation in $n^{2-\mathrm{poly}(\varepsilon)}$ time, for any arbitrarily small constant $\varepsilon > 0$. In a follow-up work, Bhattacharya, Kiss, and Saranurak [57] designed an algorithm that achieves a $(1 - \varepsilon)$-approximation in $n^{2-f(\varepsilon)}$ time, where $f(\varepsilon)$ is a small positive function of ε and $\varepsilon > 0$ is a small constant.

1.2.2 Lower Bounds for Sublinear Time Maximum Matching

With all the recent advances on the algorithmic side of the sublinear matching problem, a natural question arises: what are the fundamental limitations of these algorithms? One can explore various questions regarding the trade-off between approximation ratio and running time. The focus to answer the following question:

Question 2. *Is it possible to achieve a $(1 - \varepsilon)$-approximation in $n^{1.999}$ time?*

More specifically, is it possible to approximate the maximum matching size to within a small constant $\varepsilon > 0$ while the running time does not get close to quadratic time. Along the way, we derive several lower bounds for different trade-offs and in various regimes, including the bounded-degree graph setting.

The lower bound side of sublinear matching is much less understood compared to the algorithmic side. The only known lower bound before our series of work is that $\Omega(n)$ time is needed for obtaining any constant approximation of maximum matching, which was proved two decades ago by Parnas and Ron [159]. While this lower bound was (nearly) matched by [34] for 1/2-approximations, it is not known whether it is optimal or can be improved for better approximations. In particular, the current state of affairs leave it possible to obtain even a $(1-\varepsilon)$-approximation, for any fixed $\varepsilon > 0$, in just $O(n)$ time. Not only such a result would be amazing on its own, but as we will later discuss, it will have deep consequences in the study of dynamic graphs. It is also worth noting that in their beautiful work, Yoshida, Yamamoto, and Ito [176] showed existence of an $O(n) + \Delta^{O(1/\varepsilon^2)}$ time algorithm that obtains a $(1-\varepsilon)$-approximation[1]. While this is not a sublinear time algorithm for the full range of Δ, it runs in $O(n)$ time for $\Delta = n^{O(\varepsilon^2)}$. This shows that any potential $\omega(n)$ time lower bound must be proved on graphs of large degree.

We note that the essence of the $\Omega(n)$ lower bound of Parnas and Ron [159] is that $o(n)$ queries are not enough to even see all neighbors of a single vertex. Using this, [159] constructs an input distribution where no $o(n)$ time algorithm can see any edge of any (approximately) optimal matching. Indeed one key challenge that any super linear lower bound needs to overcome is to show that even though there are, say, $O(n^{1.1})$ time algorithms that "see" as many as $n^{\Omega(1)}$ edges of an optimal matching, they are still unable to obtain a a good approximation.

In Section 4.2, we present the first superlinear (in n) lower bound for the sublinear-time matching problem.

Result 5. *For any fixed $\alpha > 0$, any (possibly randomized) algorithm obtaining a $(2/3+\alpha)$-approximation of the size of maximum needs to make at least $n^{1.2-o(1)}$ adjacency list queries to the graph. This holds even if the graph is bipartite and has a matching of size $\Theta(n)$.*

Bounded Degree Graphs: The (approximate) maximum matching problem has been studied extensively both in the literature of LCAs [165, 9, 144, 132, 97, 58] and the closely related model of sublinear time algorithms [159, 154, 176, 34, 68, 43, 43] for bounded degree graphs where the maximum degree Δ is constant. There has been a sequence of improvements on LCAs [162, 144, 101, 97, 144, 132, 58]. The best-known algorithm for a $(1, \varepsilon n)$-approximate maximum matching is due to Levi, Rubinfeld, and Yodpinyanee [144] which adapts the elegant sublinear time algorithm of Yoshida, Yamamoto, and Ito [176] to the LCA model, achieving a running time of $(\Delta/\varepsilon)^{O(1/\varepsilon^2)} \operatorname{poly}\log(n)$ per query. Note that the LCA of [144, 176] runs in time $\operatorname{poly}(\Delta, \log n)$ whenever ε is constant. Thus it runs efficiently even in the case of "graphs of non-constant degree" [144]. If instead of a $(1, \varepsilon n)$-approximation we desire a $(1/2, 0)$ approximation, then this can be done in $O(\Delta \operatorname{poly}\log n)$ time [34].[2]

The only LCA lower bound in the literature for maximum matching, due to Parnas and Ron [159] from 2007, proves that any LCA computing a constant approximation of maximum matching needs to spend $\Omega(\Delta)$ time. The essence of the lower bound of [159] is a construction, where each vertex has degree $\Theta(\Delta)$ and has only one "important" edge that has to be in any constant approximate matching. Thus, any LCA that reports a constant approximate matching must scan a constant fraction of neighbors of the vertex being queried to find this important edge, implying the claimed $\Omega(\Delta)$ lower bound. Note that this approach cannot

[1] More precisely, [176] give a multiplicative-additive $(1-\varepsilon, o(n))$ approximation in $\Delta^{O_\varepsilon(1)}$ time. The claimed bound follows by slightly tweaking their algorithm using techniques developed in [34] for multiplicative approximations.

[2] We note that the LCA model is not directly studied in [34], but the abovementioned bound follows as a corollary of [34].

possibly result in an $\omega(\Delta)$ lower bound. Our goal is to strengthen this lower bound and narrow the gap between the algorithmic upper bound and the lower bound.

In Section 4.3, we establish a lower bound in the bounded-degree regime, resolving a long-standing open question that had remained open for more than a decade (see, in particular, Problem 39 on sublinear.info[3]). We provide a negative resolution to this question by showing that a running time of $\Delta^{\Omega(1/\varepsilon)}$ is necessary.

Result 6. *Let $\varepsilon \leq 0.01$. For any choice of $\log^4 n \leq \Delta \leq n^{\varepsilon}$, there is an n-vertex bipartite graph G of maximum degree Δ such that any randomized algorithm that with probability at least 0.51 provides a $(1, \varepsilon n)$-approximation for the size of the maximum matching in G must make $\Delta^{\Omega(1/\varepsilon)}$ adjacency list queries.*

Our construction also has implications in the LCA model. We prove a new lower bound on the complexity of LCAs for $(1, \varepsilon n)$-approximate matchings. We show that:

Result 7. *Let $\varepsilon \leq 0.01$. For any choice of $\log^4 n \leq \Delta \leq n^{\varepsilon}$, there is an n-vertex bipartite graph G of maximum degree Δ such that any LCA that with probability at least 0.51 computes a $(1, \varepsilon n)$-approximate maximum matching of G must make at least $\Delta^{\Omega(1/\varepsilon)}$ queries to G.*

The construction presented in the result mentioned above for sparse graphs can be adapted to yield a lower bound for dense graphs. While we do not formally prove this result, with a slight modification, the construction can be turned into an input distribution showing that any algorithm estimating the size of the maximum matching with additive error εn must use at least $\Omega(n\sqrt{n})$ time. However, there is a fundamental barrier to extending this lower bound beyond $n\sqrt{n}$ which we discussed in more detailed in Section 4.4.1. This leaves open the possibility of an algorithm running in time as small as $n\sqrt{n} \cdot \text{poly}(1/\varepsilon)$ and achieving a $(1, \varepsilon n)$-approximation. Whether such extremely fast algorithms exist remains an open question.

In Section 4.4, we close this huge gap by showing that the algorithm of Bhattacharya, Kiss, and Saranurak [57] is close to optimal. That is, we present a new lower bound that shows near-quadratic in n time is necessary in order to achieve a $(1, \varepsilon n)$-approximation of the maximum matching size.

Result 8. *For any $\delta > 0$ there is an $\varepsilon = \varepsilon(\delta) > 0$ such that any (randomized) algorithm that (with probability at least 2/3) estimates the size of maximum matching of an n-vertex graph up to an additive error of εn has to make $\Omega(n^{2-\delta})$ adjacency list queries to the graph.*

Finally, in Section 4.5, we show how to prove a lower bound for adjacency matrix access model. For dense graphs, a natural way to access the graph is via *adjacency matrix* queries—commonly referred to as the *dense graph model*. In this model, an algorithm can (adaptively) query whether there exists an edge between any pair of vertices of its choice. The goal is to solve a graph problem with as few queries as possible. For such an access model, our focus is particularly on a fundamental question:

Question 3. *How many adjacency matrix queries are required to estimate the size of a maximum matching in an n-vertex graph G?*

[3] `https://sublinear.info/index.php?title=Open_Problems:39`

Since none of the lower bounds from the previous results apply to the adjacency matrix model, it is important to close the large gap between the known lower and upper bounds in this setting. More specifically, the only known lower bound for this model is the folklore bound of $\Omega(n)$ for any multiplicative-additive approximation. This follows from considering an input graph that is either a random perfect matching or an empty graph. Distinguishing between these two cases requires identifying one of the $\Theta(n)$ adjacent pairs among the $\Theta(n^2)$ possible pairs, which clearly demands $\Omega(n)$ queries even with randomization. This situation leaves a substantial gap between the lower and upper bounds for additive εn errors: while the lower bound is only linear (i.e., $\Omega(n)$), the best known upper bound is barely subquadratic (i.e., $n^{2-\Omega_\varepsilon(1)}$) [57]. Our work focuses on this gap and closes it completely. Specifically, we show that the algorithm of [57] is (essentially) optimal.

Result 9. *For every $\delta > 0$ there exists $\varepsilon > 0$ (i.e., ε is only a function of δ) such that any algorithm (possibly randomized) that estimates (with probability at least 2/3) the size of maximum matching up to an additive error of εn must make at least $\Omega(n^{2-\delta})$ queries to the adjacency matrix of the graph.*

Connection to the adjacency list model: Let us now discuss the *adjacency list* query access model for which we have a much better understanding of lower bounds for estimating the size of the maximum matching. In this model, instead of specifying two vertices, each query of the algorithm specifies a vertex v and a number i; the response is then the ID of the i-th neighbor of v (or $\perp$ if v has less than i neighbors). In the previous results, we established strong lower bounds for estimating the maximum matching size in the adjacency list model.

Unfortunately this progress in the adjacency list model does not lead to any non-trivial (i.e., super-linear in n) lower bounds in the adjacency matrix model. Specifically, *proving adjacency matrix lower bounds appears to be much more difficult than for adjacency list* for two reasons:

(i) **The ability to query vertex-induced subgraphs:** An algorithm in the adjacency matrix model can select a small subset $U \subseteq V$ and learn the entire induced subgraph $G[U]$ with just $O(|U|^2)$ queries. Not only does this ability prevent a straightforward extension of adjacency list lower bounds to adjacency matrix ones, but in fact has led to formal separations between the two models for some problems. For example, a $(\Delta+1)$ vertex coloring in dense graphs with $\Delta = \Theta(n)$ can be found with $\widetilde{O}(n^2/\Delta) = \widetilde{O}(n)$ adjacency matrix queries but requires $\Omega(n\Delta) = \Omega(n^2)$ adjacency list queries [17]. There are also many property testing problems that can be solved with constant adjacency matrix queries by selecting a constant size U at random and querying all the pairs in U, see e.g. [8, 108].

(ii) **Discovering non-edges:** Another key difference between the two models is that each query in the adjacency list model discovers an edge, whereas adjacency matrix queries reveal both edges and non-edges in the graph. Therefore, any analysis would have to show that non-edges do not reveal too much information about the hard instance. Indeed, as we will discuss extensively in Section 4.5.1, this is the main obstacle that we overcome in proving this result.

Other implications of our result: There are many natural scenarios for computing matchings where the access to the underlying graph is through adjacency matrix queries. Here we briefly mention some of these scenarios and the implications of our result:

- **Estimating the earth mover's distance:** The earth mover's distance (EMD) is the most natural measure of similarity between two distributions defined over the elements of a distance space (often a metric). Beretta and Rubinstein [45] showed that if the distributions are defined over n elements, then there is an algorithm obtains an additive ε-approximation of EMD with $n^{2-\Omega_\varepsilon(1)}$ queries to the distance metric, i.e., with truly subquadratic queries.

  The algorithm of [45] does not make use of the common metric assumption and works for arbitrary distance. Whether this barely subquadratic time algorithm can be improved to say $O_\varepsilon(n^{1.9})$ remained open even in the case of metric spaces (see also the related paper of Andoni and Zhang [10]). Our lower bound strongly rules out this possibility, even in the case of $(1, 2)$-metrics. In order to obtain this result, it is important that our lower bound works in the adjacency matrix model (as opposed to adjacency lists) as it naturally captures pairwise distance queries.

- **Dynamic algorithms:** In a recent line of work [32, 60, 57, 37, 18, 26] sublinear time algorithms for maximum matching have been utilized to obtain significant improvements for the maximum matching problem in the dynamic setting. The role that sublinear time algorithms play in the works of [32, 60, 57] is very different from that of [37, 18]. But, curiously, all of these results rely on adjacency matrix queries as opposed to adjacency matrix queries. Our lower bound is therefore of interest to the dynamic community. Concretely, our result implies that an update-time of barely sublinear in n (i.e., the bound achieved by [57]) is best one can hope for with a black-box application sublinear time algorithms.

1.2.3 Maximum Matching in Dynamic and Streaming Model

We study the maximum matching problem in the fully dynamic setting. Given a graph G which undergoes both edge insertion and edge deletion updates, the goal is to maintain a large matching while spending a small time per update. Denoting the number of vertices by n, the holy grail in dynamic graphs is to achieve algorithms with $\text{poly}(\log n)$ update-time, as this would be polynomial in the size of each update (which can be represented with $\Theta(\log n)$ bits). Unfortunately, known conditional hardness results rule out any $O(n^{1-\varepsilon})$ update-time algorithm for maintaining an *exact* maximum matching [1, 77, 122]. As such, much of the focus in the literature has been on *approximate* maximum matchings [11, 38, 39, 51, 50, 48, 56, 55, 54, 53, 64, 109, 117, 153, 157, 43, 32, 60].

For over a decade, we have had algorithms maintaining a greedy maximal matching, and thus a 1/2-approximation of maximum matching, in $\text{poly}(\log n)$ time per update [29, 170, 31]. At the expense of using a larger polynomial in n update time, it is known that the approximation can be improved using various *matching sparsifiers* developed in the literature [117, 50, 38]. However, the problem of maintaining a better-than-1/2-approximation in $\text{poly}(\log n)$ time had remained open until last year where for a small $\varepsilon_0 > 0$, the concurrent works of Behnezhad [32] ($\varepsilon_0 \sim 0.001$) and Bhattacharya, Kiss, Saranurak, and Wajc [60] ($\varepsilon_0 \sim 0.006$) achieved a $(1/2 + \varepsilon_0)$-approximation provided that the goal is to maintain just the size (and not the edge-set) of the matching. This state of affairs leaves two major open problems:

- *Can we also maintain the edges of a* $1/2 + \Omega(1)$ *approximate matching in* $\text{poly}(\log n)$ *time?*
- *What is the best approximation of maximum matching size achievable in* $\text{poly}(\log n)$ *time?*

Our focus in this work is on the latter question.

A gap between bipartite and general graphs: The algorithms of [32, 60] have two phases. In the first phase, they maintain a maximal matching explicitly using the fast algorithms of [29, 170, 31]. In the second phase, they augment this maximal matching by employing the sublinear time matching size estimator of [34]. Although this leads to only slightly better than 1/2-approximation in general graphs, it is shown in [32, 60] that it leads to a much better (almost) $(2-\sqrt{2}\sim 0.585)$-approximation if the input graph is bipartite.

Such two-phase algorithms have also long been studied in the context of two-pass streaming algorithms [140, 130, 86, 139, 141] for which a similar gap between general and bipartite graphs has persisted. In particular, the state-of-the-art two-pass semi-streaming algorithm for bipartite graphs, by Konrad [139] from 2018, achieves the same (almost) $(2-\sqrt{2}\sim 0.585)$-approximation. However, despite attempts [89, 130] the best approximation for general graphs is 0.538 [89].

We close the aforementioned gap between bipartite and general graphs in both models. For dynamic graphs, we prove that the approximation can be improved to (almost) $2-\sqrt{2}\sim 0.585$, matching what was known for bipartite graphs and significantly improving the previous 0.506 and 0.501-approximations of [60, 32] for general graphs.

Result 10. *For any fixed $\varepsilon > 0$, there is an algorithm that maintains a $(2-\sqrt{2}-\varepsilon)\sim 0.585$-approximation of the size of the maximum matching in* $\text{poly}(\log n)$ *worst-case update time even against adaptive adversaries.*

In the two-pass semi-streaming model, we show that the same (almost) $(2-\sqrt{2}\sim 0.585)$ approximation can be achieved for general graphs as well, matching what was known for bipartite graphs [139] and significantly improving prior 0.531 and 0.538 approximations of [130, 89] for general graphs. We emphasize that our streaming algorithm does not just estimate the size of the maximum matching, but rather returns the edges of the matching as well. Additionally, unlike our dynamic algorithm, our streaming algorithm is deterministic.

Result 11. *For any fixed $\varepsilon > 0$, there is a deterministic two-pass streaming algorithm that finds (the edges of) a $(2-\sqrt{2}-\varepsilon)\sim 0.585$-approximate maximum matching using $O(n\log n)$ space.*

Going beyond $(2-\sqrt{2})$-approximations: The $(2-\sqrt{2})$-approximation turns out to be a barrier in several settings, even for bipartite graphs. For instance, a work of Huang, Peng, Tang, Tao, Wu, and Zhang [124] establishes that no online matching algorithm under edge-arrivals (even allowing preemptions) can surpass $(2-\sqrt{2})$-approximations. While this is a different model than the ones considered in this work, it is in fact closely related to the streaming setting. See, in particular, the paper of Kapralov [131, Section 1] who points out that his techniques "can probably be extended" to the construction of Huang, Peng, Tang, Tao, Wu, and Zhang [124], ruling out *single-pass* semi-streaming algorithms achieving better than $(2-\sqrt{2})$-approximations. We also refer interested readers to the paper of Konrad and Naidu [141] which is more specifically focused on *two-pass* streaming algorithms and includes a discussion on beating $(2-\sqrt{2})$-approximations. In particular, they show this bound is tight for a certain class of algorithms and argue that "new techniques are needed in order to improve on the $(2-\sqrt{2})$ approximation factor". Given this current landscape, we believe it is an important open question for future research to either go beyond $(2-\sqrt{2})$-approximations in the fully dynamic model or the two-pass streaming model, or alternatively, prove its impossibility.

1.2.4 Traveling Salesman Problem and Path Cover in Sublinear Time Model

The traveling salesman problem (TSP) is a central problem in combinatorial optimization. Given a set V of n vertices and their pairwise distances, it asks for a Hamiltonian cycle of the minimum cost. We study sublinear time algorithms for TSP. The algorithm is given query access to the distance pairs, and the goal is to estimate the solution cost in time sublinear in the input size (which is $\Theta(n^2)$).

TSP is NP-hard to approximate within a polynomial factor for an arbitrary distance function. As such, much of the work in the literature has been on more specific distance functions. Some notable examples include *graphic TSP* [103, 151, 152, 168, 68] where the distances are the shortest paths over an arbitrary unweighted undirected graph, $(1,2)$-*TSP* [2, 68, 46, 134, 150] where the distances are 1 or 2, and more generally *metric TSP* [133, 75, 73, 169] where the distances satsify triangle inequality.

In 2003, Czumaj and Sohler [76, 75] showed that for any fixed $\varepsilon > 0$, a $(1+\varepsilon)$-approximation of the cost of metric minimum spanning tree (MST) and thus a $(2+\varepsilon)$-approximation of the cost of metric TSP can be found in $\widetilde{O}(n)$ time. Twenty years later, it still remains a major open problem to either break two-approximation in $n^{2-\Omega(1)}$ time or prove a lower bound.[4] However, better bounds are known for both graphic TSP and $(1,2)$-TSP. We present improved algorithms for these two well-studied variants of TSP. Our main tool to achieve this is an improved algorithm for the closely related *maximum path cover* problem which might be of independent interest.

Maximum Path Cover: The maximum path cover in a graph is a collection of vertex disjoint paths with the maximum number of edges in it. The (almost) 1/2-approximate maximum matching size estimator of Behnezhad [34] immediately implies an (almost) 1/4-approximation for the maximum path cover problem in $\widetilde{O}(n)$ time.[5] This can be improved to an (almost) $(3/8 = .375)$-approximation using the *matching-pair* idea of Chen, Kannan, and Khanna [68] in $\widetilde{O}(n\sqrt{n})$-time.[6] Our first main contribution is an improvement over both of these results:

Result 12. *For any $\varepsilon > 0$, there is a randomized algorithm that w.h.p. $(1/2-\varepsilon)$-approximates the size of maximum path cover in $\widetilde{O}(n \cdot \text{poly}(1/\varepsilon))$ time.*

Besides quantitavely improving prior work both in the running time and the approximation ratio, this results reaches a qualitatively important milestone as well. First, the running time is information-theoretically optimal up to poly $\log n$ factors (the lower bound holds for any constant approximation — see Section 5.2.9). Second, its approximation ratio hits a rather important barrier. We give a non-trivial reduction that shows a $(1/2+\Omega(1))$-approximation in $\widetilde{O}(n)$ time for maximum path cover would imply the same bound for maximum matching in bipartite graphs. Such a result has remained elusive for matching, which is one of the most extensively studied problems in the literature of sublinear time algorithms. See Section 5.2.9.

It is also worth noting that in bounding the running time of our algorithm, we use connections to parallel algorithms. Such a connection was previously only used for matchings [34].

[4] See e.g. Open Problem 71 on sublinear.info [115].

[5] The application of sublinear time maximum matching algorithms for approximating maximum path cover was first proposed by Gupta and Onak. See [115].

[6] We note that even though a subsequent result of Behnezhad [34] improved the running time for maximal matchings and graphic TSP from $O(n\sqrt{n})$ in [68] to $\widetilde{O}(n)$, it is not immediately clear whether the same holds for path cover and $(1,2)$-TSP as they rely on a different notion of a matching pair.

Running Time	Approximation Ratio	Metric	Reference
$\widetilde{O}(n)$	$1.75+\varepsilon$	(1,2)	Folklore
$\widetilde{O}(n\sqrt{n})$	$1.625+\varepsilon$	(1,2)	Chen, Kannan, and Khanna [68]
$\widetilde{O}(n)$	$1.5+\varepsilon$	(1,2)	**This work**
$\widetilde{O}(n)$	1.929	Graphic	Chen, Kannan, and Khanna [68]
$\widetilde{O}(n)$	1.834	Graphic	**This work**
$n^{2-\Omega(1)}$	1.667	Graphic	**This work**
$\Omega(n^2)$	$1+\varepsilon$	(1,2) & Graphic	Chen, Kannan, and Khanna [68]
$n^{1+\Omega(1)}$ (Conditional)	$1.5-\varepsilon$	(1,2) & Graphic	**This work**

Table 1.1: Comparison of running time and approximation ratio of our TSP algorithms and lower bounds with prior work.

$(1,2)$-TSP: The $(1,2)$-TSP problem has been studied extensively in the classical setting. In his landmark paper, Karp [134] showed that $(1,2)$-TSP is NP-hard. Papadimitriou and Yannakakis [158] then proved its APX-hardness. Since then there has been a significant amount of work on $(1,2)$-TSP in the classical setting. The current best known inapproximability bound for $(1,2)$-TSP is 535/534 [135]. After a series of works, the best known polynomial time approximation is 8/7 [46] which can be implemented in $O(n^3)$ time [2]. For sublinear time algorithms, an $\widetilde{O}(n)$-time (almost) 1.75-approximation is folklore [115]. Chen, Kannan, and Khanna [68] improved the approximation to (almost) 1.625 in $\widetilde{O}(n\sqrt{n})$ time.

It is not hard to see that up to a small additive error of 1, $(1,2)$-TSP is equivalent to finding a maximum path cover on the weight-1 edges and then connecting their endpoints via weight-2 edges. A simple calculation shows that any α-approximation for the maximum path cover problem leads to a $(2-\alpha)$-approximation for $(1,2)$-TSP. Our path cover algorithm immediately implies the following result as a corollary:

Result 13. *For any $\varepsilon > 0$, there is a randomized algorithm that w.h.p. $(1.5+\varepsilon)$-approximates the cost of $(1,2)$-TSP in $\widetilde{O}(n \cdot \mathrm{poly}(1/\varepsilon))$ time.*

Similarly, the running time of our algorithm is information-theoretically optimal up to $\mathrm{poly}\log n$ factors, and its approximation ratio hits a natural barrier due to a connection to sublinear time matching that we establish in this work.

Graphic TSP: The graphic TSP problem is equivalent to finding a tour of the minimium size that visits all the vertices. This is an important instance of TSP that has received a lot of attention over the years. For polynomial time algorithms, a 1.5-approximation of Christofides [73] (which also works more generally for metric TSP) had remained the best known until a series of works over the last decade improved it to $(1.5-\varepsilon_0)$

[103], 1.461 [151], 1.444 [152], and finally to 1.4 [168]. For sublinear time algorithms, Chen, Kannan, and Khanna [68] showed that an (almost) $(27/14 \approx 1.928)$-approximation of graphic TSP can be obtained in $\widetilde{O}(n\sqrt{n})$ time. The running time was subsequently improved to $\widetilde{O}(n)$ by Behnezhad [34].

We first show that plugging the path cover algorithm into the framework of [68] immediately improves their approximation from 1.928 to (almost) 1.9 while keeping the running time $\widetilde{O}(n)$. We then give a more fine tuned algorithm that obtains a much improved approximation ratio of $(11/6 \approx 1.833)$.

Result 14. *For any $\varepsilon > 0$, there is a randomized algorithm that w.h.p. $(1+\varepsilon)(\frac{11}{6} \approx 1.833)$-approximates the cost of graphic TSP in $\widetilde{O}(n \cdot \text{poly}(1/\varepsilon))$ time.*

Over the past few years, significant advancements have been made in the development of sublinear matching algorithms as discussed in this thesis. The result of [57] have led to the creation of a $(1, \varepsilon n)$-approximation algorithm for maximum matching, with running time of $n^{2-\Omega_\varepsilon(n)}$. Leveraging these sublinear algorithms, we have devised a slightly subquadratic algorithm that provides a more accurate estimation of the size of graphic TSP.

Result 15. *For any $\varepsilon > 0$, there is a randomized algorithm that w.h.p. $(1+\varepsilon)(\frac{5}{3} \approx 1.666)$-approximates the cost of graphic TSP in $n^{2-\Omega_\varepsilon(1)}$ time.*

We contrast our results with prior sublinear TSP algorithms in Table 1.1.

Further related work: Finally, we note that in a recent paper, Chen, Khanna, and Tan [70] show that assuming that the metric has a spanning tree supported on weight 1 edges, one can obtain a $(2-\varepsilon_0)$-approximation with $\widetilde{O}(n\sqrt{n})$ queries for some small unspecified constant $\varepsilon_0 > 0$. While this is a more general metric than graphic TSP and (1,2)-TSP that we study in this thesis, we note that the two papers are orthogonal and their results are incomparable. In particular, the techniques developed in this thesis are specifically designed to improve the approximation to much below 2, whereas [70] focuses on generalizing the distance function while beating 2.

1.2.5 Sublinear Algorithm for Steiner Tree and Set Cover

In the Steiner tree problem, we are given an undirected graph $G = (V, E)$, where each edge e has an associated cost $w(e)$, and a specified set of terminal vertices $T \subseteq V$. Then, the objective is to find a minimum-cost subgraph H of G that connects all terminals in T. The Steiner tree problem is one of the most fundamental problems in combinatorial optimization and has been extensively studied by the TCS community since it was included among Karp's 21 NP-Complete problems [134]. The state-of-the-art approximation factor for the Steiner tree problem is $\ln 4 + \varepsilon < 1.39$ [63], and it is known that approximating it to a factor better than $96/95$ is NP-hard [72]. This Steiner tree problem has been studied in various domains, including approximation algorithms [163, 63], online algorithms [125, 23, 149, 114], stochastic algorithms [116, 112, 95], and massive data analysis models [65, 75, 69].

In the sublinear model, we are given access to the distance matrix of the graph. Let $\mathsf{ST}(V, T, w)$ denote the weight of a minimum-weight Steiner tree on instance (V, T, w). Then, the goal is to design an algorithm that estimates $\mathsf{ST}(V, T, w)$ using the fewest possible queries to the distance matrix.

Czumaj and Sohler [75] presented the first sublinear query algorithm for the metric Steiner tree problem, showing a $(2+\varepsilon)$-approximation using $\tilde{O}(k/\varepsilon^{O(1)})$ queries through their improved algorithm for the minimum spanning tree (MST) problem. Specifically, this follows their sublinear $(1+\varepsilon)$-approximation for MST together with the well-known result by Gilbert and Pollak [104] showing that an α-approximation for MST over the metric induced on the terminals T is a (2α)-approximation for the metric Steiner tree instance with T as the terminal set.

Recently, Chen, Khanna, and Tan [69] studied the design of sublinear algorithms with strictly better-than-2 approximation for the metric Steiner tree problem. On the lower bound side, they showed that for any $\varepsilon > 0$, estimating the Steiner tree cost to within a $(5/3-\varepsilon)$-factor requires $\Omega(n^2)$ queries, even when the number of terminals $|T|$ is constant. Moreover, they showed that for any $\varepsilon > 0$, estimating the Steiner tree cost to within a $(2-\varepsilon)$-factor requires $\Omega(n+|T|^{6/5})$ queries. Additionally, they proved that for any $0 < \varepsilon < 1/3$, any algorithm that outputs a $(2-\varepsilon)$-approximate Steiner tree (not just its cost) requires $\Omega(n|T|)$ queries. On the upper bound side, they showed that it is possible to achieve a better-than-2 estimate of the Steiner tree cost in sublinear time: there exists an algorithm that, with high probability, computes a $(2-\eta)$-approximation of the Steiner tree cost using $\widetilde{O}(n^{13/7})$ queries, where $\eta > 0$ is a universal constant. At the core of their sublinear algorithm for metric Steiner tree with improved approximation guarantee, they relate the problem of achieving a better-than-2 estimation for the Steiner tree to a variant of set cover problem with a different objective.

More specifically, given a universe of elements $\mathcal{U}$ and a collection $\mathcal{F}$ of subsets of $\mathcal{U}$, in the Threshold Set Cover problem the goal is to estimate $\mathsf{ThSC}(\mathcal{U},\mathcal{F}) \coloneqq |\mathcal{U}| - \mathsf{SC}(\mathcal{U},\mathcal{F})$, where $\mathsf{SC}(\mathcal{U},\mathcal{F})$ denotes the size of an optimal set cover solution for $(\mathcal{U},\mathcal{F})$, i.e., the minimal number of sets in $\mathcal{F}$ whose union equals $\mathcal{U}$.

Specifically, given access to the adjacency matrix of the graph representation of $(\mathcal{U},\mathcal{F})$, where there is an edge between $e \in \mathcal{U}$ and $S \in \mathcal{F}$ if and only if $e \in S$, Chen, Khanna, and Tan [69] designed an algorithm that, for any constant $0 < \varepsilon < 1$, with high probability, outputs a *multiplicative-additive* $(1/4, \varepsilon|\mathcal{U}|)$-approximation for estimation of $\mathsf{ThSC}(\mathcal{U},\mathcal{F})$ using $\widetilde{O}_\varepsilon(|\mathcal{F}|^{3/2} + |\mathcal{F}|^{3/4} \cdot |\mathcal{U}|)$ queries to the adjacency matrix (or, membership queries). An estimate SOL for Threshold Set Cover on $(\mathcal{U},\mathcal{F})$ is a *multiplicative-additive* (γ_1,γ_2)-approximation, if $\gamma_1 \cdot \mathsf{ThSC}(\mathcal{U},\mathcal{F}) - \gamma_2 \leq \mathrm{SOL} \leq \mathsf{ThSC}(\mathcal{U},\mathcal{F})$.

More broadly, there has been a large body of work on solving set cover problems in the massive data models of computation over the past decade [166, 79, 119, 83, 21, 126, 14, 30, 127, 111]. In particular the work of [127, 111] consider the set cover problem in the sublinear query model. However their algorithms assumes that it has an access to the adjacency list model as opposed to the adjacency matrix model, and thus cannot be directly employed here.

Our key contribution is an algorithm for Threshold Set Cover, offering improved approximation guarantees and query complexity, as detailed below:

Result 16. *There exists an algorithm that, given a set system $(\mathcal{U},\mathcal{F})$ with oracle access to its adjacency matrix (also known as membership queries), outputs a multiplicative-additive $(1/2, \varepsilon \cdot |\mathcal{U}|)$-approximation to Threshold Set Cover, in $\widetilde{O}(|\mathcal{F}|^{5/3})$ time, with high probability.*

Note that both the query complexity and the running time of the algorithm are bounded by $\widetilde{O}(|\mathcal{F}|^{5/3})$, improving upon the algorithm by Chen, Khanna, and Tan [69] for large values of $|\mathcal{U}|$, which uses $\widetilde{O}_\varepsilon(|\mathcal{F}|^{3/2} + |\mathcal{F}|^{3/4} \cdot |\mathcal{U}|)$ membership queries and provides a multiplicative-additive $(1/4, \varepsilon|\mathcal{U}|)$-approximation for the

problem. Notably, when $|\mathcal{U}| = \omega(|\mathcal{F}|^{2/3})$, the algorithm becomes sublinear in $|\mathcal{U}| \cdot |\mathcal{F}|$, making it especially relevant for applications in the metric Steiner tree problem. More specifically, our new algorithm for Threshold Set Cover results in the following improved sublinear query algorithm for the metric Steiner tree problem, which we show in Section 5.3.4.

Result 17. *There exists an algorithm that, given an instance of metric Steiner tree denoted by (V, T, w) with oracle access to the distance matrix of (V, w), outputs a $(2-\eta)$-estimate of $\mathsf{ST}(V, T, w)$ using $\widetilde{O}(n^{5/3})$ queries to distance matrix, where $\eta > 0$ is a universal constant, with high probability.*

Notably, the query complexity of our algorithm improves upon the $\widetilde{O}(n^{13/7})$ query complexity of the algorithm of Chen, Khanna, and Tan [69].

1.2.6 Sublinear Algorithm for Steiner Forest and Maximal Independent Set

Steiner Forest is a classic network design problem where, given a weighted undirected graph $G = (V, E, w)$ where the weights are specified by $w : E \to \mathbb{R}_{\geq 0}$, along with a set of k source-sink terminal pairs $T = \{(s_1, t_1), \ldots, (s_k, t_k)\}$, the goal is to find a subgraph G' of minimum total weight such that each pair (s_i, t_i) is connected in G'. This problem generalizes the *Steiner Tree* problem (in fact it is also known as *generalized Steiner tree problem*), and hence is APX-hard. In particular, approximating it to a factor better than 96/95 is NP-hard [72].

The first approximation algorithm for the problem was a primal-dual based approach given by Agrawal, Klein, and Ravi [3] that achieved a 2-approximation. Later, Goemans and Williamson [105] provided a simplified simulation of their primal-dual algorithm, which gives a $(2 - 2/n)$-approximate solution, where n is the number of vertices. Improving the approximation guarantee of 2 has remained an open problem until very recently, when [4] broke this barrier by designing a $2 - 2^{-11}$-approximation. Furthermore, Gupta and Kumar [113] provided a simple greedy-based algorithm achieving a constant larger than 2. Designing algorithms with specific features and addressing the problem within various models such as online, parallel, and streaming remains an active area to day (see e.g. [110, 74, 128, 67, 28, 102]).

There is a large body of work on designing sublinear time algorithms for graph problems such as minimum spanning tree (MST) [65], maximal independent set (MIS) [176], matching [176, 132, 34, 42, 43, 57], spanners [143, 160, 12], metric MST [75], metric Steiner Tree [69, 148], and metric TSP [68, 70, 44], among others. Given that a sublinear time algorithm cannot afford to read the entire graph, it is instead provided an oracle to the input graph. There are two main oracle models for graph problems considered in the literature. In the adjacency list oracle, the algorithm can query (v, i), where $v \in V$ and $i \leq n$, and the oracle reports the i-th neighbor of the vertex v in its adjacency list (along with the weight of the edge in case of weighted graphs), or NULL if i is larger than the number of v's neighbors. In the adjacency matrix oracle, the algorithm can query (u, v), where $u, v \in V$, and the oracle reports whether there exists an edge between u and v (along with its weight in the case of weighted graphs). Let $\mathsf{SF}(V, T, w)$ denote the minimum weight of a Steiner Forest on instance (V, T, w). Then, the goal is to design an algorithm that estimates $\mathsf{SF}(V, T, w)$ using $o(n^2)$ queries to the distance matrix via the oracle.

The focus of this work is on designing a sublinear time algorithm for the metric Steiner Forest problem under the adjacency matrix model. We remark that for "non-metric" instances, even approximating the MST cost within any factor requires $\Omega(n^2)$ time. Consider two vertex sets S and T, each of size $n/2$, where

all pairwise distances between vertices within S or within T are zero, and all distances between S and T are one, except for a single random pair (s^*, t^*) across S and T, whose distance is zero with probability $1/2$. Any algorithm that approximates the MST cost with high probability must distinguish between a total cost of zero and one. This requires identifying the pair (s^*, t^*), which in turn requires $\Omega(n^2)$ queries.

We give the first sublinear time algorithm for approximating $\mathsf{SF}(V, T, w)$.

Result 18. *There exists an algorithm for estimating the cost of Steiner Forest within a multiplicative factor of $O(\log k)$ using $O(T_{\text{MIS}} \cdot \log k) = \widetilde{O}(n^{3/2})$ queries to the distance matrix oracle. Here, T_{MIS} denotes the best runtime of a sublinear algorithm for finding a multiplicative $O(1)$-approximation for the size of any MIS under the adjacency matrix model.*

We assume that the number k of terminal pairs is $O(n)$. Note that for any given Steiner Forest instance, there exists a list of at most $n-1$ terminal pairs (s_i, t_i) that fully characterize it. Without this assumption, an $O(k)$ term will be added to the runtime to account for reading the pairs and computing this succinct representation. Moreover, this dependence on k in the runtime is required. To see this, consider the case where there is a single vertex v that is far away from the rest of the graph, and there only exists a single pair involving v. Unless this pair is detected, the reported solution will be off by an arbitrarily large factor.

Maximal Independent Set (MIS). As a key component of our approach, we develop a sublinear algorithm for the maximal independent set (MIS) problem in the adjacency matrix model.

An independent set is a set of vertices such that no two of its elements are connected by an edge. While the *maximum* independent set problem is NP-hard to even approximate within a factor $n^{1-\varepsilon}$ [121, 178], for our purposes we only require an independent set that is *maximal*.

MIS is a basic problem in its own right. It is also broadly used as a building block and studied in computational models with limited time, memory or communication, such as distributed and parallel [145, 7, 61, 96, 90, 100], Massively Parallel Computing [142, 62, 155, 98, 101, 99], Local Computation Algorithms [154, 176, 165, 9, 162, 144, 97], dynamic [31, 22, 66], as well as in algorithms for other fundamental problems, such as matching (a maximal matching is an MIS of the corresponding line graph) and vertex cover [176, 156, 34], or correlation clustering [5, 35, 78].

Since an MIS cannot be explicitly found in sublinear time[7], the task becomes that of estimating the MIS size, or developing oracles that efficiently check whether a vertex is in some fixed MIS. The study of such algorithms was initiated by Nguyen and Onak [154] in the context of bounded-degree graphs. They considered the **random greedy maximal independent set (RGMIS)** process, which iterates over vertices according to a random permutation π and adds a vertex to the set $\text{RGMIS}(\pi)$ if none of its predecessors were already added. A natural oracle to check whether a given vertex v is in $\text{RGMIS}(\pi)$ is to ask, for all its neighbors u with lower rank in π, whether u is in $\text{RGMIS}(\pi)$, and return yes only if none of them are.

Nguyen and Onak [154] showed that for a random permutation π, the expected query complexity of this oracle is $2^{O(\Delta)}$, where Δ is the maximum degree. (They then used this to obtain a $(2, \varepsilon n)$-approximation[8] for maximum matching in time $2^{O(\Delta)}/\varepsilon^2$.) They conjectured that if the neighbors u are queried in the order

[7] Consider, for example, an instance with only one edge connecting two random vertices. To find any MIS, one must locate that edge, which requires $\Omega(n^2)$ time in the adjacency matrix model.

[8] We use this notation to say that the algorithm has a multiplicative approximation ratio of 2, with an additional additive error of up to εn.

of increasing rank in π (see Algorithm 34), the query complexity bound can be improved.

In a seminal result, Yoshida, Yamamoto and Ito [176] showed that if the query vertex v is also random, this improved oracle indeed has an expected query complexity of only $O(\Delta)$ (see Theorem 5.4.13 for the precise statement). By identifying the low-rank neighbors of a vertex in time $O(\Delta)$ in the adjacency list model or in time $O(n)$ in the adjacency matrix model, this implies an algorithm for the vertex oracle that runs in time $O(\Delta^2)$ in the adjacency list model or in time $O(\Delta n)$ in the adjacency matrix model. We note that these are not sublinear-time if $\Delta = \Omega(n)$, and they only return the answer for a single vertex query. To the best of our knowledge, no faster algorithms for MIS itself have been developed. However, the result of Yoshida, Yamamoto and Ito [176] has been highly influential for other basic problems, particularly maximum matching. By studying line graphs, they obtained a $(2, \varepsilon n)$-approximation algorithm that runs in time $\text{poly}(\Delta)/\varepsilon^2$. This was further improved by Onak, Ron, Rosen and Rubinfeld [156] and Behnezhad [34].

We give a sublinear time algorithm to estimate the RGMIS size in the adjacency matrix model. We note that we obtain a purely multiplicative approximation, without an additive error; this is indeed required for our approximation guarantee for the Steiner Forest problem.

Result 19. *For any $\varepsilon \in (0, 1)$ there is an algorithm (Algorithm 37) that, given a graph (V, E) with oracle access to its adjacency matrix, with high probability reports a multiplicative $(1 + \varepsilon)$-approximation to the value $|\text{RGMIS}(\pi)|$ for some permutation π of V and runs in $\widetilde{O}(n^{3/2}/\varepsilon^2)$ time.*

Chapter 2

Preliminaries

2.1 Notations

Throughout this thesis, we use $G = (V(G), E(G))$ to denote the input graph, where $V(G)$ and $E(G)$ represent its vertex and edge sets, respectively. We let $n = |V(G)|$ and $m = |E(G)|$ denote the number of vertices and edges. We use $\deg_G(v)$ to denote the degree of a vertex v, i.e., the number of edges with one endpoint equal to v. We use $\Delta(G)$ to denote the maximum degree over all vertices in the graph, and $\bar{d}(G)$ to denote its average degree. When the graph under consideration is clear from the context, we may omit the argument and simply write V, E, $\deg(v)$, Δ, and $\bar{d}$. Given a subset of vertices $A \subseteq V$, $G[A]$ is defined to be the induced subgraph consisting of all edges with both endpoints in A. Further, given disjoint subsets $A, B \subset V$ of vertices, $G[A, B]$ is defined to be the bipartite subgraph of G consisting of all edges between A and B. For $E' \subseteq E$, we let $G[E']$ be the subgraph of G that is induced by edges E'.

We use $A \oplus B := (A \cup B) \setminus (A \cap B)$ to denote the symmetric difference of two sets A and B. Also for any positive integer k, we use $[k]$ to denote the set $\{1, \ldots, k\}$. As is common in the literature, we use the term "with high probability" to refer to a probability of at least $1 - n^{-\alpha}$, for a sufficiently large constant $\alpha \geq 2$. Moreover, we use $\widetilde{O}(\cdot)$, $\widetilde{\Theta}(\cdot)$, and $\widetilde{\Omega}(\cdot)$ to hide the dependency on $\text{poly}(\log n)$.

Given the problem of maximizing a function $f : D \to \mathbb{R}$ defined over a domain D, with optimal value f^*, an (α, β)-multiplicative-additive approximation of f^* is a solution $s \in D$ such that $(f^*/\alpha) - \beta \leq f(s) \leq f^*$. Similarly, for a minimization problem with objective function $f : D \to \mathbb{R}$ and optimal value f^*, an (α, β)-multiplicative-additive approximation is a solution $s \in D$ satisfying $f^* \leq f(s) \leq \alpha f^* + \beta$.

2.2 Graph Theory Tools

A *multigraph* is a type of graph in which multiple edges, also known as parallel edges, are allowed between any pair of vertices. A *line graph* of a graph G is a graph that represents the adjacencies between the edges of G. More formally, given a graph G, the line graph $L(G)$ is constructed as follows:

- **Vertices**: Each vertex in $L(G)$ corresponds to an edge in G.
- **Edges**: Two vertices in $L(G)$ are connected by an edge if and only if their corresponding edges in G share a common endpoint (i.e., they are incident to the same vertex in G).

We define a bipartite graph $H = (U, V, E)$ as *biregular* if the degree of all vertices in U is identical, and likewise, the degree of all vertices in V is identical. For a directed graph, we define its *underlying graph* as the undirected graph obtained by disregarding the direction of the edges.

A *bridge (cut edge)* in a graph is an edge whose deletion increases the number of connected components. Similarly, a *cut vertex* is a vertex whose deletion (along with its edges) increases the number of connected components. A *biconnected graph* is a connected graph with no cut vertex. Also, a *biconnected component (block)* of a graph is a maximal biconnected subgraph of the original graph. A non-trivial biconnected component is a block that is not a bridge. We say a graph is *2-edge-connected* if there is no bridge in the graph. A *2-edge-connected component* of a graph is maximal 2-edge-connected subgraph of the original graph. The *bridge-block tree* of a graph is a tree obtained by contracting the 2-edge-connected components; note that the edge set of a bridge-block tree correspond to the bridges in the original graph. Let G be a directed graph. Then, C is a strongly connected component of G if it is a maximal set of vertices such that there exists a directed path between any pair of vertices u and v in C, and vice versa.

Matching: A *matching* in a graph $G = (V, E)$ is a subset of edges $M \subseteq E$ such that no two edges in M share a common endpoint. A matching M is called a *maximum matching* if it has the largest possible size among all matchings in G. We use $\mu(G)$ to denote the size of maximum matching of G. Given a vector b of integer capacities of dimension $|V(G)|$, a *b-matching* in G is a *multi-set* F of edges in G such that each vertex $v \in V$ appears no more than b_v times as an endpoint of an edge in F.

Vertex Cover: Given a graph $G = (V, E)$, a *vertex cover* is a subset of vertices $C \subseteq V$ such that every edge in E has at least one endpoint in C. In other words, for every edge $(u, v) \in E$, at least one of u or v belongs to C. A *minimum vertex cover* is a vertex cover of smallest possible size, and we denote its size by $\nu(G)$. We use the following classic theorem by König's [138].

Proposition 2.2.1 (König's Theorem)**.** *For any bipartite graph G, it holds $\mu(G) = \nu(G)$.*

Augmenting/Alternating Paths: Given a matching M of G, a path in G is an *alternating path* for M if its edges alternatively belong to M. An alternating path is an *augmenting path* for M if the first and the last edges of the path do not belong to M.

It is well-known that if a maximal matching is nearly half the size of a maximum matching, then almost all of its edges belong to length-three augmenting paths. The following statement is folklore.

Claim 2.2.2 (Folklore)**.** *Let M be a maximal matching and $M^\star$ a maximum matching. Suppose $|M| < (\frac{1}{2} + \delta)|M^\star|$. In $M \oplus M^\star$, there are at least $|M| - 4\delta|M^\star|$ length-3 augmenting paths for M.*

Random Greedy Maximal Matching: Given an input graph $G = (V, E)$ and a permutation π over the edges of E, a greedy maximal matching can be obtained by sequentially iterating over the edges in E according to π and adding each edge to the maximal matching if none of its adjacent edges have already been added. We let $\text{GMM}(G, \pi)$ denote this maximal matching. A random greedy maximal matching, i.e., $\text{RGMM}(G)$ is the matching obtained by picking a permutation π uniformly at random and outputting $\text{GMM}(G, \pi)$.

We use the following proposition about the size of greedy matchings in vertex-subsampled subgraphs. It was first proved in [39] using the techniques developed in [140].

Proposition 2.2.3 ([39, Lemma 5.2])**.** *Let $G(V, U, E)$ be a bipartite graph, let π be an arbitrary permutation over E, let $p \in (0, 1)$, and let M be an arbitrary matching in G. 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* $\mathrm{GMM}([W \cup U], \pi)$*; then*

$$\mathbf{E}_W[X] \geq p(|M| - 2p|V|).$$

Proposition 2.2.4 ([34])**.** *There exists an algorithm that, given adjacency list access to a graph G of average degree d, for a random vertex v and a random permutation π, determines if v is matched in* $\mathrm{GMM}(G, \pi)$ *in $\widetilde{O}(d)$ expected time.*

Fractional Matching: Given a graph $G = (V, E)$, a fractional matching of G is a set of weights $x : E \to [0, 1]$ on the edges such that for any vertex u the following condition is satisfied:

$$\sum_{e \in \delta(u)} x_e \leq 1.$$

For a vertex set $S \subseteq V$, we define $x(S) = \sum_{e \in G[S]} x_e$.

The following proposition is an application of the blossom inequality if we relax the constraints to only consider subsets of vertices with a size of at most $1/\varepsilon$. We refer readers to [24] for the proof and to section 25.2 of [167] for a detailed discussion about blossom inequalities.

Proposition 2.2.5. *Let G be any graph, and let x be a fractional matching on G such that for every vertex set $S \subseteq V$ that $|S| < 1/\varepsilon$, we have*

$$\sum_{e \in G[S]} x_e \leq \left\lfloor \frac{|S|}{2} \right\rfloor.$$

Then, it holds that $\mu(G) \geq (1 - \varepsilon) \sum_e x_e$. We refer to the above inequality as the blossom inequality.

Maximal Independent Set (MIS): Given a graph $G = (V, E)$, an *independent set* is a subset of vertices $I \subseteq V$ such that no two vertices in I are adjacent. A *maximal independent set (MIS)* is an independent set I that cannot be extended by adding any vertex from $V \setminus I$ without creating an edge between two vertices in the set.

Random Greedy Maximal Independent Set: Given a graph $G = (V, E)$, a random greedy maximal independent set is constructed by first selecting a random permutation π of the vertices V. The algorithm then iterates over the vertices in the order specified by π, adding each vertex to the independent set if none of its neighbors are already in the set (i.e., it does not share an edge with any vertex already included in the independent set). This process continues until all vertices have been considered. The term "random" comes from the fact that the permutation π is chosen uniformly at random among all possible permutations of the vertices.

Parallel Randomized Greedy Maximal Independent Set: Let G be a graph, and let π be a permutation of its vertices. In each iteration, we select all vertices whose rank is lower than that of all their neighbors and then remove these vertices along with their neighbors from the graph. The number of iterations required for G to become empty is referred to as the round complexity, denoted by $\rho(G, \pi)$. It is not hard to see that

the MIS produced by the parallel randomized greedy maximal independent set is the same as the output of the random greedy maximal independent set for a fixed permutation π.

Bigraphic pairs of sequences: We use the following results on *bigraphic pairs of sequences* defined below.

Definition 2.2.6 (Bigraphic Pairs of Sequences)**.** *Let $a = (a_1, a_2, \dots a_n)$ and $b = (b_1, b_2, \dots, b_m)$ be two sequences of non-negative integers. We say this is a bigraphic pair of sequences if there exists a bipartite graph where a corresponds to the degree sequence of one part of the graph and b corresponds to the degree sequence of the other part.*

Proposition 2.2.7 (Gale–Ryser Theorem)**.** *Let $(a_1, a_2, \dots a_n)$ and $(b_1, b_2, \dots, b_m)$ be two sequences of non-negative integers such that $a_1 \geq a_2 \geq \dots \geq a_n$. Then, these two sequences are bigraphic if and only if $\sum_{i=1}^{n} a_i = \sum_{i=1}^{m} b_i$, and*

$$\sum_{i=1}^{r} a_i \leq \sum_{i=1}^{m} \min(b_i, r) \qquad \text{for all } 1 \leq r \leq n.$$

Erdős–Rényi Graph: An *Erdős–Rényi graph*, denoted by $G(n, p)$, is a random graph on n vertices in which each pair of vertices is connected independently with probability $p \in [0, 1]$. Equivalently, the number of edges follows a binomial distribution, and the presence or absence of each edge is independent of all other edges. A *bipartite Erdős–Rényi graph*, denoted by $G(n_1, n_2, p)$, is a random bipartite graph with vertex sets of size n_1 and n_2 in which each edge between the two parts appears independently with probability p.

Proposition 2.2.8 ([85, 84, 92])**.** *Let G be a bipartite Erdős–Rényi graph such that each part of the graph contains n vertices. If the probability of the existence of each edge is at least $(\log^2 n)/n$, then there exists a perfect matching in G with high probability.*

2.3 Probabilistic Tools

We use the following standard concentration inequalities.

Proposition 2.3.1 (Chernoff Bound)**.** *Let $X_1, X_2, \dots, X_n$ be independent Bernoulli random variables, and let $X = \sum_{i=1}^{n} X_i$. For any $t > 0$, $\Pr[|X - \mathbf{E}[X]| \geq t] \leq 2 \exp\left(-\frac{t^2}{3\,\mathbf{E}[X]}\right)$.*

Proposition 2.3.2 (Hoeffding's Inequality)**.** *Let $X_1, X_2, \dots, X_n$ be independent random variables such that $a \leq X_i \leq b$. Let $\overline{X} = (\sum_{i=1}^{n} X_i)/n$. For any $t > 0$, $\Pr[|\overline{X} - \mathbf{E}[\overline{X}]| \geq t] \leq 2 \exp\left(-\frac{2nt}{(b-a)^2}\right)$.*

Proposition 2.3.3 (Markov inequality)**.** *If X is a non-negative random variable, then for any $a > 0$, it holds that*

$$\Pr(X \geq a) \leq \frac{\mathbf{E}[X]}{a}.$$

Definition 2.3.4 (Negative Association [129, 136, 171])**.** *Let $X_1, X_2, \dots, X_n$ be a set of random variables. We say this set is negatively associated if for any two disjoint index sets $I, J \subseteq [n]$, and two functions f and g, both either monotonically increasing or monotonically decreasing, the following condition is satisfied:*

$$\mathbf{E}[f(X_i : i \in I) \cdot g(X_j : j \in J)] \leq \mathbf{E}[f(X_i : i \in I)] \cdot \mathbf{E}[g(X_j : j \in J)].$$

Proposition 2.3.5 (Chernoff Bound Negatively Associated Variables)**.** *Let $X_1, X_2, \ldots, X_n$ be a set of negatively associated Bernoulli random variables. Let $X = \sum_{i=1}^{n} X_i$. Then,*

$$\Pr\left[|X - \mathbf{E}[X]| \geq (1+\alpha)\,\mathbf{E}[X]\right] \leq \left(\frac{e^{\alpha}}{(1+\alpha)^{1+\alpha}}\right)^{\mathbf{E}[X]}.$$

Yao's Minimax Principle: We use the following theorem to prove the lower bound for randomized algorithms.

Proposition 2.3.6 (Yao's Lemma. Theorem 3 of [175])**.** *Let P be a problem over input domain X. For an input distribution $\mathcal{D}$ over X, let $B_{\mathcal{D},\lambda}$ be the set of deterministic algorithms that solve P with probability $1-\lambda$. Let $F_{1,\lambda}(P)$ be the distributional complexity of P with parameter λ:*

$$F_{1,\lambda}(P) := \sup_{\mathcal{D}} \inf_{a \in B_{\mathcal{D},\lambda}} c(a, \mathcal{D}),$$

where $c(a, D)$ denotes the average cost of a on $\mathcal{D}$. Let R_λ be the set of randomized algorithms that solve P with probability $1-\lambda$ for any input x. Let $F_{2,\lambda}(P)$ be the randomized complexity of P with parameter λ:

$$F_{2,\lambda}(P) := \inf_{A \in R_\lambda} \max_{x \in X} c(A, x),$$

where $c(A, x)$ is the average cost of A for input x. Then, for $0 \leq \lambda \leq \frac{1}{2}$, it holds

$$F_{2,\lambda}(P) \geq \frac{1}{2} F_{1,2\lambda}(P).$$

2.4 Problems Studied in This Thesis

Sublinear Maximum Matching: Given a graph G, represented in one of the following two ways, we study the problem of estimating the size of maximum matching:

- *Adjacency List:* In this model, the neighbors of each vertex are stored in a list sorted in an arbitrary order. Each query of the algorithm specifies a vertex v and an index i. The answer is the ID of the i-th vertex in the list of v's neighbors, or *empty* if v has less than i neighbors.
- *Adjacency Matrix:* In this model, each query of the algorithm specifies a pair of vertices u and v. The answer is 1 if u and v are adjacent, and 0 otherwise.

Since the size of the input graph can be as large as $\Omega(n^2)$, we are interested in designing algorithms that estimate the size of a maximum matching in sublinear time with respect to the input size, i.e., in $o(n^2)$ time.

Maintaining the Size of a Matching in the Fully Dynamic Model: We are given a graph on a fixed set of n vertices. Edges then can be inserted or deleted from the graph. We study dynamic algorithms for estimating the size of a matching. That is, the algorithm has to return an estimate $\widetilde{\mu}$ for the size of the maximum matching of G after each update. We say the adversary—which issues the updates—is *oblivious* if he does not change the sequence of updates based on the algorithm's previous output. On the other hand, an *adaptive* adversary may choose the sequence of updates adaptively based on the algorithm's outputs.

Sublinear Metric Traveling Salesman Problem (TSP): Given a set V of n vertices and their pairwise distances, the *metric TSP* asks for a Hamiltonian cycle of minimum cost. In the sublinear model, the algorithm is given query access to the distance pairs, which satisfy the metric constraint (i.e., the distances obey the triangle inequality), and the goal is to estimate the cost of an optimal solution in time sublinear in the input size, which is $\Theta(n^2)$. We use $\tau(V)$ to denote the size of the optimal TSP.

A *graphic TSP* is a special case of the metric TSP where the distances correspond to the shortest-path distances in an unweighted graph. Similarly, in a $(1,2)$-*TSP*, the distance between any pair of vertices is either 1 or 2, and the goal is again to find a Hamiltonian cycle of minimum total cost.

Sublinear Metric Steiner Tree: Given a set V of n vertices and their pairwise distances, a set of terminal points $T \subseteq V$, and query access to the distance matrix of a metric space over V, the *metric Steiner tree problem* asks for a minimum-cost tree that connects all terminals in T. In the sublinear model, the goal is to estimate the cost of an optimal Steiner tree in time sublinear in the input size, which is $\Theta(n^2)$, using only queries to the distance matrix.

Sublinear Metric Steiner Forest: Given a set V of n vertices and their pairwise distances, a set of terminal pairs $\mathcal{P} = \{(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)\}$ with $s_i, t_i \in V$, and query access to the distance matrix of a metric space over V, the *metric Steiner forest problem* asks for a minimum-cost forest that connects each terminal pair (s_i, t_i). In the sublinear model, the goal is to estimate the cost of an optimal Steiner forest in time sublinear in the input size, which is $\Theta(n^2)$, using only queries to the distance matrix.

Sublinear Path Cover Problem: Given an unweighted graph G, a *path cover* in G is a collection of vertex-disjoint paths in G. A *maximum path cover* is a path cover of G with the maximum number of edges in it (note that we are counting the total number of edges, not the number of paths). We use $\rho(G)$ to denote the size of a maximum path cover in G. In the sublinear model, the goal is to estimate $\rho(G)$ in time sublinear in the size of the input graph.

Chapter 3

Sublinear Time Algorithms for Maximum Matching

In this chapter, we study the sublinear matching problem from an algorithmic perspective. In Section 3.1 and Section 3.2, we present two algorithms that slightly improve upon the (1/2)-approximation. More specifically, the first algorithm achieves a $(1/2 + \Omega(1))$-approximation and can be implemented in $O(n^{1+\varepsilon})$ time, where constant $\varepsilon > 0$ can be made arbitrarily small. In Section 3.2, we design a simpler algorithm that achieves a 0.5109-approximation with a running time of $\widetilde{O}(n\sqrt{n})$. Finally, in Section 3.3, we present a $(2/3 - \varepsilon)$-approximation algorithm running in $n^{2-\text{poly}(\varepsilon)}$ time for any arbitrarily small constant $\varepsilon > 0$. We further complement the results of this section by designing a $(2/3 + \Omega(1))$-approximation algorithm that runs in strongly sublinear time, specifically $n^{2-\Omega(1)}$ for bipartite graphs.

3.1 Beating Greedy Matching in Sublinear Time

Using n, m, Δ, and $\bar{d}$ to respectively denote the number of vertices, the number of edges, the maximum degree, and the average degree in the graph, the results of this section can be summarized as follows:

Theorem 3.1.1. *For any constant $\varepsilon > 0$, there is a constant $\delta > 2^{-O(1/\varepsilon)}$ along with an algorithm that w.h.p. estimates the size of maximum matching up to a:*

(1) *multiplicative factor of $(\frac{1}{2} + \delta)$ in the adjacency list model in $\widetilde{O}(n + \Delta^{1+\varepsilon})$ time,*

(2) *multiplicative-additive factor of $(\frac{1}{2} + \delta, o(n))$ in the adjacency list model in $\widetilde{O}(\bar{d} \cdot \Delta^{\varepsilon})$ time,*

(3) *multiplicative-additive factor of $(\frac{1}{2} + \delta, o(n))$ in the adjacency matrix model in $O(n^{1+\varepsilon})$ time.*

A few remarks about the three results of Theorem 3.1.1:

- The constant $\delta > 0$ in Theorem 3.1.1 is miniscule. We did not attempt to optimize it, but do not expect our algorithm to obtain a significantly better than half approximation.
- Theorem 3.1.1–(1) comes close to an $\Omega(n)$ lower bound that holds for any $O(1)$-approximation in the adjacency list model. Any such algorithm must distinguish an empty graph from one with a single edge. This clearly requires $\Omega(n)$ queries in the adjacency list model.
- Theorem 3.1.1–(2) comes close to an $\Omega(\bar{d})$ lower bound for any $(O(1), o(n))$-approximation in the adjacency list model due to [159]. Observe that $\bar{d}\Delta^{\varepsilon} \ll m$ for any $\varepsilon < 1$. Therefore, this algorithm *always* runs in sublinear time in the number of edges in the graph.
- Theorem 3.1.1–(3) comes close to an $\Omega(n)$ lower bound for any $(O(1), o(n))$-approximation in the adjacency matrix model. Any such algorithm must distinguish an empty graph from one that includes a random perfect matching. This requires $\Omega(n)$ adjacency matrix queries.
- It takes $\Omega(n^2)$ queries to the adjacency matrix to distinguish an empty graph from one with only a single edge. Since this must be done for any multiplicative $O(1)$-approximation, no non-trivial such algorithm (i.e., one with $o(n^2)$ queries) exists for this model.

3.1.1 Technical Overview

We first give a brief overview of existing approaches and discuss why it takes quadratic time to implement. Then, we overview our main tool in breaking this quadratic time barrier through a less "adaptive" augmentation algorithm.

The Quadratic Barrier: A Brief Discussion of Earlier Techniques

Discovering (short) *augmenting paths*[1] is a natural way of improving the approximation for the maximum matching problem. For example, the Hopcroft-Karp [123] algorithm starts with an empty matching and iteratively applies a maximal set of vertex disjoint (short) augmenting paths. Each step of Hopcroft-Karp can essentially be viewed as a *maximal independent set* (MIS) instance. Put one vertex for each (short) augmenting path and connect two vertices if their corresponding augmenting paths share a vertex. An MIS

[1] See chapter 2 for the formal definition of augmenting paths.

in this graph corresponds to a maximal set of vertex disjoint augmenting paths. Building on this idea and by giving a size estimator for MIS, Yoshida, Yamamoto, and Ito [176] showed that for any integer $k \geq 1$, a $(\frac{k}{k+1}, o(n))$-approximation can be obtained in $O_k(\Delta^{6k(k+1)})$ time[2]. This is sublinear when Δ is sufficiently small.

Although the MIS-based approach is powerful enough to go well beyond 1/2-approximation in poly(Δ) time, it does not seem to help with general graphs where Δ can be large. This holds even if we limit ourselves to length-3 augmenting paths, which is needed for beating 1/2. The MIS size estimator of [176] crucially requires time at least linear in the average degree. Since every edge of a maximal matching can belong to $\Omega(\Delta^2)$ length-3 augmenting paths (with $\Omega(\Delta)$ choices from each endpoint of the edge), this average degree in the MIS graph can be $\Omega(\Delta^2)$ where Δ is the original graph's maximum degree. This makes it unlikely for this approach to yield an $o(\Delta^2)$ time algorithm. Additionally, not being able to construct the MIS graph explicitly in whole, and not having the edges of the maximal matching we are trying to augment also impose other Δ factors in the running time, arriving at the rather large poly(Δ) bound of [176].

There is an alternative way of discovering length-3 augmenting path which works directly with matchings instead of independent sets. The idea is to first find a maximal matching M of G, then find another maximal matching S on a subgraph H of G which includes a subset of the edges that have exactly one endpoint matched by M. Note that if both endpoints of an edge $e \in M$ are matched in S, then we get a length-3 augmenting path. This framework was first used by [140] in the context of random-order streaming algorithms, but has since been applied to various other settings [55, 39, 118]. Because the second graph H is defined *adaptively* based on M, we cannot simply run two independent instances of existing maximal matching estimators as black-box. In fact, this framework also hits a quadratic-in-degree time barrier as we describe next. To describe this barrier, we first briefly overview the key ideas behind the maximal matching size estimator of [34].

Consider a maximal matching S that is constructed greedily by iterating over the edges in some ordering π. It is not hard to see that $e \in S$ iff there is no edge e' incident to e such that $\pi(e') < \pi(e)$ and $e' \in S$. Therefore to determine whether $e \in S$, it suffices to go over the lower rank neighboring edges of e (in the increasing order of their ranks) and recursively query them to either find one that belongs to S, or conclude that e must be in S. This approach was first suggested by Nguyen and Onak [154]. The main question is the total number of recursive calls needed for the process to finish. The result of Behnezhad [34] is that for a *random* vertex v and a *random* permutation π, the process terminates in $O(\bar{d} \cdot \log n)$ expected total time, coming close to an $\Omega(\bar{d})$ lower bound.

Now, let us revisit the above two-step algorithm which first constructs a maximal matching M of G and then another maximal matching S of a subgraph H of G. Consider the task of determining whether a vertex v is matched by S. From [34], we get that for a random vertex v, this should be doable by exploring $\widetilde{O}(\bar{d}_H)$ edges of H in expectation where here we use $\bar{d}_X$ to denote the average degree of graph X. The challenge, however, is that since H is defined adaptively based on M, we do not a priori know the neighbors of v in H. In particular, to know whether an edge (u, v) exists in H, we have to ensure that exactly one of u and v is matched in M. Therefore, for exploring $\widetilde{O}(\bar{d}_H)$ edges in H, the naive approach would make $\widetilde{O}(\bar{d}_H)$ vertex queries to M. Note that these calls are not necessarily to random vertices anymore, which is crucial for the bound of [34] to work. But even if we manage to get an $\widetilde{O}(\bar{d}_G)$ time bound on each one of these calls we

[2] O_k ignores dependencies on k.

arrive at a total running time of $\widetilde{\Theta}(\bar{d}_H \cdot \bar{d}_G)$ which again can be as large as $\Omega(n^2)$.

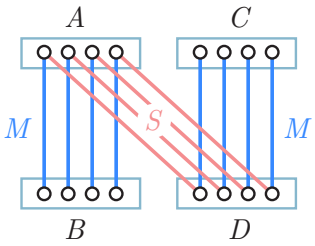
Our Contribution: A Less "Adaptive" Augmentation Algorithm

The two-stage algorithm we discussed earlier, constructs M fully and then adaptively picks the edges of S based on M. Our first step towards proving Theorem 3.1.1 is introducing a less "adaptive" algorithm that interleaves the construction of the two matchings M and S.

Our algorithm starts by constructing a sequence T, containing a single element $(e, \textsc{Extend})$ and $K \geq 1$ distinct elements $(e, \textsc{Start})$, corresponding to every edge e in the graph, where K is a parameter of the algorithm. We will process T in a *random order*. The role of having multiple copies of the Start elements is to bias them to appear earlier in the random permutation. We start by initializing two empty matchings M and S and then iterate over these randomly sorted $m(K+1)$ elements of T. Whenever we see an $(e, \textsc{Start})$ element, we add e to M iff both endpoints of e are unmatched in M. Therefore, M will be a random greedy maximal matching of G. Whenever we see an $(e, \textsc{Extend})$ element, we add e to S under a few conditions. The first condition is that both endpoints of e must be yet unmatched by S; this is to ensure that S continues to be a matching. The second condition is that at most one endpoint of e can be already matched by M. Our final algorithm, formalized as Algorithm 1, also checks two more technical conditions before adding e to S which are needed for the approximation ratio analysis. Once all of T is processed, the algorithm returns a maximum matching of $M \cup S$.

Observe that even though at the time of adding an edge (u, v) to S, at most one of its endpoints is matched in M, the other endpoint may get matched in M later in the process. Such edges of S cannot be used in length-3 augmenting paths for M. One way to avoid these bad events is to set K large enough so that all the Start elements appear before all Extend elements. However, this will negatively impact the running time. Specifically, the local query process to determine whether a random vertex v is matched in either of S or M takes $\widetilde{O}(\bar{d} \cdot K)$ time. This means that we need to set K to be much smaller than Δ to beat the quadratic-time-barrier. Indeed we set $K \approx \Delta^{\varepsilon}$ to get the bounds of Theorem 3.1.1.

When $K \ll \Delta$, we will inevitably have many edges of S for which both endpoints are matched in M. For example, consider the construction illustrated on the right with four vertex parts A, B, C, D. There is a regular bipartite graph of degree $\Theta(\Delta)$ between A and B, a regular bipartite graph of degree $\Theta(\Delta^{0.99})$ between A and D, and a regular bipartite graph of degree $\Theta(\Delta^{0.98}/K)$ between C and D. In this construction, the edges of M match A to B and C to D nearly completely. The edges of S match A to D nearly completely. This happens because there are many more Extend elements in (A, D) than there are Start elements in (C, D). Therefore, at the time of adding S, the part of M from C to D is not yet constructed.



Despite this bad event, we show that the edges of S are still useful in augmenting M. One key insight apparent in the above example is that the edges of (C, D) in M tend to have rank (in the permutation of T) roughly K times larger than those edges of (A, B) in M. We generalize this to all graphs and show that if an edge of S connects two edges of M, then the ranks of these edges of M must differ by a factor of roughly K.[3] Therefore, if instead of considering all edges of M for augmentation, we consider a subset $M_{j^\star}$ of M that is a constant fraction of M and at the same time any two edges in $M_{j^\star}$ have ranks within K factor of

[3]To be more precise, we only prove this for most edges of S, but not all.

each other, then S cannot connect two edges of $M_{j^\star}$ and so we can focus on augmenting just $M_{j^\star}$ instead of the whole matching M. See Section 3.1.2 for the full details of the algorithm and its approximation analysis.

So far we have only described an algorithm that finds a $(\frac{1}{2} + \Omega(1))$-approximate matching and have not yet described how to estimate its output size in sublinear time. To do this, we first define a query process akin to the one described above for maximal matching. That is, for a given edge e we define two query processes that respectively return whether $e \in M$ and $e \in S$. We then analyze the expected number of the recursive calls for a random start vertex by building on the techniques of [34, 176]. Several challenges arise along the way that are unique to our algorithm and require new ideas. For instance, an EXTEND element remains relevant (i.e., can still be added to S) until seeing two START elements in M, one from each endpoint. This is unlike the greedy approach, say for MIS or maximal matching, where an element (respectively a vertex and an edge) becomes irrelevant right after seeing one neighbor in the solution. Also note that we should not simply count the number of edges in M and S. Rather, we have to count the number of edges of M plus the number of length-3 augmenting paths that we find. To do this, when we find an edge $(u, v) \in S$ and an edge $(v, w) \in M$, we also query whether w is matched in S to a vertex left unmatched by M or not. This complicates the analysis because this vertex w is not picked uniformly at random anymore. The details of the query process and its analysis are provided in Section 3.1.3.

3.1.2 A Meta Algorithm for Beating the (1/2)-Approximation

The Algorithm

In this section, we formalize our new "less adaptive" meta algorithm that we informally overviewed in Section 3.1.1. Next, we show that its approximation ratio is strictly better-than-half. We later show in Section 3.1.3 that the size of its output matching can be estimated in sublinear time.

Before formalizing the algorithm, let us give a few useful definitions. Given an n-vertex graph of maximum degree Δ, and a parameter $\varepsilon \in (0, .25)$, define:

- $p := 0.007$.
- $D := (c \cdot \Delta \cdot \log n)^{\varepsilon}$ where we later fix $c \geq 1$ to be sufficiently large function of ε.
- $K := 10D \log^2 n$.
- $\alpha_i := 1/D^{i-1}$ for $i \in [2/\varepsilon]$ and $\alpha_{2/\varepsilon+1} = 0$. Note that $0 = \alpha_{2/\varepsilon+1} < \alpha_{2/\varepsilon} < \ldots < \alpha_2 < \alpha_1 = 1$.

We are now ready to state the algorithm, which is formalized below as Algorithm 1.

The Approximation Guarantee

In this section, we prove the following approximation guarantee for Algorithm 1. (We note that we have not attempted to optimize the constants in the statement.)

Theorem 3.1.2. *Let G be any n-vertex graph. Let M and S be the matchings produced by Algorithm 1 run on G for parameter $\varepsilon \in (0, .25)$. Then for some $\delta > 2^{-O(1/\varepsilon)}$,*

$$\mathbf{E}[\mu(M \cup S)] \geq \left(\frac{1}{2} + \delta\right) \mu(G).$$

Algorithm 1: An algorithm for beating half-approximate matching.

Input: An n-vertex m-edge graph $G = (V, E)$ of max degree Δ.
Parameter: $\varepsilon \in (0, .25)$.
Define K, p, and $\alpha_{2/\varepsilon}, \ldots, \alpha_1$ as above.
Construct a sequence T, which for any edge $e \in E$ includes K copies of $(e, \textsc{Start})$ and one copy of $(e, \textsc{Extend})$. Then random shuffle the elements in T.
For any vertex $v \in V$ pick a color $c_v \in \{\textsc{Blue}, \textsc{Red}\}$ uniformly and independently.
Initialize $M \leftarrow \emptyset, S \leftarrow \emptyset$.; `// Both` M `and` S `will be matchings of` G`.`
Initialize $M_1 \leftarrow \emptyset, \ldots, M_{2/\varepsilon} \leftarrow \emptyset$.; `// These will partition the edges in` M`.`
Draw $j^\star$ from $[2/\varepsilon]$ uniformly at random.
for $i = 1$ *to* $|T|$ **do**
    Let $(e = \{u, v\}, X)$ be the i-th element in T.
    if $X = \textsc{Start}$ *and* $\deg_M(u) = \deg_M(v) = 0$ **then**
        Add e to M.
        Add e to the unique M_i, $i \in [2/\varepsilon]$ where $\alpha_{i+1}|T| < i \leq \alpha_i|T|$.
        if $c_v = c_u$ *or* $e \notin M_{j^\star}$ **then**
            Mark both u and v as *frozen*.
        else
            With probability $1 - p$ mark both u and v as *frozen*.
    if $X = \textsc{Extend}$, $\deg_S(u) = \deg_S(v) = 0$, $\deg_M(v) + \deg_M(u) \leq 1$, $c_u \neq c_v$, *and neither endpoint of* e *is frozen* **then**
        Add e to S.
return the maximum matching in $M \cup S$.

Basic Notation and Definitions

For any element ℓ we use $\pi(\ell)$ to denote the location of ℓ in T. For any edge $e \in E$ we use $\pi_{\textsc{Start}}(e)$ (resp. $\pi_{\textsc{Extend}}(e)$) to denote the minimum $i \in [|T|]$ such that the i-th index of T includes element $(e, \textsc{Start})$ (resp. $(e, \textsc{Extend})$). For any element $\ell \in T$ we say "*at the time of processing* ℓ" to refer to the iteration of the for loop in Algorithm 1 when i equals $\pi(\ell)$.

Next, we define unusual edges as follows:

Definition 3.1.3 (unusual edges)**.** *We say an edge $e \in M$ is* unusual*, if there is some edge $e' = (u, v)$ incident to e such that one of the following holds:*

- $\pi_{\textsc{Start}}(e) < \pi_{\textsc{Extend}}(e') < D \cdot \pi_{\textsc{Start}}(e)$*, and at the time of processing $(e', \textsc{Extend})$ the only edge in M that is incident to e' is e.*
- $\pi_{\textsc{Extend}}(e') < \pi_{\textsc{Start}}(e)$*, and at the time of processing $(e', \textsc{Extend})$ no edge in M is incident to e' in M.*

We use A to denote the subset of unusual edges of M.

In Section 3.1.2 we prove the following Lemma 3.1.4 which shows only a small fraction of the edges in M will be unusual. This is one of our key insights towards our proof of Theorem 3.1.2, and is the main place where having K copies of $(e, \textsc{Start})$ compared to one copy of $(e, \textsc{Extend})$ in T is used crucially.

Lemma 3.1.4. $\mathbf{E}\,|A| = o(|M|)$.

The Main Argument

In this section, we present the main building blocks of the proof of Theorem 3.1.2, deferring the proof of one key lemma (Lemma 3.1.10) to a later section.

Our proof of Theorem 3.1.2 relies on four independent sources of randomization, which with a slight abuse of notation we denote by $j^\star$, T, C, and F:

- T: The order in which Algorithm 1 processes T.
- C: The colors assigned to the vertices in Algorithm 1 of Algorithm 1.
- $j^\star$: The index chosen in Algorithm 1 of Algorithm 1.
- F: The set of edges in M that get frozen in Algorithm 1 of Algorithm 1.

All four sources of randomization are needed for the proof of Theorem 3.1.2. But it would be convenient to first condition on T because:

Observation 3.1.5. *Conditioning on T fully reveals the maximal matching M, the set A of unusual edges in M, and all of $M_1, \ldots, M_{2/\varepsilon}$.*

Note, however, that S remains random even after conditioning on T as it depends on the other three sources of randomization too.

Because M is a maximal matching of G, we immediately get $|M| \geq \mu(G)/2$. Our plan is to show that if M is only half the size of $\mu(G)$, then in expectation S augments it well enough that M and S together include a larger matching. More formally, recall that our goal in Theorem 3.1.2 is to prove that $\mathbf{E}[\mu(M \cup S)] \geq (\frac{1}{2} + \delta)\mu(G)$. This clearly holds if $|M| \geq (\frac{1}{2} + \delta)\mu(G)$. So let us for the rest of the proof assume that M is smaller.

Assumption 3.1.6. $|M| < (\frac{1}{2} + \delta)\mu(G)$.

Plugging this assumption into Claim 2.2.2 gives:

Observation 3.1.7. *At least $|M| - 4\delta|M^\star|$ edges of M belong to length-3 augmenting paths in $M \oplus M^\star$.*

Our next claim shows that there is one subset M_j of M that has several nice properties. We will later argue that M_j will be well augmented by S in expectation.

Claim 3.1.8. *Define $q(x) := 2^{20(x-3/\varepsilon)}$. There is M_j such that all the following hold:*

1. $|M_j| \geq q(j)|M|$,
2. $|M_1| + \ldots + |M_{j-1}| \leq 2^{-19}|M_j|$,
3. *For any two edges $e, e' \in M_j$, $\pi_{\textsc{Start}}(e')/D \leq \pi_{\textsc{Start}}(e) \leq \pi_{\textsc{Start}}(e') \cdot D$.*

Proof. First, there should exist $j \in [2/\varepsilon]$ such that $|M_j| \geq q(j)|M|$ as otherwise

$$|M_1| + \ldots + |M_{2/\varepsilon}| < \sum_{i=1}^{2/\varepsilon} q(i)|M| = \frac{2^{20} + \ldots + 2^{40/\varepsilon}}{2^{60/\varepsilon}}|M| < |M|,$$

which contradicts the fact that M_i's partition M. Take the smallest j with $|M_j| \geq q(j)|M|$, noting that the first property of the lemma is satisfied for M_j. We have

$$|M_1| + \ldots + |M_{j-1}| < \sum_{i=1}^{j-1} q(i)|M| < \frac{2^{20} + \ldots + 2^{20(j-1)}}{2^{60/\varepsilon}}|M| < 2^{20j-19-60/\varepsilon}|M| < 2^{-19}|M_j|,$$

so the second property also holds for M_j.

For the third property, note from the definition of $\alpha_1, \ldots, \alpha_{2/\varepsilon+1}$ that if $j \neq \frac{2}{\varepsilon}$, then all edges e in M_j have the same $\pi_{\text{START}}(e)$ up to a factor of D. So it suffices to show $j \neq \frac{2}{\varepsilon}$. Observe that for every $e \in M_{2/\varepsilon}$, by definition we have

$$\pi_{\text{START}}(e) \leq \alpha_{\frac{2}{\varepsilon}}|T| = \frac{|T|}{D^{2/\varepsilon-1}} \leq \frac{2mK}{D^{2/\varepsilon-1}} = \frac{20m\log n}{D^{2/\varepsilon-2}} = \frac{20m\log n}{(c\Delta\log n)^{\varepsilon(2/\varepsilon-2)}} < \frac{20m\log n}{c\Delta\log n} = \frac{20m}{c\Delta}.$$

Now by choosing c to be a sufficiently large function of ε, we can further guarantee that

$$\pi_{\text{START}}(e) < \frac{q(1)m}{4\Delta} \leq q(1)|M|,$$

where the last inequality follows because[4] $\mu(G) \geq \frac{m}{2\Delta}$ and $|M| \geq \mu(G)/2$. Since there are less than $q(1)|M|$ edges e for which $\pi_{\text{START}}(e) < q(1)|M|$, we get $|M_{2/\varepsilon}| < q(1)|M|$. Combined with the first property of the claim that $|M_j| \geq q(j)|M|$ and given that $q(x) \geq q(1)$ for every $x \geq 1$, we get that $j \neq 2/\varepsilon$, implying the third property and completing the proof. □

Now consider the set P_j all length three augmenting paths in $M^\star \oplus M$ where the middle edge belongs to M_j and that the vertices along these paths are alternatively BLUE and RED as follows:

$$P_j := \left\{ (x,u,v,y) \;\middle|\; \begin{array}{l} (x,u,v,y) \text{ is an augmenting path for } M,\ (x,u) \in M^\star, (v,y) \in M^\star, \\ (u,v) \in M_j,\ c_x = \text{RED},\ c_u = \text{BLUE},\ c_v = \text{RED},\ c_y = \text{BLUE} \end{array} \right\}.$$

Observation 3.1.9. *Conditioning on T and C fully reveals P_j.*

Proof. Definition P_j only depends on the vertex colors which are determined by C, and matchings M_j and M which are fully determined by T. □

The following lemma, which conditions on everything except F, is the key to Theorem 3.1.2. We defer its proof to the later section.

Lemma 3.1.10. *Let us condition on T, C, $j^\star = j$, and let P_j and M_j be as above. Let Y denote the number of length three augmenting paths for M in $M \oplus S$. Then*

$$\mathbf{E}_F\left[Y \mid T, C, j^\star = j\right] \geq p|P_j| - \left(4p^2 + 2^{-18}\right)|M_j| - 12|A|.$$

Let us first see how Lemma 3.1.10 implies Theorem 3.1.2.

Proof of Theorem 3.1.2. We assume Assumption 3.1.6 holds, or otherwise the theorem is trivial. Recall from Observation 3.1.7 that at most $4\delta\mu(G)$ edges of M (and thus M_j) are not in length-three augmenting paths

[4]Edge color the graph greedily using 2Δ colors and pick the largest color class which will be a matching.

in $M \oplus M^\star$. Since the colors are random and independent, each of these length-three augmenting paths is colored in the way specified in the definition of P_j with probability exactly $1/2^4$. Hence,

$$\mathbf{E}_C[|P_j|] \geq \frac{1}{2^4}(|M_j| - 4\delta\mu(G)). \tag{3.1}$$

Additionally, recall from Claim 3.1.8 that

$$|M_j| \geq q(j)|M| \geq q(1)|M| \geq \frac{q(1)}{2}\mu(G). \tag{3.2}$$

Also recall from Lemma 3.1.4 that

$$\mathbf{E}_T[|A|] = o(\mu(G)). \tag{3.3}$$

Taking expectation over C and T from both sides of the inequality of Lemma 3.1.10, we get

$$\begin{aligned}
\mathbf{E}_{F,C,T}[Y \mid j^\star = j] &\geq \mathbf{E}_{C,T}\left[p|P_j| - \left(4p^2 + 2^{-18}\right)|M_j| - 12|A|\right] \\
&\geq \frac{p}{16}(|M_j| - 4\delta\mu(G)) - \left(4p^2 + 2^{-18}\right)|M_j| - o(\mu(G)) && \text{(By (3.1) and (3.3))} \\
&= \left(\frac{p}{16} - 4p^2 - 2^{-18}\right)|M_j| - \frac{p\delta}{4}\mu(G) - o(\mu(G)) \\
&\geq \left(\left(\frac{p}{16} - 4p^2 - 2^{-18}\right)\frac{q(1)}{2} - \frac{p\delta}{4} - o(1)\right)\mu(G) && \text{(By (3.2).)} \\
&> \left(10^{-4}q(1) - 0.002 \cdot \delta - o(1)\right)\mu(G) && \text{(Since } p = 0.007.\text{)} \\
&> \frac{2\delta}{\varepsilon}\mu(G). && \text{(Since } q(1) = 2^{20-60/\varepsilon},\ \delta = 2^{-70/\varepsilon}.\text{)}
\end{aligned}$$

This, in turn, implies that

$$\mathbf{E}_{F,C,T,j^\star}[Y] \geq \Pr[j^\star = j] \cdot \mathbf{E}_{F,C,T}[Y \mid j^\star = j] \geq \frac{\varepsilon}{2} \cdot \frac{2\delta}{\varepsilon}\mu(G) = \delta\mu(G).$$

Since Y is a lower bound on the number of length three augmenting paths for M in $M \oplus S$, we get

$$\mathbf{E}_{F,C,T,j^\star}[\mu(M \cup S)] \geq |M| + \mathbf{E}_{F,C,T,j^\star}[Y] \geq \left(\frac{1}{2} + \delta\right)\mu(G),$$

which is the desired bound. □

Proof of Lemma 3.1.10

Proof. For brevity, we use $\mathbf{E}'[X]$ as a shorthand for $\mathbf{E}_F[X \mid T, C, j^\star = j]$ throughout the proof.

Recall that in Algorithm 1, when visiting an element $((u, v), \textsc{Extend})$ in T, we add it to S if all the following conditions hold, where we have deliberately broken the last condition in Algorithm 1 of Algorithm 1 into two sub-conditions (C4) and (C5):

(C1) $\deg_S(u) = \deg_S(v) = 0$,

(C2) $\deg_M(v) + \deg_M(u) \leq 1$,

(C3) $c_u \neq c_v$,

(C4) neither of u or v is frozen in Algorithm 1 of Algorithm 1.

(C5) neither of u or v is frozen in Algorithm 1 of Algorithm 1.

Since we have conditioned on T and C, both the matching M and the vertex colors are fully revealed. Thus, every element $((u,v), \textsc{Extend}) \in T$ which upon being visited violates one of the conditions (C2), (C3), or (C4) is fully revealed a priori and can be discarded. On the flip side, condition (C5) depends on F and so remains random.

Now suppose for the sake of the analysis that we also discard all elements $(e, \textsc{Extend})$ where e is incident to an unusual edge in M. We emphasize that discarding these elements might change matching S, but we will show later that this effect is not significant.

Useful definitions: Next, we give a few useful definitions that we use in the proof. First, define

$$\hat{B} := \{v \mid \exists (u,v) \in M_j, c_v = \textsc{Blue}, c_u = \textsc{Red}\}$$
$$\hat{R} := \{u \mid \exists (u,v) \in M_j, c_v = \textsc{Blue}, c_u = \textsc{Red}\}.$$

Also let U be the set of vertices that are left unmatched by $M_j, \ldots, M_{2/\varepsilon}$, and define

$$U_R := \{u \mid u \in U, c_u = \textsc{Red}\}, \qquad U_B := \{v \mid v \in U, c_v = \textsc{Blue}\}.$$

Moreover, define matchings M_B and M_R as

$$M_B := \{(x,u) \mid \exists (x,u,\cdot,\cdot) \in P_j\}, \qquad M_R := \{(y,v) \mid \exists (\cdot,\cdot,v,y) \in P_j\}.$$

Finally, define H_B (resp. H_R) to be the bipartite graph with vertex parts $\hat{B}$ and U_R (resp. $\hat{R}$ and U_B), including an edge e of G between its vertex parts iff $(e, \textsc{Extend})$ is not discarded.

It can be confirmed from the definitions above that the sets $\hat{B}$, $\hat{R}$, U_R, U_B are all disjoint. This implies that H_B and H_R are vertex disjoint. The next observation also follows immediately from the definitions above.

Observation 3.1.11. *All edges of M_B (resp. M_R) belong to H_B (resp. H_R).*

See Figure 3.1 for an illustration of some of these definitions.

We show that H_B and H_R include all the remaining Extend elements.

Claim 3.1.12. *For every element $(e, \textsc{Extend})$ that is not discarded, e belongs to H_B or H_R.*

Proof. Take an undiscarded element $(e = (u,v), \textsc{Extend})$. Take the edge $e' = (u,w) \in M$ with the smallest $\pi_{\textsc{Start}}(e')$ that is incident to e. Note that such e' should exist because M is a maximal matching of G. There are three possible cases, and only the last one does not lead to a contradiction:

- **Case 1** – $\pi_{\textsc{Extend}}(e) < \pi_{\textsc{Start}}(e')$: In this case, e' is unusual by the second condition of Definition 3.1.3 and so $(e, \textsc{Extend})$ must be discarded, a contradiction.
- **Case 2** – $\pi_{\textsc{Extend}}(e) \geq \pi_{\textsc{Start}}(e')$ and ($e' \notin M_{j^\star}$ or $c_u = c_w$): In this case, the endpoints of e' must be frozen in Algorithm 1 of Algorithm 1. So condition (C4) does not hold for e when processing $(e, \textsc{Extend})$ and we should discard it, a contradiction.

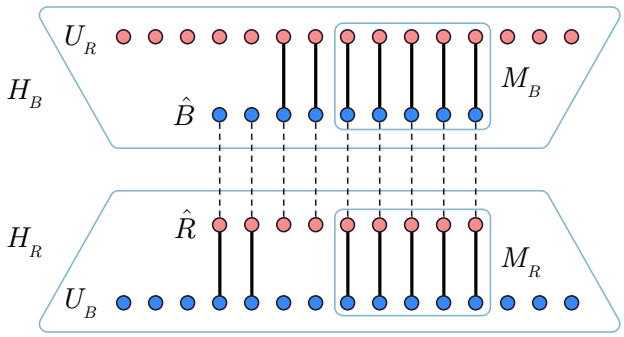


Figure 3.1: Illustration of $\hat{B}$, $\hat{R}$, U_B, U_R, H_B, H_R, and matchings M_B, M_R. The solid edges are the edges of $M^\star$, and the dashed edges are the edges of M_j whose endpoints have different colors.

- **Case 3** – $\pi_{\text{EXTEND}}(e) \geq \pi_{\text{START}}(e')$ and $e' \in M_{j^\star}$ and $c_u \neq c_w$: This is the only case that does not lead to a contradiction.

The condition of Case 3 immediately implies that if $c_u = \text{BLUE}$ then $u \in \hat{B}$ otherwise $u \in \hat{R}$.

Next, we show that v must be unmatched by $M_{j^\star}, \ldots, M_{2/\varepsilon}$, and so $v \in U$. First, note that if v is also matched by M through an edge e'', then by definition of e' it must satisfy $\pi_{\text{START}}(e'') > \pi_{\text{START}}(e')$. Since $e' \in M_{j^\star}$ and the ranks of the edges of $M_{j^\star+1}, \ldots, M_{2/\varepsilon}$ are all smaller than those in $M_{j^\star}$, this implies $e'' \notin M_{j^\star+1}, \ldots, M_{2/\varepsilon}$. So it remains to show $e'' \notin M_{j^\star}$. To see this, note that if $e'' \in M_{j^\star} = M_j$, then from Claim 3.1.8 part 3, we get $\pi_{\text{START}}(e'') < \pi_{\text{START}}(e') \cdot D$. Now if $\pi_{\text{EXTEND}}(e) > \pi_{\text{START}}(e'')$, then at the time of processing (e, EXTEND) both e' and e'' are in M and we have $\deg_M(u) + \deg_M(v) = 2$, which means (e, EXTEND) violates (C2) and must be discarded, a contradiction. So we should have $\pi_{\text{EXTEND}}(e) < \pi_{\text{START}}(e'') < \pi_{\text{START}}(e') \cdot D$. This is again a contradiction because by the first condition of Definition 3.1.3, e' must be unusual, and so e must be discarded.

Finally, note that since (e, EXTEND) is not discarded, it should satisfy (C3) and so $c_u \neq c_v$. Combined with the discussion above this means that if $u \in \hat{B}$ then $v \in U_R$ and if $u \in \hat{R}$ then $v \in U_B$. Hence e must belong to one of H_B and H_R, completing the proof. □

Now consider the construction of S from the remaining undiscarded elements. We iterate over these elements in the order specified by T, and whenever we see an element (e, EXTEND) that satisfies all of (C1)–(C5) we add it to S. As discussed, conditions (C2)–(C4) are automatically satisfied by all undiscarded elements, which only leaves (C1) and (C5). Condition (C1) is simply the greedy matching constraint. Condition (C5) depends on the randomization in F. An edge in H_B (resp. H_R) satifies (C5) if its endpoint in $\hat{B}$ (resp. $\hat{R}$) is not frozen. An important observation is that each vertex in $\hat{B}$ (resp. $\hat{R}$) is frozen independently from the other vertices of $\hat{B}$ (resp. $\hat{R}$) with probability $1 - p$. (We emphasize though that these decisions are not mutually independent when we consider the vertices of both $\hat{B}$ and $\hat{R}$ together.)

Let S_B and S_R be the subset of edges of S that respectively belong to H_B and H_R. Our goal is to apply Proposition 2.2.3 on both H_B and H_R.

First, we apply Proposition 2.2.3 by letting $G = H_B$, $V = \hat{B}$, $U = U_R$, the subsample W being the subset of vertices in $\hat{B}$ that are not frozen, and $M = M_B$ (recalling from Observation 3.1.11 that M_B is completely inside H_B). Using X_B to denote the set of vertices in $V(M_B) \cap \hat{B}$ that get matched in S to U_R, we get from

Proposition 2.2.3 that:

$$\mathbf{E}'[|X_B|] \geq p(|M_B| - 2p|\hat{B}|) \geq p|P_j| - 2p^2|M_j|.$$

Next, we apply Proposition 2.2.3 by letting $G = H_R$, $V = \hat{R}$, $U = U_B$, the subsample W being the subset of vertices in $\hat{R}$ that are not frozen, and $M = M_R$ (recalling from Observation 3.1.11 that M_R is completely inside H_R). Using X_R to denote the set of vertices in $V(M_R) \cap \hat{R}$ that get matched in S to U_B, we get from Proposition 2.2.3 that:

$$\mathbf{E}'[|X_R|] \geq p(|M_R| - 2p|\hat{R}|) \geq p|P_j| - 2p^2|M_j|.$$

Adding back the discarded EXTEND **elements:** We now add back the EXTEND elements incident to unusual edges that we discarded earlier. To do so, we iteratively take an arbitrary vertex of an arbitrary unusual edge in M, and add back all the EXTEND elements incident to it that we discarded, and re-compute matching S.

Claim 3.1.13. *Let S_1 and S_2 be the edges in matching S before and after adding back the discarded edges of a vertex v. At most two vertices can be matched in one of S_1, S_2 bot not the other.*

Proof. Since S_1 and S_2 are both matchings, $S_1 \Delta S_2$ is a collection of paths and cycles. Thus any vertex whose matching-status differs in S_1 and S_2 must be an endpoint of a path in $S_1 \Delta S_2$. Suppose for contradiction that there are more than two such vertices. Then we have more than one path in $S_1 \Delta S_2$. Take the lowest rank edge e in the path that does not include v. It can be confirmed that whether the conditions (C1)–(C5) are satisfied for e remains the same in both S_1 and S_2, so either e belongs to both or neither, contradicting that $e \in S_1 \Delta S_2$. □

Let us now define X'_B and X'_R to be the analogs of X_B and X_R after we add back the discarded edges incident to unusual edges of M. More precisely, let X'_B (resp. X'_R) denote the set of vertices in $V(M_B) \cap \hat{B}$ (resp. $V(M_R) \cap \hat{R}$) that are matched in S to U_R (resp. U_B).

Note that a vertex v may belong to $X_B \setminus X'_B$ for two reasons: either v was matched in S before adding back the discarded edges but then got unmatched after doing so, or v remains matched in S, but to a vertex not in U_R. Claim 3.1.13 bounds the total number of vertices of the former type by $2 \times 2|A| = 4|A|$. For the latter type, note from Claim 3.1.12 that any edge of v that is not discarded goes to U_R. So the new match of v after adding the discarded edges must be a discarded edge. But each discarded edge that belongs to S must match one endpoint of one of the $|A|$ unusual edges in M, so the total number of such edges is no more than $2|A|$. Thus, overall

$$\mathbf{E}'[|X'_B|] \geq \mathbf{E}'[|X_B|] - 6|A| \geq p|P_j| - 2p^2|M_j| - 6|A|. \tag{3.4}$$

Applying the same argument on X'_R gives

$$\mathbf{E}'[|X'_R|] \geq \mathbf{E}'[|X_R|] - 6|A| \geq p|P_j| - 2p^2|M_j| - 6|A|. \tag{3.5}$$

Now, let UF_j denote the set of M_j edges of P_j that are unfrozen. Each edge $e \in UF_j$ has one endpoint colored BLUE and one that is colored RED by definition of P_j. The set of BLUE (resp. RED) endpoints of

UF_j that are matched in S to a vertex in U_R (resp. U_B) is exactly X'_B (resp. X'_R). We have:

$$\begin{aligned}\mathbf{E}'[\#\text{ of } (u,v)\in UF_j \text{ s.t. } u\in X'_B, v\in X'_R] &\geq \mathbf{E}'[|UF_j| - (|UF_j| - |X'_B|) - (|UF_j| - |X'_R|)]\\ &= \mathbf{E}'[|X'_B|] + \mathbf{E}'[|X'_R|] - \mathbf{E}'[|UF_j|]\\ &\geq p|P_j| - 4p^2|M_j| - 12|A|. \end{aligned} \tag{3.6}$$

The last inequality follows from (3.4) and (3.5), and the fact that $\mathbf{E}[|UF_j|] = p|P_j|$ because P_j has $|P_j|$ edges in M_j and each one is unfrozen with probability p.

To finish the proof, note that if for an edge $(u,v)\in UF_j$ we have $u\in X'_B$ and $v\in X'_R$, then S matches both u and v to vertices that are unmatched by $M_j,\ldots,M_{2/\varepsilon}$. Therefore, only if these vertices are also unmatched by $M_1,\ldots,M_{j-1}$ we have a length three augmenting path. Therefore,

$$\begin{aligned}\mathbf{E}'[Y] &\geq \mathbf{E}'[\#\text{ of } (u,v)\in UF_j \text{ s.t. } u\in X'_B, v\in X'_R] - 2\sum_{i=1}^{j-1}|M_i|\\ &\geq p|P_j| - 4p^2|M_j| - 12|A| - 2\times\frac{1}{1000}|M_j| && \text{(By (3.6) and Claim 3.1.8.)}\\ &= p|P_j| - \left(4p^2+\frac{1}{500}\right)|M_j| - 12|A|.\end{aligned}$$

This finishes the proof of Lemma 3.1.10 □

Bounding Unusual Edges

In this section we prove Lemma 3.1.4 that $\mathbf{E}\,|A| = o(|M|)$.

We start by proving the following auxiliary claim.

Claim 3.1.14. *With probability $1-1/n^2$, every edge $e\in E$ satisfies $\pi_{\textsc{Start}}(e)\leq 8m\log n$.*

Proof. Let $z := 8m\log n$. We show that with probability $1-1/n^2$ every edge $e\in E$ satisfies $\pi_{\textsc{Start}}(e)\leq z$. This statement is trivial if $|T|\leq z$ so assume $|T|>z$. Fix an arbitrary edge $e\in E$. For the event $\pi_{\textsc{Start}}(e) > z$ to happen, all K copies of $(e,\textsc{Start})$ should appear after the first z elements in T. Thus, noting that $|T| = m(K+1)$, $z = 8m\log n$, and $K\geq 1$, we have

$$\Pr[\pi_{\textsc{Start}}(e)\leq z] \geq 1-\left(1-\frac{z}{|T|}\right)^K = 1-\left(1-\frac{8m\log n}{m(K+1)}\right)^K \geq 1-e^{-\frac{8K\log n}{K+1}} \geq 1-n^{-4}.$$

A union bound over all choices of e, which there are less than n^2 many, proves our first claim. □

We are now ready to prove Lemma 3.1.4.

Proof of Lemma 3.1.4. We count the number of unusual edges separately based on the two cases in Definition 3.1.3. Consider the second case. Let A' be the set of $(e',\textsc{Extend})$ elements in T such that at the time of processing $(e',\textsc{Extend})$, both endpoints of e' are unmatched in M. If we show that $\mathbf{E}_\pi\,|A'| = o(|M|)$, since each edge in A' has at most two incident edges in M, the number of unusual edges in the second case of Definition 3.1.3 will be $o(|M|)$. Consider the following equivalent construction of A'. We iterate over T, processing its elements one by one. If we see an element $(e,\textsc{Start})$ we add e to M and remove all the

unprocessed elements (e', X), $X \in \{\textsc{Extend}, \textsc{Start}\}$ from T such that e' shares an endpoint with e. If we see an element $(e, \textsc{Extend})$, we add it to A'.

At any time during this process, the remaining $\textsc{Start}$ elements are at least K times more than the $\textsc{Extend}$ element since for each $\textsc{Extend}$ element $(e, \textsc{Extend})$ that remains in T, all K copies of $(e, \textsc{Start})$ must also remain in T. This means that every time we reveal the next remaining element of T, it is an $\textsc{Extend}$ element with probability at most $1/(K+1)$. Furthermore, there are at most $\mu(G)$ steps where we process a $\textsc{Start}$ element since each time we add an edge to M and clearly $|M| \leq \mu(G)$. This implies that $\mathbf{E}\,|A'| \leq \frac{\mu(G)}{K}$. Combining $\mu(G) \leq 2|M|$ and $K = \Omega(\log n)$, we have $\mathbf{E}\,|A'| = o(|M|)$.

Now consider the first case in Definition 3.1.3. Define $\sigma : T \to \mathbb{R}$ as $\sigma((e, \textsc{Start})) := \pi((e, \textsc{Start}))$ and $\sigma((e, \textsc{Extend})) := \pi((e, \textsc{Extend}))/D$. Similar to the proof of the second case, we process the elements in T one by one. However, instead of π, we process them in the increasing order of their σ values. Let A'' be the set of $\textsc{Extend}$ elements in T such that at the time of processing $(e', \textsc{Extend})$ both endpoints of e' are unmatched in M. Note that if an edge e is unusual because of the first case of Definition 3.1.3, there exists an element $(e', \textsc{Extend})$ such that $\pi_{\textsc{Extend}}(e')/D < \pi_{\textsc{Start}}(e)$ and at the time of processing $(e', \textsc{Extend})$, the only incident edge to e' in M is e. Hence, we must process $(e', \textsc{Extend})$ before $(e, \textsc{Start})$ according to new ordering of elements and at the time of processing $(e', \textsc{Extend})$, there is no incident edge to e' in M. Therefore, $|A''|$ is an upperbound for number of unusual edges in first case of Definition 3.1.3. Similar to the previous case, it suffices to show $\mathbf{E}_\pi\,|A''| = o(|M|)$ since each edge in A'' has at most two incident edges in M. Consider again the following equivalent construction of A'', where we iterate over the elements based on σ one by one. If we see an element $(e, \textsc{Start})$ we add e to M and remove all the unprocessed elements (e', X), $X \in \{\textsc{Extend}, \textsc{Start}\}$ such that e' shares an endpoint with e. If we see an element $(e, \textsc{Extend})$, we add it to A''.

Let E_S be the event that $\pi_{\textsc{Start}}(e) \leq 8m \log n$ for all edges $e \in E$, and let $\bar{E}_S$ be the complement event. Noting from Claim 3.1.14 that $\Pr[\bar{E}_S] \leq 1/n^2$. We get

$$\begin{aligned}
\Pr[\sigma((e, \textsc{Extend})) \leq \pi_{\textsc{Start}}(e)] &= \Pr\left[\pi_{\textsc{Extend}}(e) \leq D \cdot \pi_{\textsc{Start}}(e)\right] \\
&\leq \Pr[\bar{E}_S] + \Pr\left[\pi_{\textsc{Extend}}(e) \leq D \cdot \pi_{\textsc{Start}}(e) \mid E_S\right] \\
&\leq \frac{1}{n^2} + \frac{8mD \log n}{m(K+1)} \\
&\leq \frac{8D \log n}{K}.
\end{aligned}$$

Hence, the probability of the element that we are processing to be an $\textsc{Extend}$ element is at most $\frac{8D \log n}{K}$. Similar to the former case, since there are at most $\mu(G)$ steps in the process that an $\textsc{Start}$ element is added to M, we have $\mathbf{E}\,|A''| \leq \frac{\mu(G) \cdot 16D \log n}{K}$. Combining with $\mu(G) \leq 2|M|$, $D = (c \cdot \Delta \cdot \log n)^\varepsilon$, and $K = 10D \log^2 n$, implies $\mathbf{E}\,|A''| = o(|M|)$ which completes the proof. □

3.1.3 A Local Query Algorithm and its Complexity

In this section, we present a local query process that determines whether a given vertex has any edge in matchings M and S of Algorithm 1 without constructing the whole output of Algorithm 1.

While all the randomizations in Algorithm 1 were crucial for the approximation guarantee of Section 3.1.2 to hold, the bounds of this section only rely on the random shuffling of the sequence T. In particular, the

result of this section continues to hold even if all the other coin flips of Algorithm 1 besides the order of T are picked adversarially.

To help the discussion above simplify the presentation of this section, we regard the index $j^\star$ and the color c_v of any vertex v as given. Alternatively, one could pick the color c_v of any vertex uniformly at random whenever we access c_v.

Note from Algorithm 1 that whether a vertex is frozen depends on its edge in M, if any. In particular, a vertex v is frozen iff it is matched in M and its match is also frozen. This can essentially be seen as freezing the edges of M instead of the vertices, which will be the more convenient view for our purpose in this section. More generally, instead of just the START elements that end up in M, we define (Definition 3.1.15) whether a START element of T is frozen or not. This way, we say a vertex v is frozen iff there is an edge e incident to v such that $e \in M$ and the corresponding (e, START) element in T that adds e to M is frozen.

Definition 3.1.15. *We say an element $\ell = ((u, v), \text{START}) \in T$ is* frozen *if either of the following conditions holds:*

- $c_u = c_v$,
- $\pi(\ell) \leq \alpha_{j^\star+1}|T|$ *or* $\pi(\ell) > \alpha_{j^\star}|T|$,
- *The corresponding* Bernoulli$(1-p)$ *variable in Algorithm 1 of Algorithm 1 is one.*

We emphasize again that the randomization in the last bullet of Definition 3.1.15 is only needed for the approximation guarantee, and can be regarded as (adversarially) fixed for our purpose in this section.

Now let π be a permutation of the sequence T consisting of edge copies in graph G. We write $\text{MS}(G, \pi)$ to denote the subgraph $M \cup S$ constructed by Algorithm 1 when processing T in the order of π. The argument π in $\text{MS}(G, \pi)$ is meant to emphasize that once we feed π into Algorithm 1, its output is uniquely determined as we have fixed the other sources of randomization.

Having discussed our basic analysis setup, our local query process for determining whether a given vertex v is matched in M or S is formalized below as Algorithm 2. The algorithm calls two other edge subroutines formalized as Algorithms 3 and 4. Subroutine Algorithm 3 determines if a START element is matched in M by recursively calling the subroutine for incident START elements with a lower rank. Similarly, subroutine Algorithm 4 determines if an EXTEND element is matched in S by recursively calling the subroutines for incident START and EXTEND elements with lower ranks.

Let $F(v, \pi)$ denote the total number of recursive calls to the edge oracles of Algorithms 3 and 4 during the execution of $\text{VO}(v, \pi)$. The following theorem is the main result of this section.

Theorem 3.1.16. *For a randomly chosen vertex v and a random permutation π of T,*

$$\mathbf{E}_{v,\pi}[F(v, \pi)] = O(K\bar{d} \cdot \log^4 n),$$

where $\bar{d}$ is the average degree of G.

Correctness of the Oracles

In this section, we prove the correctness of the vertex oracle. Namely, we prove that:

Claim 3.1.17. *Let $v \in V$ and ST, EX be the outputs of* $\text{VO}(v, \pi)$. *It holds:*

Algorithm 2: The vertex oracle $\mathrm{VO}(u, \pi)$ which determines the matching-status of u in the outputs M and S according to permutation π.

Let $\ell_1 = (e_1, X_1), \ldots, \ell_r = (e_r, X_r)$ be the elements in T such that $e_i = (u, v_i)$, and $\pi(\ell_1) < \ldots < \pi(\ell_r)$.
$ST \leftarrow$ False, $EX \leftarrow$ False.
for i *in* $1 \ldots r$ **do**
  if $X_i =$ Extend and $c_u = c_{v_i}$ **then continue**.
  if $ST =$ False *and* $X_i =$ Start *and* $\mathrm{EOS}(\ell_i, v_i, \pi) =$ True **then**
    $ST \leftarrow$ True
  if $EX =$ False *and* $X_i =$ Extend *and* $\mathrm{EOE}(\ell_i, v_i, \pi, ST) =$ True **then**
    $EX \leftarrow$ True
return ST, EX.

Algorithm 3: The edge oracle $\mathrm{EOS}(\ell = (e, \textsc{Start}), u, \pi)$ for Start elements. Here u in the input must be an endpoint of e.

if $\mathrm{EOS}((e, \textsc{Start}), u, \pi)$ is already computed **then return** the computed answer.
Let $\ell_1 = (e_1, X_1), \ldots, \ell_r = (e_r, X_r)$ be the elements in T such that $e_i = (u, v_i)$, $X_i =$ Start for all i, and $\pi(\ell_1) < \ldots < \pi(\ell_r) < \pi(\ell)$.
for i *in* $1 \ldots r$ **do**
  if $\mathrm{EOS}(\ell_i, v_i, \pi) =$ True **then return** False.
return True.

Algorithm 4: The edge oracle $\mathrm{EOE}(\ell = (e, \textsc{Extend}), u, \pi, ST_w)$ for Extend elements. Here $e = (u, w)$ and ST_w indicates whether vertex w is matched by M in $\mathrm{MS}(G, \pi)$ through an element of rank smaller than $\pi(\ell)$.

if $\mathrm{EOE}((e, \textsc{Extend}), u, \pi, ST_w)$ is already computed **then return** the computed answer.
Let $\ell_1 = (e_1, X_1), \ldots, \ell_r = (e_r, X_r)$ be the elements in T such that $e_i = (u, v_i)$, and $\pi(\ell_1) < \ldots < \pi(\ell_r) < \pi(\ell)$.
$ST_u \leftarrow$ False
for i *in* $1 \ldots r$ **do**
  if $X_i =$ Extend and $c_u = c_{v_i}$ **then continue**.
  if $ST_u =$ False, $X_i =$ Start, *and* $\mathrm{EOS}(\ell_i, v_i, \pi) =$ True **then**
    $ST_u \leftarrow$ True
    if ℓ_i is frozen **then return** False.
    if $ST_w =$ True **then return** False.
  if $X_i =$ Extend and $\mathrm{EOE}(l_i, v_i, \pi, ST_u) =$ True **then return** False.
return True.

- $ST =$ True *iff* v *has an incident* Start *element in* $\mathrm{MS}(G, \pi)$.
- $EX =$ True *iff* v *has an incident* Extend *element in* $\mathrm{MS}(G, \pi)$.

Let us first prove two auxiliary claims about the correctness of the two edge oracles.

Claim 3.1.18. *For any* $\ell = ((u, w), \textsc{Start})$, *if* $\mathrm{EOS}(\ell, u, \pi)$ *is called during computing* $\mathrm{VO}(v, \pi)$, *then* $\mathrm{EOS}(\ell, u, \pi) =$ True *iff* $\ell \in \mathrm{MS}(G, \pi)$.

Proof. We prove Claim 3.1.18 by induction on $\pi(\ell)$. Suppose that the statement holds for all Start elements with a ranking lower than $\pi(\ell)$. If $\mathrm{VO}(v, \pi)$ directly calls $\mathrm{EOS}(\ell, u, \pi)$, it had already called $\mathrm{EOS}(\ell'', u, \pi)$

for every START elements ℓ'' incident on u with a lower ranking. Moreover, if $\text{EOS}(\ell, u, \pi)$ is called by $\text{EOS}(\ell', w, \pi)$ or $\text{EOE}(\ell', w, \pi, \cdot)$, then all START elements on edges (w, u') with a smaller rank must be queried before ℓ by the description of oracles. All these calls to the edge oracle, EOS, must return FALSE since the edge oracle queries $\text{EOS}(\ell, u, \pi)$. By the induction hypothesis, there is no START element incident to w with a smaller rank in $\text{MS}(G, \pi)$. Also, $\text{EOS}(\ell, u, \pi)$ queries all the START elements incident to u with lower rank and returns TRUE if none of them is in $\text{MS}(G, \pi)$. By the induction hypothesis, these calls are answered correctly, completing the proof. □

Claim 3.1.19. *Let $\ell = ((u, w), \text{EXTEND})$, and let ST_w indicate if w has a* START *element incident to it in* $\text{MS}(G, \pi)$ *with a rank smaller than $\pi(\ell)$. If $\text{EOE}(\ell, u, \pi, ST_w)$ is called during computing $\text{VO}(v, \pi)$, then $\text{EOE}(\ell, u, \pi, ST_w) =$* TRUE *iff $\ell \in \text{MS}(G, \pi)$.*

Proof. We prove the statement by induction on $\pi(\ell)$. With a similar argument as in the proof of Claim 3.1.18, we can show that EXTEND elements incident to w with lower ranks are queried before ℓ, and that the results of all these calls are FALSE. Also, if there exists a START element incident to w with a lower rank in $\text{MS}(G, \pi)$, then by the description of EOE, it is queried before and ST_w is TRUE. Note that if such an edge exists, it must not be frozen, otherwise, the oracle does not query $\text{EOE}(\ell, u, \pi, ST_w)$. Hence, if $\ell \notin \text{MS}(G, \pi)$, there must be a frozen or an EXTEND element incident to u, or $ST_w =$ TRUE and there exists a START element incident to u. Since $\text{EOE}(\ell, u, \pi, ST_w)$ queries all the edges incident to u with lower rank to determines if a START or EXTEND element exists in $\text{MS}(G, \pi)$, the proof is complete by induction hypothesis. □

We are now ready to prove Claim 3.1.17.

Proof of Claim 3.1.17. Since the vertex oracle queries the edge oracle EOS for all incident START elements in increasing order of their ranking until it finds a START element in $\text{MS}(G, \pi)$, by Claim 3.1.18, the first property holds. Since Algorithm 2 and Algorithm 4 query incident edges in increasing order, if $\text{EOE}(\ell, u, \pi, ST_w)$ is queried for an EXTEND element $\ell = ((u, w), \text{EXTEND})$, ST_w is computed correctly before calling $\text{EOE}(\ell, u, \pi, ST_w)$ which implies that the condition in Claim 3.1.19 holds. Therefore, with the similar argument for EXTEND elements and correctness of edge oracle EOE by Claim 3.1.19, the second property holds. □

Query Complexity of the Oracles

In this section, we prove Theorem 3.1.16. Consider $\ell = (e, X)$, an element in T. If $X =$ START, we write $Q(\ell, v, \pi)$ to denote the total number of recursive calls to $\text{EOS}(\ell, \cdot, \pi, \cdot)$ during the execution of $\text{VO}(v, \pi)$. If $X =$ EXTEND, we write $Q(\ell, v, \pi)$ to denote the total number of recursive calls to $\text{EOE}(\ell, \cdot, \pi, \cdot)$ during the execution of $\text{VO}(v, \pi)$.

Observation 3.1.20. *For a permutation π, at most one* START *copy of every edge is queried in recursive calls of* EOS.

Proof. For a fixed edge, the edge oracle on its START copies only generates a new recursive call on the first call because of the caching in Algorithm 3 of Algorithm 3. □

Observation 3.1.21. *For every element $\ell = (e, X)$ in T and permutation π, $Q(\ell, \pi) \leq O(n^2)$.*

Proof. Let $e = \{x, y\}$. For a fixed vertex u, either the vertex oracle $\text{VO}(u, \pi)$ queries the edge oracle for ℓ directly, or through some incident elemenet ℓ'. Note that by Observation 3.1.20, for each edge e' incident to e, at most one of its START copies generates a new recursive call. Hence, the edge oracle of ℓ is called through at most $2(\deg(x) - 1) + 2(\deg(y) - 1)$ of its incident edges (one of its START copy and one EXTEND element) which implies that there are at most $4(n - 1) + 1$ calls during the course of $\text{VO}(u, \pi)$. In other words, we have that $Q(\ell, u, \pi) \leq 4n - 3$. Therefore,

$$Q(\ell, \pi) \leq \sum_{u \in V} Q(\ell, u, \pi) \leq n(4n - 3) \leq O(n^2). \quad \square$$

The main contribution of this section is to show that the expected $Q(\ell, \pi)$ for a random permutation π is $O(\log^4 n)$ which is formalized in the following lemma.

Lemma 3.1.22. *For any element $\ell \in T$, we have $\mathbf{E}_\pi[Q(\ell, \pi)] = O(\log^4 n)$.*

Assuming the correctness of Lemma 3.1.22, we can complete the proof of Theorem 3.1.16.

Proof of Theorem 3.1.16.

$$\begin{aligned}
\mathbf{E}_{v,\pi}[F(v, \pi)] = \frac{1}{n} \mathbf{E}_\pi \left[\sum_{v \in V} F(v, \pi) \right] &= \frac{1}{n} \mathbf{E}_\pi \left[\sum_{v \in V} \sum_{\ell \in T} Q(\ell, v, \pi) \right] \\
&= \frac{1}{n} \mathbf{E}_\pi \left[\sum_{\ell \in T} \sum_{v \in V} Q(\ell, v, \pi) \right] = \frac{1}{n} \mathbf{E}_\pi \left[\sum_{\ell \in T} Q(\ell, \pi) \right] \\
&= \frac{1}{n} \sum_{\ell \in T} \mathbf{E}_\pi[Q(\ell, \pi)] = \frac{1}{n} \sum_{\ell \in T} O(\log^4 n) \\
&= \frac{1}{n} O(Km \cdot \log^4 n) = O(K\bar{d} \cdot \log^4 n). \quad \square
\end{aligned}$$

During the recursive calls to the edge oracles that start from $\text{VO}(v, \pi)$, edges corresponding to the elements in the stack of recursive calls create a path. Because, we query $\text{VO}(v, \pi)$ at the beginning, one of the endpoints of this path is vertex v. We direct the elements in the path away from v towards the other endpoint. We call such a directed path starting from the bottom of the stack all the way to the top, a (v, π)-query-path. For an edge $e = (x, y)$, let $\vec{e}$ denote the directed edge from x to y and $\overleftarrow{e}$ denote a directed edge from y to x. Similarly, for an element $\ell = ((x, y), X)$, $\vec{\ell}$ and $\overleftarrow{\ell}$ represent the direction of e.

Let $Q(\vec{\ell}, \pi) \subseteq Q(\ell, \pi)$ be the set of all query paths that end at $\vec{\ell}$ (with the same direction). In the following lemma, we bound the query complexity based on the direction of ℓ. We use this lemma to prove Lemma 3.1.22.

Lemma 3.1.23. *For any element $\ell \in T$, we have $\mathbf{E}_\pi[Q(\vec{\ell}, \pi)] = O(\log^4 n)$.*

Proof of Lemma 3.1.22. Since $Q(\ell, \pi) = Q(\vec{\ell}, \pi) \cup Q(\overleftarrow{\ell}, \pi)$, by Lemma 3.1.23 we have

$$\mathbf{E}_\pi[Q(\ell, \pi)] \leq \mathbf{E}_\pi[Q(\vec{\ell}, \pi)] + \mathbf{E}_\pi[Q(\overleftarrow{\ell}, \pi)] = O(\log^4 n) + O(\log^4 n) = O(\log^4 n). \quad \square$$

In the rest of this section, we focus on proving Lemma 3.1.23. The first helpful observation is that all the EXTEND elements in a (v, π)-query-path appear before the START elements.

Observation 3.1.24. *Let $\vec{P} = (\vec{\ell_r}, \dots, \vec{\ell_1})$ be a (v, π)-query-path and $\vec{\ell_i} = (\vec{e_i}, X_i)$ for all i. Then there exists a j $(0 \le j \le r)$ such that $X_i = \textsc{Start}$ for $0 < i \le j$, and $X_i = \textsc{Extend}$ for $j < i \le r$.*

Proof. Let j be the maximum index such that $X_j = \textsc{Start}$. If there is no such index, the proof is complete since we can assume that $j = 0$. Now we claim that for all $i < j$, we have $X_i = \textsc{Start}$. This can be verified by noting that Algorithm 3 only queries the edge oracle for Start elements. □

Given a permutation π and a path of elements $\vec{P} = (\vec{\ell_r}, \dots, \vec{\ell_1})$, we define $\phi(\pi, \vec{P})$ to be another permutation σ over edges such that:

$$(\sigma(\ell_1), \sigma(\ell_2), \dots, \sigma(\ell_{r-1}), \sigma(\ell_r)) \coloneqq (\pi(\ell_2), \pi(\ell_3), \dots, \pi(\ell_r), \pi(\ell_1))$$
$$\pi(\ell') = \sigma(\ell') \qquad \forall \ell' \notin \vec{P}.$$

Given an element $\vec{\ell}$, by using the above mapping function we can construct a bipartite graph H with two parts A and B such that each part has $((K+1)m)!$ vertices showing different permutations of elements. For a permutation $\pi \in A$ and a (v, π)-query-path $\vec{P}$ that ends at ℓ for some arbitrary vertex v, we connect π in A to $\phi(\pi, \vec{P})$ in B. Note that by the construction of H, $\deg(\pi_A) = Q(\vec{\ell}, \pi_A)$ for all $\pi_A \in A$, since we have a unique element for each query-path that ends at $\vec{\ell}$ with permutation π_A. Hence, in order to prove Lemma 3.1.22, it is sufficient to prove that $\mathbf{E}_{\pi_A \sim A}[\deg_H(\pi_A)] = O(\log^4 n)$.

Let $\mathcal{Q}(\vec{\ell}, \pi)$ be the set of all query-paths for permutation π that ends at $\vec{\ell}$. Let $\beta = c \log^2 n$ for some large constant c and Π be the set of all possible $((K+1)m)!$ permutations. We partition Π into two subsets of *likely* and *unlikely* permutations, denoted by L and U, respectively, as follows:

$$L \coloneqq \left\{ \pi \in \Pi \;\middle|\; \max_{\vec{P} \in \mathcal{Q}(\vec{\ell}, \pi)} |\vec{P}| \le \beta \right\} \qquad U \coloneqq \Pi \setminus L.$$

In words, likely permutations are those where the longest query-path ending at $\vec{\ell}$ has a length of at most β and unlikely permutations are the remaining ones. Let A_L be the set of vertices corresponding to the likely permutations in A and A_U be the set of vertices corresponding to unlikely permutations. The intuition behind this partitioning is that the set of unlikely permutations makes up a tiny fraction of all permutations which is formalized in Lemma 3.1.25.

Lemma 3.1.25. *If c is a large enough constant, then we have $|A_U| \le ((K+1)m)!/n^2$.*

Furthermore, we show that each vertex $\pi_B \in B$ has at most $O(\beta^2)$ neighbors among the likely permutations in part A of our bipartite graph H.

Lemma 3.1.26. *Let π_B be a vertex in B. Then π_B has at most β^2 neighbors in A_L.*

Proof of Lemma 3.1.23. We provide an upper bound on $|E(H)|$. First, note that by Lemma 3.1.25, we have $|A_U| \le ((K+1)m)!/n^2$. Furthermore, by Observation 3.1.21, degree of each vertex $\pi_A \in A$ is at most $O(n^2)$ which implies that the total number of edges incident to vertices of A_U is at most

$$E(A_U, B) \le ((K+1)m)!/n^2 \cdot O(n^2) \le O\big(((K+1)m)!\big).$$

Moreover, by Lemma 3.1.26, each vertex $\pi_B \in B$ has at most $O(\beta^2)$ neighbors in A_L. Since H is a bipartite graph, total number of incident edges to A_L is at most $O(\beta^2) \cdot |A_L|$. Therefore, sum of degrees of

all vertices in A is at most

$$O(\beta^2) \cdot |A_L| + E(A_U, B) \leq O(\beta^2 \cdot ((K+1)m)!).$$

For a random vertex in A, the expected degree is $\frac{O(\beta^2 \cdot ((K+1)m)!)}{((K+1)m)!} = O(\beta^2)$, which completes the proof since $\beta = c \log^2 n$ and $\deg(\pi_A) = Q(\vec{\ell}, \pi_A)$. □

In the following two subsections, we prove Lemmas 3.1.25 and 3.1.26.

Proof of Lemma 3.1.26

This proof is the most technical part of this part of the analysis. We show that for two query-paths $\vec{P}$ and $\vec{P}'$ that end at $\vec{\ell}$, and two permutations π and π', if $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P}')$, then either one of the query paths is a subpath of the other one, or they "branch" through a specific configuration of START and EXTEND elements illustrated in Figure 3.2.

To formalize this, let us formally define what a "branch" is:

Definition 3.1.27. *Let $\vec{P} = (\vec{\ell}_{r_1}, \ldots, \vec{\ell}_1)$ and $\vec{P}' = (\vec{\ell}'_{r_2}, \ldots, \vec{\ell}'_1)$ be two query-paths. Let j be an such that $\vec{\ell}_i = \vec{\ell}'_i$ for all $i \leq j < \min(r_1, r_2)$, but $\vec{\ell}_{j+1} \neq \vec{\ell}'_{j+1}$. Then $\vec{P}$ and $\vec{P}'$* branch *at element $\vec{\ell}_j$.*

And now let us define *valid* and *invalid* branches (see Figure 3.2).

Definition 3.1.28. *Let $\vec{P} = (\vec{\ell}_{r_1}, \ldots, \vec{\ell}_1)$ and $\vec{P}' = (\vec{\ell}'_{r_2}, \ldots, \vec{\ell}'_1)$ be two query-paths, where $\vec{\ell}_i = (\vec{e}_i, X_i)$, and $\vec{\ell}'_i = (\vec{e}'_i, X'_i)$. Assume that $\vec{P}$ and $\vec{P}'$ branch at element $\vec{\ell}_j$ for $j < \min(r_1, r_2)$. Moreover, let u be the shared vertex between $\vec{e}_j$ and $\vec{e}_{j+1}$. We call this branch* valid *if $X_j = \text{START}$ and $\{X_{j+1}, X'_{j+1}\} = \{\text{EXTEND}, \text{START}\}$. For all other possible configurations of X_j, X_{j+1}, and X'_{j+1}, we say the branch is* invalid.

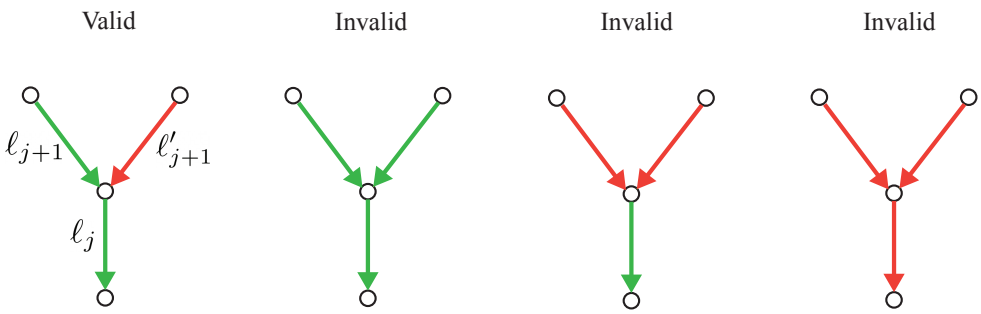


Figure 3.2: All valid and invalid branches. Green edges represent START elements and red edges represent EXTEND elements.

One key property of invalid branches is formalized in the following observation which is helpful in the rest of this section.

Observation 3.1.29. *Let $\vec{P} = (\vec{\ell}_{r_1}, \ldots, \vec{\ell}_1)$ and $\vec{P}' = (\vec{\ell}'_{r_2}, \ldots, \vec{\ell}'_1)$ be two query-paths, $\vec{\ell}_i = (\vec{e}_i, X_i)$ and $\vec{\ell}'_i = (\vec{e}'_i, X'_i)$ for all i. If there exists an invalid branch at $\vec{\ell}_j$, then none of the tuples (X_j, X_{j+1}), (X_j, X'_{j+1}), (X_{j+1}, X'_{j+1}), and (X'_{j+1}, X_{j+1}) is $(\text{EXTEND}, \text{START})$.*

Proof. Let u be the shared endpoint between $\vec{e}_j$, $\vec{e}_{j+1}$, and $\vec{e'}_{j+1}$. If $X_j = \textsc{Extend}$, by Observation 3.1.24, we have $X_{j+1} = X'_{j+1} = \textsc{Extend}$. Therefore, all tuples are $(\textsc{Extend}, \textsc{Extend})$.

Now assume that $X_j = \textsc{Start}$. If one of X_{j+1} or X'_{j+1} is $\textsc{Extend}$ and the other one is $\textsc{Start}$, then the branch is valid by Definition 3.1.28. This leaves two remaining scenarios:

- $X_{j+1} = X'_{j+1} = \textsc{Start}$: In this case, all tuples are $(\textsc{Start}, \textsc{Start})$.
- $X_{j+1} = X'_{j+1} = \textsc{Extend}$: In this case, $(X_{j+1}, X'_{j+1}) = (X'_{j+1}, X_{j+1}) = (\textsc{Extend}, \textsc{Extend})$. Moreover, $(X_j, X_{j+1}) = (X_j, X'_{j+1}) = (\textsc{Start}, \textsc{Extend})$. Thus there is no $(\textsc{Extend}, \textsc{Start})$ among the tuples considered in the statement.

The proof is thus complete. □

Next, we show that for two query-paths $\vec{P}$ and $\vec{P'}$ that end at $\vec{\ell}$, and two permutations π and π', if $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P'})$, then either the shorter query-path is a subpath of the longer one or they have exactly one valid branch. In particular, it is not possible for $\vec{P}$ and $\vec{P'}$ to have an invalid branch or have multiple valid branches under this condition.

Lemma 3.1.30. *Let π and π' be two different permutations over T, and let $\vec{P}$ and $\vec{P'}$ be (v, π)- and (v', π')-query-path, respectively, that both end at element $\vec{\ell}$. If $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P'})$, then $\vec{P}$ and $\vec{P'}$ branch at most once and this branch is valid.*

Before proving Lemma 3.1.30, we prove a sequence of auxiliary observations and claims. Let $\vec{P} = (\vec{\ell}_{r_1}, \ldots, \vec{\ell}_1)$ and $\vec{P'} = (\vec{\ell'}_{r_2}, \ldots, \vec{\ell'}_1)$ where $\vec{\ell} = \vec{\ell}_1 = \vec{\ell'}_1$. Also, let $\vec{\ell}_i = (\vec{e}_i, X_i)$ and $\vec{\ell'}_i = (\vec{e_i}', X'_i)$. For the sake of contradiction, suppose that $\vec{P}$ and $\vec{P'}$ branch at element $\vec{\ell}_b$ and the branch is invalid. This means that $\vec{\ell}_i = \vec{\ell'}_i$ for $i \leq b$ and $\vec{\ell}_{b+1} \neq \vec{\ell'}_{b+1}$. Note that $\vec{e}_{b+1}$ and $\vec{e'}_{b+1}$ can be the same edge. We do not need to separately investigate these two cases as our proof generally works for both cases.

Observation 3.1.31. *We have that $\pi(\ell_1) < \pi(\ell_2) < \ldots < \pi(\ell_{r_1})$ and $\pi'(\ell'_1) < \pi'(\ell'_2) < \ldots < \pi'(\ell'_{r_2})$.*

Proof. Algorithms 3 and 4 recursively call on elements with π values less than π value of the current element. Therefore, the stack of recursive calls will be decreasing with respect to π values. The same condition also holds for permutation π'. □

Observation 3.1.32. $\pi(\ell_b) = \pi'(\ell_{b+1})$.

Proof. Since $\vec{\ell}_{b+1}$ is not in $\vec{P'}$, we have that $\phi(\pi', \vec{P'})(\ell_{b+1}) = \pi'(\ell_{b+1})$. Also, $\phi(\pi, \vec{P})(\ell_{b+1}) = \pi(\ell_b)$ since $\phi(\pi, \vec{P})$ shifts the elements of path $\vec{P}$ by one. Given that $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P'})$, we get $\pi(\ell_b) = \pi'(\ell_{b+1})$. □

In the rest of the proof, we assume that $\pi(\ell_b) < \pi'(\ell_b)$. This is without loss of generality because we have not distinguished the two permutations π and π' in any other way.

Observation 3.1.33. $\pi'(\ell_{b+1}) < \pi'(\ell_b)$.

Proof. By combining Observation 3.1.32, our assumption that $\pi(\ell_b) < \pi'(\ell_b)$, and the fact that π' is a permutation, we have that $\pi'(\ell_{b+1}) < \pi'(\ell_b)$. □

Claim 3.1.34. *If $\pi(f) < \pi(\ell_b)$ or $\pi'(f) < \pi(\ell_b)$ for some element f, then $\pi(f) = \pi'(f)$.*

Proof. There are five different possible cases for f:

- $f \notin P \cup P'$: Since ϕ only changes the rank of elements on the query-path and $\phi(\pi, \vec{P})(f) = \phi(\pi', \vec{P'})(f)$, we have $\pi(f) = \pi'(f)$.
- $f \in \{\ell_1, \ldots, \ell_{b-1}\}$: Since $\phi(\pi, \vec{P})(\ell_{i+1}) = \phi(\pi', \vec{P'})(\ell_{i+1})$ for $1 \leq i < b$, we have $\pi(\ell_i) = \pi'(\ell_i)$. Hence, $\pi(f) = \pi'(f)$.
- $f = \ell_b$: In this case, condition $\pi(f) < \pi(\ell_b)$ does not hold since $\pi(f) = \pi(\ell_b)$. Also, $\pi'(f) = \pi'(\ell_b) > \pi(\ell_b)$ by our assumption. Therefore, condition $\pi'(f) < \pi(\ell_b)$ does not hold.
- $f \in \{\ell_{b+1}, \ldots, \ell_{r_1}\}$: By Observation 3.1.31, we have $\pi(f) > \pi(\ell_b)$. Therefore, condition $\pi(f) < \pi(\ell_b)$ does not hold. Let $f = \ell_i$ for $i > b$. Since $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P'})$, we have that $\pi'(f) = \pi(\ell_{i-1}) \geq \pi(\ell_b)$. Therefore, none of the conditions in the claim statement hold.
- $f \in \{\ell'_{b+1}, \ldots, \ell'_{r_2}\}$: By Observation 3.1.31, we have $\pi'(f) > \pi'(\ell_b)$. Combined with our assumption $\pi'(\ell_b) > \pi(\ell_b)$, this gives $\pi'(f) > \pi(\ell_b)$. Let $f = \ell'_i$ for $i > b$. Since $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P'})$, we have $\pi(f) = \pi'(\ell_{i-1}) \geq \pi'(\ell_b) > \pi(\ell_b)$. Therefore, none of the conditions in the claim statement hold.

Each of the cases above either contradicts a condition of the claim, or satisfies $\pi(f) = \pi'(f)$. The proof is thus complete. □

Observation 3.1.35. *Let f be a* START *element such that $\pi(f) = \pi'(f)$. Then f is frozen in permutation π iff it is frozen in permutation π'.*

Proof. Since we fix the color of vertices, Bernoulli random variables in Algorithm 1 of Algorithm 1, and the index j^* that is used in Algorithm 1 for both permutations π and π' over T, the only way that one of the f and f' is frozen and the other one is not, is when $\pi(f) \neq \pi(f')$. □

Claim 3.1.36. $\ell_{b+1} \in \text{MS}(G, \pi')$.

Proof. We prove the claim by contradiction. Assume that $\ell_{b+1} \notin \text{MS}(G, \pi')$. There are two possible scenarios for ℓ_{b+1} not to be in $\text{MS}(G, \pi')$:

- $X_{b+1} =$ EXTEND and $\ell_{b+1} \notin \text{MS}(G, \pi')$ because of two START elements $f, f' \in \text{MS}(G, \pi')$:

  Note that $\pi'(f) < \pi'(\ell_{b+1})$ and $\pi'(f') < \pi'(\ell_{b+1})$ since $f, f' \in \text{MS}(G, \pi')$ and $\ell_{b+1} \notin \text{MS}(G, \pi')$. On the other hand, by Observation 3.1.32, we have that $\pi(\ell_b) = \pi'(\ell_{b+1})$. Thus, $\pi'(f) < \pi(\ell_b)$ and $\pi'(f') < \pi(\ell_b)$. Hence, by Claim 3.1.34, $\pi(f) = \pi'(f)$ and $\pi(f') = \pi'(f')$, which implies that $f, f' \in \text{MS}(G, \pi)$ since π and π' are identical for ranks smaller than $\pi(\ell_b)$. Moreover, by Observation 3.1.35, each of f and f' are either frozen in both π and π' or not. Also, both f and f' are not in path P since $\pi(f) < \pi(\ell_b)$ and $\pi(f') < \pi(\ell_b)$. Let $e_{b+1} = (w, y)$ where y is the shared endpoint with e_b. Without loss of generality, assume that f is incident to w and f' is incident to y. Note that, both of f and f' cannot be incident to the same endpoint of e_{b+1} since START elements create a maximal matching. Since both edge oracle and vertex oracle queries edges in increasing order, when $\text{EOE}(\ell_{b+1}, y, \pi, ST_w)$ was called, ST_w must be TRUE since the edge oracle already queried element f before. Furthermore, $\text{EOE}(\ell_{b+1}, y, \pi, ST_w)$ calls edge oracle for f' before ℓ_b since $\pi'(f) < \pi(\ell_b)$. This implies that (v, π)-query-path $\vec{P}$ is not a valid (v, π)-query-path since the $\text{EOE}(\ell_{b+1}, y, \pi, ST_w)$ terminates after calling the edge oracle for element f' (see Figure 3.3).

- $\ell_{b+1} \notin \text{MS}(G, \pi')$ because of a single element $f \in \text{MS}(G, \pi')$:

  Let y be the shared endpoint of e_b and e_{b+1}. With the same argument as in the previous case, we get that $f \in \text{MS}(G, \pi)$ and $\pi(f) < \pi(\ell_b)$. Thus, by Observation 3.1.35, element f is either frozen in both π and π' or in neither one. Note that either both of f and ℓ_{b+1} are START elements, or f is a frozen element, or both of f and ℓ_{b+1} are EXTEND elements. Hence, by Observation 3.1.24, f must be queried before ℓ_{b+1} if it is not incident to y in the edge oracle for permutation π which implies that the edge oracle will not query ℓ_{b+1}. Furthermore, if f is incident to y, edge oracle queries f before ℓ_b which implies that it terminates and (v, π)-query-path $\vec{P}$ is not a valid (v, π)-query-path. □

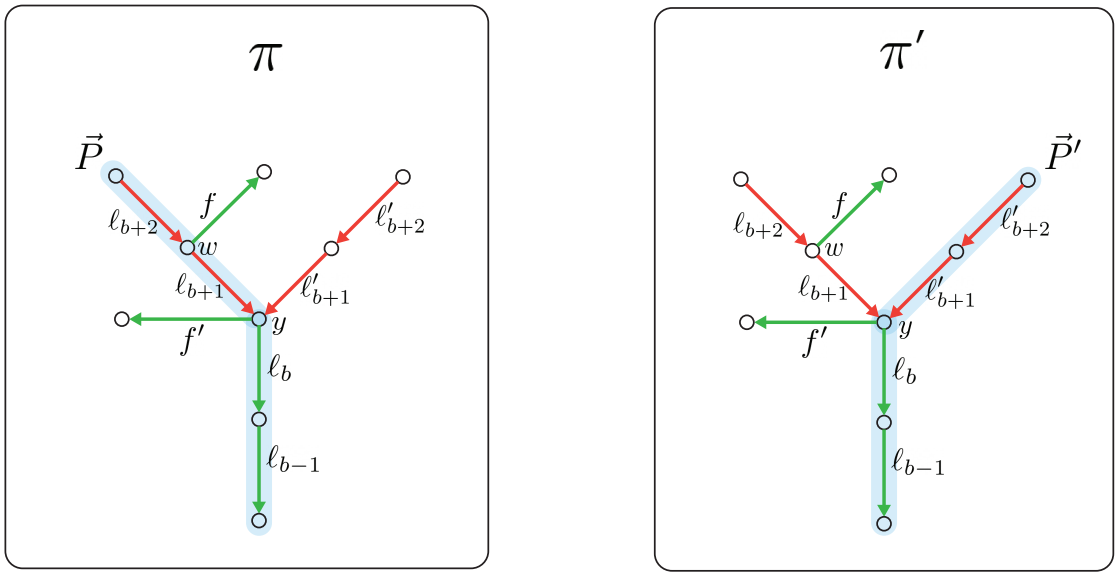


Figure 3.3: Illustration of a possible case of the first scenario in the proof of Claim 3.1.36. Green edges represent START elements and red edges represent EXTEND elements. The left figure shows a query-path $\vec{P}$ in permutation π which is highlighted in light blue. Similarly, the right figure shows a query-path $\vec{P}'$ in permutation π'. Query-path $\vec{P}$ is not a valid query-path since the $\text{EOE}(\ell_{b+1}, y, \pi, ST_w)$ terminates after calling the edge oracle on element f'.

Proof of Lemma 3.1.30. Assume for the sake of contradiction that the branch at $\vec{\ell}_b$ is an invalid branch. We prove that query-path $\vec{P}'$ is not a valid (v, π')-query-path. By Definition 3.1.28, we have that $X_{b+1} = X'_{b+1}$. Also, by Claim 3.1.36, edge oracle of ℓ'_{b+1} for permutation π' queries ℓ_{b+1} before ℓ_b since $\pi'(\ell_{b+1}) < \pi'(\ell_b)$ by Observation 3.1.33. Thus, the edge oracle for ℓ'_{b+1} terminates at this point. Therefore, P and P' contains no invalid branch. □

Now we are ready to complete the proof of Lemma 3.1.26.

Proof of Lemma 3.1.26. As above, consider two query paths $\vec{P} = (\vec{\ell}_{r_1}, \ldots, \vec{\ell}_1)$ and $\vec{P}' = (\vec{\ell}'_{r_2}, \ldots, \vec{\ell}'_1)$ that end at $\vec{\ell} = \vec{\ell}_1 = \vec{\ell}'_1$ and suppose that $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P}')$.

If ℓ is an EXTEND element, then all the elements in the two query paths $\vec{P}$ and $\vec{P}'$ must be EXTEND by Observation 3.1.24. Therefore, $\vec{P}$ and $\vec{P}'$ cannot have a valid branch since a valid branch, by Definition 3.1.28, requires two START elements. Since Lemma 3.1.30 asserts $\vec{P}$ and $\vec{P}'$ cannot have invalid branches either,

one of P and P' must be a subpath of the other. Note that all paths that end at ℓ must be a subpath of the longest query path since we do not have a branch which implies that all paths must have different lengths. Therefore, there are at most β query-paths that end at $\vec{\ell}$ since the length of the longest path is at most β.

Now suppose that ℓ is a START element. Assume that $\vec{P}$ and $\vec{P}'$ branch at $\vec{\ell}_b$. Since this cannot be an invalid branch by Lemma 3.1.30, without a loss of generality, assume that ℓ'_{b+1} is an EXTEND element. By Observation 3.1.24, all $\ell'_{b+1}, \ell'_{b+2}, \ldots, \ell'_{r_2}$ must be EXTEND elements. Let $\mathcal{P}$ be the set of all query-paths that end at $\vec{\ell}$, and $\vec{P}_s \in \mathcal{P}$ be the path with maximum number of START elements (break the tie with the longest length of EXTEND elements of the path). Note that there is no other START element in the paths of $\mathcal{P}$ other than START elements of P_s. Otherwise, we have an invalid branch since START elements appear before EXTEND elements by Observation 3.1.24.

Let $(\vec{\ell}_s, \ldots, \vec{\ell}_1)$ be the subpath of $\vec{P}_s$ consisting of START elements. We claim that for each $\vec{\ell}_i$ $(i < s)$, if there exist two query-paths in $\mathcal{P}$ that branch with $\vec{P}_s$ at $\vec{\ell}_i$, then one of them must be a subpath of the other. To see this, assume that there are two paths $\vec{P}$ and $\vec{P}'$ such that they both have a branch with $\vec{P}_s$ at $\vec{\ell}_i$. Let $\vec{f}$ and $\vec{f}'$ be the next elements on $\vec{P}$ and $\vec{P}'$. Note that it is possible that $\vec{f} = \vec{f}'$. By Definition 3.1.28, f and f' are EXTEND elements. Hence, the first elements that are not shared in both $\vec{P}$ and $\vec{P}'$ are EXTEND elements which means $\vec{P}$ and $\vec{P}'$ has an invalid branch.

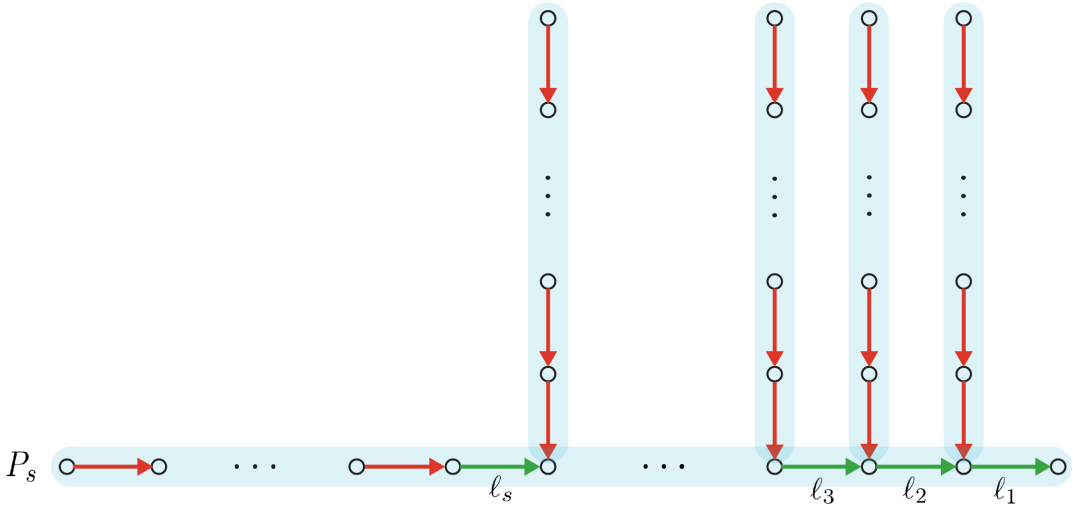


Figure 3.4: Illustration of $\hat{\mathcal{P}}$ and P_s. The green edges represent START elements and the red edges represent EXTEND elements. Paths in $\hat{\mathcal{P}}$ are highlighted in light blue.

On the other hand, note that there is no path in $\mathcal{P}$ that has a branch with $\vec{P}_s$ in EXTEND elements of $\vec{P}_s$ since there is no invalid branch by Lemma 3.1.30. Let $\hat{\mathcal{P}}$ be the union of $\vec{P}_s$ and the set of paths in $\mathcal{P}$ such that they have a valid branch with $\vec{P}_s$, and they are not a subgraph of other paths in $\mathcal{P}$ (see Figure 3.4).

By the above claim, $|\hat{\mathcal{P}}| \leq s+1$. To complete the proof, assume that for each edge between $\pi_A \in A_L$ and $\pi_B \in B$ in graph H, we write a label $(\vec{P}', |\vec{P}|)$ where $\vec{P}$ is the query-path corresponding to the edge between π_B and π_A, and $\vec{P}' \in \hat{\mathcal{P}}$ such that $\vec{P} \subseteq \vec{P}'$ (if there are multiple choices for $\vec{P}'$, choose the longest path). Since all labels must be different, and by definition of A_L we have that all query-paths have a length of at most β, there are at most $(s+1)\beta \leq 2\beta^2$ different labels, where the last inequality followed by the fact that $|\vec{P}_s| \leq \beta$. □

Proof of Lemma 3.1.25

In this section, we prove that it is very unlikely to have long query-paths during the recursive calls of edge oracle. Our proof is inspired by [61], who proved that the parallel round complexity of greedy maximal independent set is $O(\log^2 n)$. Our arguments are slightly different because our algorithm is not an instance of greedy MIS.

We define a θ-prefix of permutation π over elements T to be the first $\theta|T|$ elements in permutation π. Suppose that instead of running the edge oracle on the whole permutation of elements, we choose a θ-prefix of elements and run the edge oracle for elements of the prefix. The first useful observation is that if the algorithm calls edge oracle for one of the elements in a θ-prefix of T, all recursive calls during this call will be on elements in the prefix.

Observation 3.1.37. *Let ℓ be an element in T and π be a permutation over T. Then* $\text{EOS}(\ell, \cdot, \pi)$ *or* $\text{EOE}(\ell, \cdot, \pi, \cdot)$ *only recursively calls edge oracle for elements with a lower rank.*

Proof. Proof can be easily obtained by how we defined Algorithm 3 and Algorithm 4 since each element only recursively calls elements with a lower rank. □

The high-level approach for this section is that we partition elements of T to $O(\log n)$ continuous ranges and show that the length of the longest query-path inside each of these ranges is $O(\log n)$. Therefore, since for each element the edge oracle only calls elements with a lower rank, the length of the longest path in total should be at most $O(\log^2 n)$.

Element Partitioning: Let $\mathcal{P}_1$ be a θ_1-prefix of T. Suppose that we run Algorithm 1 on $\mathcal{P}_1$. Let $\mathcal{A}_1$ be the set of elements that are chosen by Algorithm 1 for the selected subgraph $\text{MS}(G, \pi)$. Also, let $\mathcal{D}_1$ be the set of elements outside the prefix $\mathcal{P}_1$ that cannot be in $\text{MS}(G, \pi)$ due to the existence of some elements in $\mathcal{A}_1$. We delete $\mathcal{P}_1 \cup \mathcal{D}_1$ from T and similarly choose θ_2-prefix of the remaining elements. Let $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_d$ be the partitions with parameters $\theta_1, \theta_2, \ldots, \theta_d$, and $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_d$ be sets of deleted elements as defined above. We let $\theta_i = \Omega(2^i \log(n)/(2K\Delta))$ for the i-th partition and $d = O(\log n)$.

Observation 3.1.38. *Let ℓ be an element in $\mathcal{P}_j \cup \mathcal{D}_j$ as defined above. Then* $\text{EOS}(\ell, \cdot, \pi)$ *or* $\text{EOE}(\ell, \cdot, \pi, \cdot)$ *only recursively calls edge oracle for elements in $\{\mathcal{P}_i\}_{i \le j} \cup \{\mathcal{D}_i\}_{i < j}$.*

Proof. If $\ell \in \mathcal{P}_j$, since all the elements with lower ranks are in $\{\mathcal{P}_i\}_{i \le j} \cup \{\mathcal{D}_i\}_{i < j}$, by Observation 3.1.37, the proof is complete. If $\ell \in \mathcal{D}_j$, there are some elements in $\mathcal{P}_j$ that their existence in $\text{MS}(G, \pi)$ caused ℓ not to be in $\text{MS}(G, \pi)$. Hence, since edge oracle recursively calls incident elements in their increasing ranks, edge oracles for ℓ will only query incident elements in $\{\mathcal{P}_i\}_{i \le j} \cup \{\mathcal{D}_i\}_{i < j}$. □

Note that the Start elements that appear in matching M of Algorithm 1 form a randomized greedy maximal matching, and our query process for Start elements also coincides with the query process for greedy matchings. Therefore, we can use the following result of [34] which itself builds on the bound on [91] as black-box.

Lemma 3.1.39 ([34, Lemma 3.13])**.** *Let π be a permutation over elements of T. With high probability, the maximum length of a query-path consisting of* Start *elements is $O(\log n)$.*

In the rest of this section, we show that it is unlikely to have a query-path consisting of Extend elements with length larger than $O(\log^2 n)$. Since by Observation 3.1.24 all Start elements appear after Extend

elements in a query-path, this is enough to show that the length of a query-path is bounded by $O(\log^2 n)$ with high probability.

The following two lemmas are similar to [61, Lemma 3.1 and Lemma 3.3], however, both are adapted to our setting.

Lemma 3.1.40. *Suppose that we choose a θ-prefix of a uniformly at random permutation π of elements of T, where $\theta = a/b$ for positive numbers $a \leq b \leq |T|$ such that $a/b \geq 2/|T|$. Let $\mathrm{MS}_\theta(G, \pi)$ be the subgraph produced by Algorithm 1 on this prefix. Assume that we remove all other elements outside the prefix that cannot join subgraph $\mathrm{MS}(G, \pi)$ based on the current elements in $\mathrm{MS}_\theta(G, \pi)$. All remaining* Extend *elements have at most b incident elements with probability of at least $1 - \frac{2|T|^2}{e^{a/2}}$.*

Proof. We say an element is *live* if we can add it to subgraph $\mathrm{MS}_\theta(G, \pi)$ and *dead* otherwise. Assume that we sequentially pick $(a|T|/b)$ elements. If the chosen element is live, we add the element to $\mathrm{MS}_\theta(G, \pi)$, and mark all incident elements that cannot be in $\mathrm{MS}_\theta(G, \pi)$ after adding this element, as dead. If the chosen element is dead, we do nothing. This sequential algorithm is equivalent to choosing a prefix of a permutation and then processing the permutation.

Let ℓ be an Extend element that is live and has more than b incident live elements after processing the prefix. We show that this event is unlikely. Since at the end of $(a|T|/b)$ rounds, ℓ has more than b incident live elements, before all of $(a|T|/b)$ rounds, it has more than b incident live elements. Hence, in round i of the sequential process, with probability of at least $b/(|T| - i) > b/|T|$, one of the incident live elements of ℓ will be selected. Note that if more than one incident element of ℓ is added to $\mathrm{MS}_\theta(G, \pi)$, then $\ell \notin \mathrm{MS}(G, \pi)$. (It is possible that ℓ is not in $\mathrm{MS}(G, \pi)$ because of one incident element, however, we provide a looser bound by only considering the existence of two incident elements.) Thus, the probability that at most one of its incident elements is selected is at most

$$\left(1 - \frac{b}{|T|}\right)^{\frac{a|T|}{b}} + \left(\frac{a|T|}{b}\right)\left(1 - \frac{b}{|T|}\right)^{\frac{a|T|}{b}-1} < \frac{2a|T|}{b}\left(1 - \frac{b}{|T|}\right)^{\frac{a|T|}{2b}} = \frac{2a|T|}{b}\left(1 - \frac{b}{|T|}\right)^{\frac{|T|}{b}\cdot\frac{a}{2}}, \tag{3.7}$$

where the first term of left-hand side is the probability that none of the incident elements is chosen and the second term is an upper bound for the probability that exactly one of the incident elements is selected. Combining equations (3.7), $(1 - \frac{b}{|T|})^{\frac{|T|}{b}} < (1/e)$, and $\frac{a}{b} < 1$, the probability of this event is at most $\frac{2|T|}{e^{a/2}}$. Taking a union bound over all elements completes the proof. □

Corollary 3.1.41. *If $\theta_i = \Omega(2^i \log n/(2K\Delta))$, then all remaining elements have at most $2K\Delta/2^i$ incident elements after round i.*

Proof. At the beginning, each element is incident to at most $2K\Delta$ elements since the degree of each vertex is at most Δ and we create K copies of each edge. Since $\log |T| = O(\log n)$, the proof can be obtained by Lemma 3.1.40 with $a = \Omega(\log n)$ and $b = 2K\Delta/2^i$ for partition i. □

In the previous lemma, by choosing the appropriate θ_i, the number of incident elements for each Extend element reduce by half after removing partition i. Hence, there will be at most $O(\log n)$ partitions. In the next lemma, we will show that the length of the longest query-path for Extend elements in each of $\mathcal{P}_i$ is bounded by $O(\log n)$.

Lemma 3.1.42. *Suppose that all* EXTEND *elements have at most x incident elements. Let a and b be two positive integers such that $a \geq b$ and consider a randomly ordered θ-prefix from T with $\theta < b/x$. Then the longest query-path consist of* EXTEND *elements has length $O(a)$ with probability of at least $1 - |T|(b/a)^a$.*

Proof. Let $(i_1, i_2, \ldots, i_{k+1})$ be $k+1$ different indices in the θ-prefix. Choosing the prefix elements sequentially is equivalent to choosing a randomly ordered prefix. Let $(\ell_{i_1}, \ell_{i_2}, \ldots, \ell_{i_{k+1}})$ be elements in these indices that create a query-path. Hence, the probability that ℓ_{i_1} and ℓ_{i_2} are incident is at most $x/(|T|-1)$ since when we select ℓ_{i_2}, element ℓ_{i_1} is already chosen and there are $|T|-1$ choices remaining and ℓ_1 has at most x incident elements. With the same argument, the probability that ℓ_2 and ℓ_3 be incident is at most $x/(|T|-2)$. Hence, the probability of having such a path is at most $(x/(|T|-k))^k$. Taking a union bound over all possible $k+1$ indices of the prefix, we get the following bound for having a query-path of length $k+1$. By assuming that $k < |T|/2$, we have

$$\begin{aligned}
\binom{\theta|T|}{k+1} \cdot (x/(|T|-k))^k &\leq \left(\frac{e\theta|T|}{k+1}\right)^{k+1} \cdot \left(\frac{x}{|T|-k}\right)^k \\
&= \frac{e\theta|T|}{k+1} \cdot \left(\frac{ex\theta|T|}{(k+1)(|T|-k)}\right)^k \\
&\leq \frac{e\theta|T|}{k+1} \cdot \left(\frac{2ex\theta}{k+1}\right)^k && (k \leq |T|/2) \\
&\leq |T| \left(\frac{2ex\theta}{k+1}\right)^{k+1}.
\end{aligned}$$

By setting $k = 2ea - 1$ and $\theta < b/x$ the above bound will be at most $|T|(b/a)^a$. Therefore, with probability $1 - |T|(b/a)^a$, the longest query-path has length $O(a)$. Note that if $k \geq |T|/2$, it implies that $2ea \geq |T|/2$ which the lemma clearly holds since $a = O(|T|)$. □

Corollary 3.1.43. *Suppose that all* EXTEND *elements have at most x incident elements. The longest query-path consisting of* EXTEND *elements, has length of at most $O(\log n)$ with probability $1 - \frac{1}{n^4}$ for a $O(\log(n)/x)$-prefix of T.*

Proof. Setting $a = 5b = O(\log |T|) = O(\log n)$ in Lemma 3.1.42 completes the proof. □

Now we are ready to complete the proof of Lemma 3.1.25.

Proof of Lemma 3.1.25. First, if the edge oracle recursively calls a START element, then we get from Lemma 3.1.39 and Observation 3.1.24 that the remaining length of the query-path will not exceed $O(\log n)$ with high probability. Therefore, we only need to show that the length of the longest query-path involving only EXTEND elements is bounded by $O(\log^2 n)$ with high probability. Note that according to the partitioning, if we choose $\theta_i = \Omega(2^i \log(n)/(2K\Delta))$, by Lemma 3.1.40, after round i each EXTEND element has at most $(2K\Delta)/2^i$ incident elements. This implies that the number of parts is $O(\log n)$ (i.e. $d = O(\log n)$). Furthermore, by Lemma 3.1.42, the longest path consisting of EXTEND elements in each part has length at most $O(\log n)$.

Now consider a query-path P. By Observation 3.1.38, at most one element of P is in each D_i for $i \leq d$. Moreover, by the above argument, there are at most $O(\log n)$ elements of P in each of $\mathcal{P}_i$ (see Figure 3.5). Therefore, the longest length of a query-path is bounded by $O(\log^2 n)$ with high probability (i.e. with probability of at least $1 - 1/n^2$).

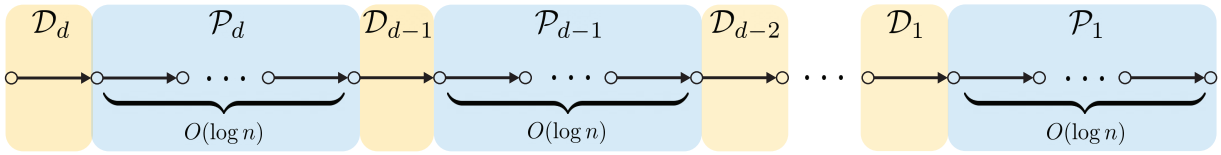


Figure 3.5: A possible query-path according to the partitioning. Blue boxes represent $\mathcal{P}_i$ and yellow boxes represent $\mathcal{D}_i$.

For each unlikely permutation $\pi \in U$, there exists a query-path of length larger than β. Since $\beta = c \log^2 n$, by choosing c large enough we have that $|U|/|\Pi| \leq 1/n^2$. Therefore, $|U| \leq ((K+1)m)!/n^2$ which implies that $|A_U| \leq ((K+1)m)!/n^2$ since A_U represents vertices that correspond to unlikely permutations. □

3.1.4 Our Estimator for the Adjacency List Model

In this section, we use the oracles introduced in the previous section to estimate the size of the matching in Algorithm 1. We assume that without loss of generality, the algorithm knows Δ, $\bar{d}$, and there is no singleton vertex in the graph (note that the algorithm can simply query degree of each vertex and compute Δ and $\bar{d}$).

Note that our upper bound of Section 3.1.3 is on $F(v, \pi)$ which recall is the number of recursive calls to the oracles, but not necessarily the running time needed to implement it. Generating the whole permutation π requires $|T| = \Theta(Km)$ time, which is too large for our purpose. Therefore, we have to generate π on the fly during the recursive calls whenever needed. Using the techniques developed first by [156] and further used by [34], we show in Section 3.1.6 that indeed it is possible to get an $\widetilde{O}(F(v, \pi))$ time implementation. In particular, we show that:

Lemma 3.1.44. *In the adjacency list model, there is a data structure that given a graph G, (implicitly) fixes a random permutation π over its edge set. Then for any vertex v, the data structure returns whether v has any edge in outputs M and S of Algorithm 1 according to a random permutation π. Each query v to the data structure is answered in $\tilde{O}(F(v, \pi))$ time w.h.p. where $F(v, \pi)$ is as defined in Section 3.1.3. Additionally, the vertices we feed into the oracle can be adaptively chosen depending on the responses to the previous calls.*

In order to estimate the output of our algorithm, we sample r random vertices, and for each vertex, we check if it is the endpoint of a Start element. Moreover, for each sampled vertex, we check if the vertex is an endpoint of a length three augmenting path that is created by a Start element as a middle edge and two other Extend elements.

Multiplicative Approximation

We use the following well-known claim about the size of maximum matching, a similar bound to which was also used in [34].

Claim 3.1.45. *Let G be a graph with maximum degree Δ and average degree $\bar{d}$. Then $\mu(G) \geq \frac{n\bar{d}}{4\Delta}$.*

Proof. Greedily edge color the graph using 2Δ colors. The color with the largest size is a matching of size at least $m/2\Delta = n\bar{d}/4\Delta$. □

Algorithm 5: Final algorithm for adjacency list.

H!
$r \leftarrow (384 \cdot \Delta \log n)/(\delta^2 \bar{d})$.
Sample r vertices $u_1, u_2, \ldots, u_r$ uniformly at random from V with replacement.
Run vertex oracle for each u_i and let S_i and E_i be the indicator that u_i has an incident START and EXTEND elements that appear in the output of Algorithm 1.
for i *in* $1 \ldots r$ **do**
    $X_i \leftarrow 0$
    if $S_i = 1$ **then** $X_i \leftarrow 1$;
    if $S_i = 0$ *and* $E_i = 1$ **then**
        Let w_i be the other endpoint of EXTEND element incident to u.
        Run vertex oracle for w_i (with the same permutation) and let S'_i be the indicator that w_i has an incident START element.
        if $S'_i = 0$ **then continue**;
        Let x_i be other endpoint of START element incident to w_i.
        Run vertex oracle for x_i (with the same permutation) and let E'_i be the indicator that x_i has an incident EXTEND element.
        if $E'_i = 0$ **then continue**;
        Let y_i be other endpoint of EXTEND element incident to x_i.
        Run vertex oracle for y_i (with the same permutation) and let S''_i be the indicator that y_i has an incident START element.
        if $S''_i = 0$ **then** $X_i \leftarrow 1$;
Let $X = \sum_{i \in [r]} X_i$ and $f = X/r$.
Let $\tilde{\mu} = (1 - \frac{\delta}{2}) fn/2$.
return $\tilde{\mu}$

We use this claim to show that the number of samples that is needed to estimate the output of Algorithm 1 is $r = \tilde{\Theta}(\Delta/\bar{d})$.

Remark 1. *In Algorithm 5, we do not estimate $\mu(M \cup S)$. Instead, we estimate the size of the maximal matching M and augment length-three augmenting paths in $M \cup S$. Since the bound in Theorem 3.1.2 is based on augmenting length three augmenting paths of $M \cup S$, we get the same approximation guarantee.*

We can change the vertex oracle and edge oracle to return the incident elements that appear in subgraph $M \cup S$. However, for simplicity of oracles, we return the indicators for having START and EXTEND elements but we assume that we have access to these elements.

Lemma 3.1.46. *Let $\tilde{\mu}$ be the output of Algorithm 5. With high probability,*

$$\left(\frac{1}{2} + \frac{\delta}{4}\right) \mu(G) \leq \tilde{\mu} \leq \mu(G),$$

where δ is as in the statement of Theorem 3.1.2.

Proof. Let X_i be the indicator for each vertex i which shows in output of Algorithm 1, either i has a START incident element, or it is an endpoint of a length three augmenting path created by a START element in the middle along with two EXTEND elements. Note that the way X_i is computed in Algorithm 5 is the same as the definition given earlier.

Let $\hat{M}(G, \pi)$ be the set of edges of the matching created by augmenting the length-three augmenting

paths of $M \cup S$. By Remark 1 and Theorem 3.1.2,

$$\left(\frac{1}{2} + \delta\right) \mu(G) \leq \mathbf{E}_\pi \left|\hat{M}(G, \pi)\right| \leq \mu(G). \tag{3.8}$$

Since the number of matched vertices is twice the number of edges in the matching,

$$\mathbf{E}[X_i] = \Pr[X_i = 1] = \frac{2\,\mathbf{E}_\pi \left|\hat{M}(G, \pi)\right|}{n}.$$

Similarly, we have

$$\mathbf{E}[X] = \frac{2r\,\mathbf{E}_\pi \left|\hat{M}(G, \pi)\right|}{n}. \tag{3.9}$$

Since X is the sum of r independent Bernoulli random variables, the Chernoff bound (Proposition 2.3.1) implies

$$\Pr[|X - \mathbf{E}[X]| \leq \sqrt{6\,\mathbf{E}[X] \log n}] \leq 2 \exp\left(-\frac{6\,\mathbf{E}[X] \log n}{3\,\mathbf{E}[X]}\right) = \frac{2}{n^2}. \tag{3.10}$$

Now, $fn = Xn/r$. Therefore, using the above equation, fn is in the following range with probability $1 - 2/n^2$.

$$\begin{aligned}
fn \in \frac{n(\mathbf{E}[X] \pm \sqrt{6\,\mathbf{E}[X] \log n)}}{r} &= \frac{n\,\mathbf{E}[X]}{r} \pm \frac{\sqrt{6n^2\,\mathbf{E}[X] \log n}}{r} \\
&= 2\,\mathbf{E}\left|\hat{M}(G, \pi)\right| \pm \sqrt{\frac{12n\,\mathbf{E}\left|\hat{M}(G, \pi)\right| \log n}{r}} && \text{(By (3.9))} \\
&= 2\,\mathbf{E}\left|\hat{M}(G, \pi)\right| \pm \sqrt{\frac{n\,\mathbf{E}\left|\hat{M}(G, \pi)\right| \bar{d}}{32 \cdot \delta^{-2} \cdot \Delta}} && \left(\text{Since } r = \tfrac{384 \cdot \Delta \log n}{\delta^2 \bar{d}}\right) \\
&\geq 2\,\mathbf{E}\left|\hat{M}(G, \pi)\right| \pm \sqrt{\frac{\mu(G)\,\mathbf{E}\left|\hat{M}(G, \pi)\right|}{8 \cdot \delta^{-2}}} && \left(\text{Since } \mu(G) \geq \tfrac{n\bar{d}}{4\Delta}\right)
\end{aligned}$$

By (3.8), we have $2\,\mathbf{E}\left|\hat{M}(G, \pi)\right| \geq \mu(G)$. Combining with Claim 3.1.45, we get

$$fn \in 2\,\mathbf{E}\left|\hat{M}(G, \pi)\right| \pm \sqrt{\frac{\mathbf{E}\left|\hat{M}(G, \pi)\right|^2}{4\delta^{-2}}} = (2 \pm \frac{\delta}{2})\,\mathbf{E}\left|\hat{M}(G, \pi)\right|.$$

Since $\tilde{\mu} = (1 - \frac{\delta}{2}) fn/2$, we have that

$$(1 - \delta)\,\mathbf{E}\left|\hat{M}(G, \pi)\right| \leq \tilde{\mu} \leq \mathbf{E}\left|\hat{M}(G, \pi)\right|$$

Next, note that by (3.8), $(\frac{1}{2} + \delta)\mu(G) \leq \mathbf{E}\left|\hat{M}(G, \pi)\right| \leq \mu(G)$. Hence,

$$(1 - \delta)\left(\frac{1}{2} + \delta\right) \mu(G) \leq \tilde{\mu} \leq \mu(G),$$

and thus

$$\left(\frac{1}{2} + \frac{\delta}{4}\right) \mu(G) \leq \tilde{\mu} \leq \mu(G). \quad \square$$

Lemma 3.1.47. *Algorithm 5 runs in $\tilde{O}(n + \Delta^{1+\varepsilon})$ with high probability.*

Proof. We show that for each of r sampled vertices in Algorithm 5, the iteration corresponding to the vertex in Line 5 takes $\tilde{O}(K\bar{d})$ time. First, note that for a random permutation π, the vertex oracle for a vertex u can be called a constant number of times. The reason is that the algorithm only queries vertex oracle for vertices of length three augmenting paths. Also, by Theorem 3.1.16, the vertex oracle takes $\tilde{O}(K\bar{d})$ time for random vertex and permutation. Therefore, if we run the vertex oracle a constant time for a fixed vertex, still the expected running time will be $\tilde{O}(K\bar{d})$. Since the number of samples is $O(\Delta \log n/\bar{d})$, the total running time for all samples will be $\tilde{O}(K\Delta)$. Furthermore, we spent $O(n)$ time for computing Δ, $\bar{d}$, and finding the isolated vertices.

In order to achieve a high probability bound on running time, we run $\Theta(\log n)$ instances of the algorithm simultaneously and return the estimation of the first instance that terminates. Since the expected running time is $\tilde{O}(n + K\Delta)$, the first instance terminates with probability $1 - 1/\text{poly}(n)$ in $\tilde{O}(n + K\Delta)$.

Plugging $K = \tilde{O}(\Delta^{\varepsilon})$ completes the proof. □

Multiplicative-Additive Approximation

We use the same algorithm as Algorithm 5, however, the $o(n)$ additive error allows us to sample $\tilde{\Theta}(1)$ vertices instead of $\tilde{\Theta}(\Delta/\bar{d})$ vertices. Moreover, we no longer need to estimate $\bar{d}$ and Δ since the number of samples is independent of these parameters.

Lemma 3.1.48. *Let $\tilde{\mu}$ be the output of Algorithm 5 with parameter $r = 12\log^3 n$ and estimation $\tilde{\mu} = fn/2 - \frac{n}{2\log n}$. With high probability,*

$$\left(\frac{1}{2} + \delta\right)\mu(G) - \frac{n}{\log n} \leq \tilde{\mu} \leq \mu(G).$$

Proof. Let X_i be defined the same as Lemma 3.1.46. With the exact same argument, inequalities (3.8), (3.9), and (3.10) hold with new parameter and estimation. Hence, with probability of at least $1 - 2/n^2$,

$$\begin{aligned}
fn \in \frac{n(\mathbf{E}[X] \pm \sqrt{6\,\mathbf{E}[X]\log n})}{r} &= \frac{n\,\mathbf{E}[X]}{r} \pm \sqrt{\frac{6n^2\,\mathbf{E}[X]\log n}{r^2}} \\
&= 2\,\mathbf{E}\,|\hat{M}(G,\pi)| \pm \sqrt{\frac{12n\,\mathbf{E}\,|\hat{M}(G,\pi)|\log n}{r}} && \text{(By (3.9))} \\
&= 2\,\mathbf{E}\,|\hat{M}(G,\pi)| \pm \sqrt{\frac{n\,\mathbf{E}\,|\hat{M}(G,\pi)|}{\log^2 n}} && \text{(Since } r = 12\cdot\log^3 n\text{)} \\
&\in 2\,\mathbf{E}\,|\hat{M}(G,\pi)| \pm \frac{n}{\log n} && \text{(Since } \mathbf{E}\,|\hat{M}(G,\pi)| \leq n\text{).}
\end{aligned}$$

Since $\tilde{\mu} = fn/2 - \frac{n}{2\log n}$, we have that

$$\mathbf{E}\,|\hat{M}(G,\pi)| - \frac{n}{\log n} \leq \tilde{\mu} \leq \mathbf{E}\,|\hat{M}(G,\pi)|.$$

By plugging (3.8), we get

$$\left(\frac{1}{2} + \delta\right)\mu(G) - \frac{n}{\log n} \leq \tilde{\mu} \leq \mu(G). \quad \square$$

Lemma 3.1.49. *Algorithm 5 with parameter $r = 12\log^3 n$, runs in $\tilde{O}(\bar{d} \cdot \Delta^{\varepsilon})$ with high probability.*

Proof. First, note that we do not need to spend $\tilde{O}(n)$ time to estimate Δ and $\bar{d}$, since the number of samples is independent of these parameters. By the exact same argument as the proof of Lemma 3.1.47, we can show that the expected running time for each sampled vertex is $\tilde{O}(K\bar{d})$. Since $r = \tilde{(}O)(1)$, the total running time will be $\tilde{O}(K\bar{d})$.

To get a high probability bound on the running time, similar to Lemma 3.1.47, we run $\Theta(\log n)$ instances of the algorithm simultaneously. Using the same argument, the first instance terminates with probability $1 - 1/\text{poly}(n)$ in $\tilde{O}(K\bar{d})$.

Plugging $K = \tilde{O}(\Delta^{\varepsilon})$ completes the proof. □

3.1.5 Our Estimator for the Adjacency Matrix Model

In this section, we give a reduction from the adjacency matrix model to the adjacency list model such that each query in the adjacency list can be implemented with a constant number of queries in the adjacency matrix model. Such a reduction appears in [34, Section 5]. We use a similar idea with a minor modification in the parameters of the construction.

Let $\gamma = (4\log n) \cdot n$. We construct a graph $H = (V_H, E_H)$ as follows:

- V_H is the union of V_1, V_2 and $U_1, U_2, \ldots, U_n$ such that:
  - V_1 and V_2 are two copies of the vertex set of the original graph G.
  - U_i is a vertex set of size γ for each $i \in [n]$.
- We define the edge set such that degree of each vertex is in $\{1, n, n+\gamma\}$:
  - Degree of each vertex $v \in V_1$ is n. The i-th neighbor of v is the i-th vertex in V_1 if $(v,i) \in E$, otherwise, its i-th neighbor is the i-th vertex in V_2 for $i \leq n$. Note that graph $(V_1, E_H \cap (V_1 \times V_1))$ is isomorphic to G.
  - Degree of each vertex $v \in V_2$ is $n+\gamma$. The i-th neighbor of v is the i-th vertex in V_2 if $(v,i) \in E$, otherwise, its i-th neighbor is the i-th vertex in V_1 for $i \leq n$. For all $n < i \leq n+\gamma$, the i-th neighbor of v is i-th vertex in U_v.
  - Degree of each vertex $u \in U_i$ is one for $i \in [n]$. The only neighbor of u is the i-th vertex of V_2.

Observation 3.1.50. *For each vertex $v \in V_H$ and $i \in [\deg_H(v)]$, the i-th neighbor of vertex v can be determined using at most one query to the adjacency matrix.*

Proof. First, note that the degree of each vertex is not dependent on the original graph G. Hence, we do not need any queries to find the degree of each vertex. For each vertex $v \in U_1 \cup U_2 \cup \ldots \cup U_n$, one can find its only neighbor without any queries by the construction of H. For each vertex $v \in V_1 \cup V_2$, the i-th neighbor is either the i-th vertex of V_1 or the i-vertex of V_2. Therefore, with at most one query, one can determine the i-th neighbor of vertex v. □

Consider a random permutation π over the list of elements T, consisting of START and EXTEND copies of E_H. Intuitively, for almost all vertices $v \in V_2$, the first incident START element to v in T is an edge between

v and U_v. Similarly, the first two incident EXTEND elements to v are between v and U_v. We say v is a *bad* vertex, if it violates the mentioned conditions. Let $R \subseteq V_2$ be the set of bad vertices. Since the permutation over edges in $H[V_1 \cup R]$ is uniformly at random, using Theorem 3.1.2, we can provide a bound on the size of the matching produced by the algorithm.

Observation 3.1.51. *For each $v \in V_2 \setminus R$, the incident* START *and* EXTEND *elements in* $\mathrm{MS}(H, \pi)$ *are between v and U_v. Moreover, both incident* START *and* EXTEND *elements in* $\mathrm{MS}(H, \pi)$ *appears before all edges between v and $V_1 \cup V_2$.*

Proof. By the definition of R, the first START element incident to v in the permutation is between v and U_v. Let this edge be (v, w). Hence, the algorithm adds (v, w) to $\mathrm{MS}(H, \pi)$. Note that all START elements incident to v after (v, w) in the permutation cannot be in $\mathrm{MS}(H, \pi)$ since START elements create a maximal matching. By definition of R, the first two EXTEND elements incident to v are between v and U_v. Let (v, u_1) and (v, u_2) be these two edges ($u_1 \neq u_2$ since there is one EXTEND copy). Therefore, one of (v, u_1) and (v, u_2) must be added to $\mathrm{MS}(H, \pi)$ and no other EXTEND element incident to v can be added to $\mathrm{MS}(H, \pi)$ since EXTEND elements create a maximal matching. □

Observation 3.1.52. *It holds that*

$$\left(\frac{1}{2} + \delta\right) \mu(H[V_1 \cup R]) \leq \mathbf{E}_\pi \left|\mathrm{MS}(H, \pi) \cap ((V_1 \cup R) \times (V_1 \cup R))\right| \leq \mu(H[V_1 \cup R]).$$

Proof. Note that by Observation 3.1.51, all vertices in $V_2 \setminus R$ have incident START and EXTEND elements in $\mathrm{MS}(H, \pi)$ which appear before edges between $V1 \cup R$ and $V_2 \setminus R$ in the permutation. Hence, none of these edges can be added to the subgraph $\mathrm{MS}(H, \pi)$ in Algorithm 1. Since the permutation over edges in $H[V_1 \cup R]$ is uniformly at random, using Theorem 3.1.2, we have the given bound □

Next, we provide an upper bound for the size of R.

Observation 3.1.53. *It holds that* $\mathbf{E}_\pi |R| \leq \frac{n}{2 \log n}$.

Proof. For vertex $v \in V_2$ and a random permutation π over E_H, the first incident START element is between v and U_v, with probability of at least $\frac{K\gamma}{(n+\gamma)K} \geq 1 - \frac{1}{4 \log n}$. Furthremore, with probability $\frac{\gamma(\gamma-1)}{(n+\gamma)(n+\gamma-1)} \geq 1 - \frac{1}{4 \log n}$, the first two EXTEND elements are between v and U_v. Since these events are independent, the probability of v not being a bad vertex is at least

$$\left(1 - \frac{1}{4 \log n}\right)^2 \geq 1 - \frac{1}{2 \log n}.$$

Therefore, $|R|$ is at most $\frac{n}{2 \log n}$ in expectation over a random permutation. □

With the intuition that a few vertices in V_2 are matched to vertices in $V_1 \cup V_2$, our goal is to estimate the number of vertices that have a matching edge in V_1. Since we have an upper bound on the size of the R, we are able to estimate the number of matching edges in $H[V_1] = H[G]$. In order to count the number of matching edges in V_1, we need to run the vertex oracle for vertices of $V_1 \cup V_2$. We show that the expected running time of vertex oracle on a random vertex of $V_1 \cup V_2$ is $\tilde{O}(n^{1+\varepsilon})$.

Claim 3.1.54. *Let v be a random vertex in $V_1 \cup V_2$ and π be a random permutation over E_H. Then*

$$\mathbf{E}_{v\sim(V_1\cup V_2),\pi}[F(v,\pi)] = \tilde{O}(n^{1+\varepsilon}).$$

Proof. By Theorem 3.1.16, we have

$$\mathbf{E}_{v\sim V_H,\pi}[F(v,\pi)] = O\left(K\frac{|E_H|}{|V_H|}\log^4|V_H|\right).$$

Summing over all vertices of V_H, we get

$$\sum_{v\in V_H}\mathbf{E}_{\pi}[F(v,\pi)] = O(K|E_H|\cdot\log^4|V_H|) = \tilde{O}(n^{2+\varepsilon}),$$

because $|V_H| = O(n^2)$, $|E_H| = O(n^2+n\gamma)$, $K = \tilde{O}(n^{\varepsilon})$, and $\gamma = (4\log n)\cdot n$. Therefore,

$$\mathbf{E}_{v\sim(V_1\cup V_2),\pi}[F(v,\pi)] \le \left(\sum_{v\in V_H}\mathbf{E}_{\pi}[F(v,\pi)]\right)/|(V_1\cup V_2)| = \tilde{O}(n^{1+\varepsilon}). \quad \square$$

Claim 3.1.55. *Let v be a random vertex in V_1 and π be a random permutation over E_H. Then*

$$\mathbf{E}_{v\sim V_1,\pi}[F(v,\pi)] = \tilde{O}(n^{1+\varepsilon}).$$

Proof. Proof follows by combining Claim 3.1.54 and $|V_1| = |V_2|$. $\square$

Lemma 3.1.56. *Let $\tilde{\mu}$ be the output of Algorithm 6. With high probability,*

$$\left(\frac{1}{2}+\delta\right)\mu(G) - \frac{n}{\log n} \le \tilde{\mu} \le \mu(G).$$

Proof. Let $\hat{M}(H,\pi)$ be the intersection of edges between V_1 and set of edges of the matching that is created by augmenting the length-three augmenting paths of $M\cup S$. We claim

$$\left(\frac{1}{2}+\delta\right)\mu(H[V_1]) - \frac{n}{2\log n} \le \mathbf{E}_{\pi}|\hat{M}(H,\pi)| \le \mu(H[V_1]). \tag{3.11}$$

By combining Observation 3.1.52 and Observation 3.1.53, imply that there are at most $\frac{n}{2\log n}$ edges in output of algorithm in $H[V_1\cup R]$ with at least one endpoint in R. Therefore, by combining Remark 1, Theorem 3.1.2, and the bound for number of edges with at least one endpoint in R, we have the first inequality. Furthermore, since $\hat{M}(H,\pi)$ is a matching of $H[V_1]$, we have the second inequality.

By definition, X_i is the indicator of the event that a vertex $i\in V_1$ is matched in the output of Algorithm 1 to another vertex in V_1. Since the number of matched vertices is twice the number of matching edges,

$$\mathbf{E}[X_i] = \Pr[X_i = 1] = \frac{2\,\mathbf{E}|\hat{M}(H,\pi)|}{n}.$$

Algorithm 6: Final algorithm for adjacency matrix.

Let $H = (V_H, E_H)$ as described above.
$r \leftarrow 48 \cdot \log^3 n$.
Sample r vertices $u_1, u_2, \ldots, u_r$ uniformly at random from V_1 with replacement.
Run vertex oracle for each u_i and let S_i and E_i be the indicator that u_i has an incident START and EXTEND elements that appear in output of Algorithm 1.
for *i in* $1 \ldots r$ **do**
    $X_i \leftarrow 0$
    if $S_i = 1$ **then**
        Let w_i be the other endpoint of START edge incident to u.
        if $w_i \in V_1$ **then** $X_i \leftarrow 1$;
    if $S_i = 0$ *and* $E_i = 1$ **then**
        Let w_i be the other endpoint of EXTEND element incident to u.
        Run vertex oracle for w_i (with same π) and let S'_i be the indicator that w_i has an incident START edge.
        if $S'_i = 0$ **then continue**;
        Let x_i be other endpoint of START element incident to w_i.
        Run vertex oracle for x_i (with same π) and let E'_i be the indicator that x_i has an incident EXTEND element.
        if $E'_i = 0$ **then continue**;
        Let y_i be other endpoint of EXTEND element incident to x_i.
        Run vertex oracle for y_i (with same π) and let S''_i be the indicator that y_i has an incident START element.
        if $S''_i = 0$ *and* $w_i \in V_1$ **then** $X_i \leftarrow 1$;
Let $X = \sum_{i \in [r]} X_i$ and $f = X/r$.
Let $\tilde{\mu} = fn/2 - \frac{n}{4 \log n}$.
return $\tilde{\mu}$

Similarly,

$$\mathbf{E}[X] = \frac{2r \, \mathbf{E}\,|\hat{M}(H,\pi)|}{n}. \tag{3.12}$$

Since X is the sum of r independent Bernoulli random variables, the Chernoff bound (Proposition 2.3.1) implies

$$\Pr[|X - \mathbf{E}[X]| \le \sqrt{6 \, \mathbf{E}[X] \log n}] \le 2 \exp\left(-\frac{6 \, \mathbf{E}[X] \log n}{3 \, \mathbf{E}[X]}\right) = \frac{2}{n^2}.$$

Note that $fn = Xn/r$, so using the above equation, fn is in the following range with probability $1 - 2/n^2$:

$$\begin{aligned} fn \in \frac{n(\mathbf{E}[X] \pm \sqrt{6 \, \mathbf{E}[X] \log n})}{r} &= \frac{n \, \mathbf{E}[X]}{r} \pm \sqrt{\frac{6n^2 \, \mathbf{E}[X] \log n}{r^2}} \\ &= 2 \, \mathbf{E}\,|\hat{M}(H,\pi)| \pm \sqrt{\frac{12n \, \mathbf{E}\,|\hat{M}(H,\pi)| \log n}{r}} && \text{(By (3.12))} \\ &= 2 \, \mathbf{E}\,|\hat{M}(H,\pi)| \pm \sqrt{\frac{n \, \mathbf{E}\,|\hat{M}(H,\pi)|}{4 \log^2 n}} && \text{(Since } r = 48 \cdot \log^3 n\text{)} \end{aligned}$$

$$\in 2\,\mathbf{E}\,|\hat{M}(H,\pi)| \pm \frac{n}{2\log n} \qquad \text{(Since } \mathbf{E}\,|\hat{M}(H,\pi)| \leq n\text{).}$$

Now, $\tilde{\mu} = fn/2 - \frac{n}{4\log n}$. Therefore,

$$\mathbf{E}\,|\hat{M}(H,\pi)| - \frac{n}{2\log n} \leq \tilde{\mu} \leq \mathbf{E}\,|\hat{M}(H,\pi)|.$$

Combining the above range with (3.11) implies

$$\left(\frac{1}{2}+\delta\right)\mu(G) - \frac{n}{\log n} \leq \tilde{\mu} \leq \mu(G). \quad \square$$

Lemma 3.1.57. *Algorithm 6 runs in $\tilde{O}(n^{1+\varepsilon})$ with high probability.*

Proof. First, we show that the iteration in Line 5, for each of r sampled vertices from V_1 takes $\tilde{O}(n^{1+\varepsilon})$ time in expectation. With the same argument as Lemma 3.1.47, the vertex oracle for a vertex u can be called a constant number of times. Moreover, by Claim 3.1.54 and Claim 3.1.55, the vertex oracle takes $\tilde{O}(n^{1+\varepsilon})$ time for a random vertex in $V_1 \cup V_2$ and a random permutation (since set R is a uniformly at random set in V_2). Therefore, if we run the vertex oracle constant time for a fixed vertex, still the expected running time will be $\tilde{O}(n^{1+\varepsilon})$.

In order to achieve a high probability bound on the running time, similar to the proof of Lemma 3.1.47, we run $\Theta(\log n)$ instances of the algorithm simultaneously. Using the same argument, the first instance terminates with probability $1 - 1/\text{poly}(n)$ in $\tilde{O}(n^{1+\varepsilon})$. $\square$

Proof of Theorem 3.1.1.

- By combining Lemma 3.1.46 and Lemma 3.1.47, we get multiplicative factor of $(\frac{1}{2}+\delta)$ in the adjacency list model in $\widetilde{O}(n + \Delta^{1+\varepsilon})$ time.
- By combining Lemma 3.1.48 and Lemma 3.1.49, we get multiplicative-additive factor of $(\frac{1}{2}+\delta, o(n))$ in the adjacency list model in $\widetilde{O}(\bar{d}\cdot\Delta^{\varepsilon})$ time.
- By combining Lemma 3.1.56 and Lemma 3.1.57, we get multiplicative-additive factor of $(\frac{1}{2}+\delta, o(n))$ in the adjacency matrix model in $\tilde{O}(n^{1+\varepsilon})$ time. For a constant ε, running this algorithm for a slightly smaller value of ε allows us to get rid of polylogarithmic factor in the running time and achieve an $O(n^{1+\varepsilon})$ time algorithm. $\square$

3.1.6 Implementation Details

In this section, we provide an implementation of our vertex and edge oracle. The idea is similar to the [34, Appendix A]. The main difference is that instead of having at most one edge between two vertices, here we have $K+1$ edges where K of them correspond to START copies and one of them corresponds to EXTEND copy. The idea is to generate a random permutation π locally and sort edges based on π to create the permutation.

Note that in the vertex oracle and edge oracle, when an START incident element appears in $\text{MS}(G,\pi)$, the algorithm no longer queries on the START incident elements (the same also holds for EXTEND elements). Therefore, instead of having one graph, we assume that we have two graphs G_s and G_e that are isomorphic

to G. Also, we assume that each edge in G_s has K copies that is corresponds to the number of START elements. In other words, graph G_s is the a graph made by all START elements of G and G_e is the a graph made by all EXTEND elements of G.

Let $\text{LOWEST}_{G_t}(u, i)$ for $t \in \{s, e\}$ be a procedure that returns a pair of an edge e in G_t and its ranking such that e is the i-th lowest rank edge incident to u in G_t. We use the same implementation of LOWEST_{G_t} procedure as the [34, Appendix A] (note that in this paper, the procedure only returns the edge. However, in the implementation of the LOWEST, they compute the edge ranking and we can simply return the edge ranking). Let $\deg_{G_t}(u)$ be the degree of vertex u in graph G_t. By definition of the G_s and G_e, we have that $\deg_{G_s}(u) = K \deg_G(u)$ and $\deg_{G_e}(u) = \deg_G(u)$.

Claim 3.1.58 ([34])**.** *Let u be a vertex and suppose that we call procedure $\text{LOWEST}_{G_t}(u, i)$ for all $1 \leq i \leq j$. The total time to implement all these calls is $\tilde{O}(j)$ with high probability for all $u \in V$.*

We present the implementation of our oracles in the following three algorithms. Note that we can generate vertex colors (i.e. c_v for vertex v) on the fly when it is needed. Also, we can toss a coin with a probability $1 - p$ for each edge to determine if it is a frozen edge or not. Therefore, once we have the rank of edges in our oracles, we can distinguish whether an edge is frozen or not based on the Definition 3.1.15. However, to make algorithms easier to read, we do not include the technical details of this part.

Lemma 3.1.44. *In the adjacency list model, there is a data structure that given a graph G, (implicitly) fixes a random permutation π over its edge set. Then for any vertex v, the data structure returns whether v has any edge in outputs M and S of Algorithm 1 according to a random permutation π. Each query v to the data structure is answered in $\tilde{O}(F(v, \pi))$ time w.h.p. where $F(v, \pi)$ is as defined in Section 3.1.3. Additionally, the vertices we feed into the oracle can be adaptively chosen depending on the responses to the previous calls.*

Proof. Note that in Algorithm 7, we only call LOWEST procedure for the neighbors that we recursively call edge oracle for them. Similarly, in Algorithm 8 and Algorithm 9 we only call LOWEST procedure for incident elements that we will recursively call the edge oracle on them. Therefore, by Claim 3.1.58, the total time spent on LOWEST procedure calls is $\tilde{O}(F(v, \pi))$ for a random permutation π and every vertex v with high probability. □

3.2 A Simpler Sublinear Algorithm to Beat Greedy Matching

In this section, we show how to beat the factor 0.5 with a simpler algorithm in time that is strongly sublinear. Specifically, we present an algorithm that runs in time $\widetilde{O}(n\sqrt{n})$ and achieves an approximation factor of 0.5109 for estimating the size of the maximum matching. It is worth noting that our algorithm is much simpler, both in terms of implementation and analysis, compared to the prior section's results. We prove the following two results in this section.

Theorem 3.2.1. *There exists an algorithm that, given access to the adjacency list of a graph, estimates the size of the maximum matching with a multiplicative approximation factor of 0.5109 and runs in $\widetilde{O}(n\sqrt{n})$ time with high probability.*

Algorithm 7: Implementation of the vertex oracle $\mathrm{VO}(u)$.

$j_s \leftarrow 1, j_e \leftarrow 1$
$v_s, \pi_s \leftarrow \textsc{Lowest}_{G_s}(u, j_s)$
$v_e, \pi_e \leftarrow \textsc{Lowest}_{G_e}(u, j_e)$
$ST \leftarrow \textsc{False}$, $EX \leftarrow \textsc{False}$
while $j_s \leq \deg_{G_s}(u)$ *or* $j_e \leq \deg_{G_e}(u)$ **do**
  if $\pi_s < \pi_e$ **then**
    $X \leftarrow \textsc{Start}$
    $\ell = ((u, v_s), X)$
  else
    $X \leftarrow \textsc{Extend}$
    $\ell = ((u, v_e), X)$
  if $X = \textsc{Extend}$ *and* $c_u = c_{v_e}$ **then continue**;
  if $ST = \textsc{False}$ *and* $X = \textsc{Start}$ *and* $\mathrm{EOS}(\ell, v_s) = \textsc{True}$ **then**
    $ST \leftarrow \textsc{True}$
    $j_s \leftarrow j_s + 1$
    if $j_s \leq \deg_{G_s}(u)$ **then**
      $v_s, \pi_s \leftarrow \textsc{Lowest}_{G_s}(u, j_s)$
    else
      $v_s, \pi_s \leftarrow \infty, \infty$
  if $EX = \textsc{False}$ *and* $X = \textsc{Extend}$ *and* $\mathrm{EOE}(\ell, v_e, ST) = \textsc{True}$ **then**
    $EX \leftarrow \textsc{True}$
    $j_e \leftarrow j_e + 1$
    if $j_e \leq \deg_{G_e}(u)$ **then**
      $v_e, \pi_e \leftarrow \textsc{Lowest}_{G_e}(u, j_e)$
    else
      $v_e, \pi_e \leftarrow \infty, \infty$
  return ST, EX

Algorithm 8: Implementation of the edge oracle for $\textsc{Start}$ elements $\mathrm{EOS}(\ell, u)$.

if $\mathrm{EOS}(\ell, u)$ *is already computed* **then return** the computed answer;
$j \leftarrow 1$
$w, \pi_w \leftarrow \textsc{Lowest}_{G_s}(u, j)$
while $w \neq v$ **do**
  $\ell' \leftarrow ((u, w), \textsc{Start})$
  if $\mathrm{EOS}(\ell', w) = \textsc{True}$ **then return** $\textsc{False}$;
  $j \leftarrow j + 1$
  $w \leftarrow \textsc{Lowest}_{G_s}(u, j)$
return $\textsc{True}$

Algorithm 9: Implementation of the edge oracle for EXTEND elements $\mathrm{EOE}(\ell, u, ST_w)$.

if $\mathrm{EOE}(\ell, u, ST_w)$ *is already computed* **then return** the computed answer;
$j_s \leftarrow 1, j_e \leftarrow 1$
$v_s, \pi_s \leftarrow \textsc{Lowest}_{G_s}(u, j_s)$
$v_e, \pi_e \leftarrow \textsc{Lowest}_{G_e}(u, j_e)$
$ST_u \leftarrow \textsc{False}$
if $\pi_s < \pi_e$ **then**
  $w \leftarrow v_s, X' \leftarrow \textsc{Start}$
else
  $w \leftarrow v_e, X' \leftarrow \textsc{Extend}$
$\ell' \leftarrow ((u, w), X')$
while $w \neq v$ *or* $X' \neq X$ **do**
  if $X' = \textsc{Extend}$ *and* $c_u = c_w$ **then continue**;
  if $X' = \textsc{Start}$ **then**
    if $ST_u = \textsc{False}$, *and* $\mathrm{EOS}(\ell', w) = \textsc{True}$ **then**
      $ST_u \leftarrow \textsc{True}$
      if ℓ' *is frozen* **then return** FALSE;
      if $ST_w = \textsc{True}$ **then return** FALSE;
    $j_s \leftarrow j_s + 1$
    if $j_s \leq \deg_{G_s}(u)$ **then**
      $v_s, \pi_s \leftarrow \textsc{Lowest}_{G_s}(u, j_s)$
    else
      $v_s, \pi_s \leftarrow \infty, \infty$
  if $X' = \textsc{Extend}$ **then**
    if $\mathrm{EOE}(\ell', w, ST_u) = \textsc{True}$ **then return** FALSE;
    $j_e \leftarrow j_e + 1$
    if $j_e \leq \deg_{G_e}(u)$ **then**
      $v_e, \pi_e \leftarrow \textsc{Lowest}_{G_e}(u, j_e)$
    else
      $v_e, \pi_e \leftarrow \infty, \infty$
  if $\pi_s < \pi_e$ **then**
    $w \leftarrow v_s, X' \leftarrow \textsc{Start}$
  else
    $w \leftarrow v_e, X' \leftarrow \textsc{Extend}$
  $\ell' \leftarrow ((u, w), X')$
return TRUE

Theorem 3.2.2. *There exists an algorithm that, given access to the adjacency matrix of a graph, estimates the size of the maximum matching with a multiplicative-additive approximation factor of* $(0.5109, o(n))$ *and runs in* $\widetilde{O}(n\sqrt{n})$ *time with high probability.*

In Section 3.2.1, we provide an overview of the challenges encountered while designing our algorithm and the techniques used to address them. We first develop an algorithm for bipartite graphs with a multiplicative-additive error in Section 3.2.2, avoiding additional challenges that arise from general graphs, trying to obtain multiplicative error (in the adjacency list model), or working with the adjacency matrix. In Section 3.2.3, we extend our algorithm to handle general graphs. In Section 3.2.4, we demonstrate how to achieve a multiplicative approximation guarantee. Finally, in Section 3.2.5, we present a simple reduction showing that our algorithm also works in the adjacency matrix model with a multiplicative-additive error.

3.2.1 Technical Overview

In this section, we provide an overview of the techniques used to design our algorithm. We begin with the two-pass semi-streaming algorithm of Konrad and Naidu [141] for bipartite graphs. In the first pass, the algorithm constructs a maximal matching M. In the second pass, it constructs a maximal b-matching between vertices matched in M and those unmatched in M. More specifically, each vertex in $V(M)$ has a capacity of k, while each vertex in $V \setminus V(M)$ has a capacity of kb, where $b = 1 + \sqrt{2}$ and k is a large constant. The idea is that if M is far from maximum, the b-matching will contain many length-3 augmenting paths that can be used to augment M. This algorithm obtains a $(2 - \sqrt{2}) \approx 0.585$-approximation.

Our goal is to develop a sublinear-time algorithm by translating this semi-streaming two-pass algorithm to the sublinear time model. When trying to do so, several challenges arise. In this section we describe them step by step, and show how to overcome them.

Challenge (1): constructing a maximal matching in sublinear time is not possible. In fact, finding all edges of any constant-factor approximation of the maximum matching is impossible in sublinear time due to [159]. Dynamic algorithms for maximum matching [32, 60] use the following approach: they maintain a maximal matching M and then apply the sublinear-time **random greedy maximal matching (RGMM)** algorithm of Behnezhad [34] to estimate the size of the maximal b-matching. In our setting, we cannot afford to explicitly construct M. However, we can obtain oracle access to M using the sublinear-time RGMM algorithm of [34]. More specifically, we can query whether a vertex v is matched in M or not in $\widetilde{O}(n)$ time. Therefore, a possible solution to address the first challenge is to design two nested oracles: the outer oracle attempts to build a maximal b-matching, whereas the inner oracle checks the status of vertices (matched or not in M) to correctly filter edges and assign capacities to each vertex.

Challenge (2): two nested oracles require $\Omega(n^2)$ time. The algorithm of [34] runs in $\widetilde{O}(\bar{d}(G))$ time, where $\bar{d}(G)$ denotes the average degree of the graph G. Additionally, for the outer oracle, it requires $\widetilde{O}(\bar{d}(G[V(M), V \setminus V(M)]))$ time (i.e., queries to the inner oracle). Unfortunately, it is possible for both $\bar{d}(G)$ and $\bar{d}(G[V(M), V \setminus V(M)])$ to be as large as $\Omega(n)$. For example, consider a graph with a vertex set $A \cup B$, where $|A| = |B| = n/2$. The edges within A form a complete bipartite graph, while there is an $\varepsilon n/2$-regular graph between A and B. After running the RGMM algorithm, most edges in the maximal matching belong

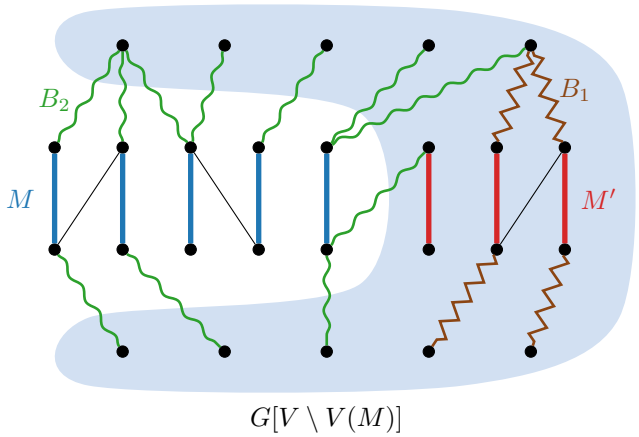


Figure 3.6: Our algorithm explicitly constructs a matching M (blue), which need not be maximal in G. We extend it with another matching M' (red), such that $M \cup M'$ is maximal. The highlighted (light blue) subgraph $G[V \setminus V(M)]$ has degree at most $\sqrt{n}$ with high probability. In case 1, our algorithm augments M' using a b-matching B_1 (zigzag edges, brown). In case 2, our algorithm augments M using a b-matching B_2 (swirly edges, green).

to $G[A]$, and most vertices in B are unmatched. Consequently, we have $\bar{d}(G[V(M), V \setminus V(M)]) = \Omega(n)$.

To address this issue, we sparsify the original graph while manually constructing a matching M. In a preprocessing step, starting from an empty M, for each unmatched vertex in the graph, we sample $\widetilde{\Theta}(\sqrt{n})$ neighbors uniformly at random. If an unmatched neighbor exists, we match the two vertices and add this edge to M. Using this preprocessing step, we show that after spending $\widetilde{O}(n\sqrt{n})$ time, the induced subgraph of vertices that remain unmatched in M has a maximum degree of $\sqrt{n}$ with high probability. Moreover, since we explicitly materialize M, we are able to check if any vertex is matched in M in $O(1)$ time, eliminating the need for costly oracle calls. Note that M need not be maximal in G; therefore we next extend it to a maximal matching.

Let M' be a maximal matching in $G[V \setminus V(M)]$ obtained by running the sublinear time RGMM algorithm of [34]. Now $M \cup M'$ is a maximal matching for G. Inspired by the two-pass semi-streaming algorithm of [141], we attempt to augment the maximal matching $M \cup M'$ in two possible ways (see also Figure 3.6):

1. Augment M' using a b-matching between $V(M')$ and $V \setminus V(M) \setminus V(M')$. The algorithm then outputs the size of the augmented matching, plus the size of the previously constructed matching M.

2. Augment M using a b-matching between $V(M)$ and $V \setminus V(M)$. The algorithm then outputs the size of the augmented matching.

The key intuition here (think of the case when $M \cup M'$ yields only a 0.5-approximation) is that either M' is sufficiently large, making $|M| + (2-\sqrt{2}) \cdot 2|M'|$ larger than the approximation guarantee, or M itself is large enough so that $(2-\sqrt{2}) \cdot 2|M|$ meets our approximation guarantee (note that $(2-\sqrt{2})$ is the approximation guarantee of [141]). Augmenting M using a b-matching is easier since we have explicit access to M and only need to run a single RGMM oracle to estimate the size of the b-matching. Our first estimate, which requires finding a b-matching between $V(M')$ and $V \setminus V(M) \setminus V(M')$, is more challenging since we do not have explicit access to M'. To avoid the $\Omega(n^2)$ running time of the two nested oracles, we make crucial use of the aforementioned property that the subgraph of vertices unmatched in M has low induced degree (at

most $\sqrt{n}$); this is the reason why we only try to augment M' rather than $M \cup M'$. We will discuss this in the next paragraphs.

Challenge (3): the algorithm does not have access to the adjacency list of $G[V \setminus V(M)]$. After the sparsification step, the average degree d of $G[V \setminus V(M)]$ is at most $\sqrt{n}$. Hence, if the algorithm had access to the adjacency list of $G[V \setminus V(M)]$, it could run the nested oracles in $\widetilde{O}(d^2) = \widetilde{O}(n)$ time by executing two RGMM algorithms: inner oracle for computing M' and outer oracle for the b-matching to augment M'. But, since the nested oracles may visit up to n vertices, and retrieving the full adjacency list of a vertex in $G[V \setminus V(M)]$ requires $\Omega(n)$ time, it seems that the overall running time of the algorithm could still be as high as $\Omega(n^2)$.

Here, we leverage two key properties of the RGMM algorithm to refine the runtime analysis. The first property is that at each step, the algorithm requires only a random neighbor of the currently visited vertex. Intuitively, if a vertex has degree $\Theta(n)$ in G, in expectation it takes $O(n/d)$ samples from the adjacency list of the original graph to encounter a vertex from $G[V \setminus V(M)]$. Thus, if all vertices in $G[V \setminus V(M)]$ had degree d, one could easily argue that the running time of the algorithm is $\widetilde{O}(d^2 \cdot n/d) = \widetilde{O}(n\sqrt{n})$. However, vertex degrees can vary, and for a vertex with a constant degree, we would need $\Omega(n)$ samples from the adjacency list of G to find a single neighbor in $G[V \setminus V(M)]$. To address this challenge, we utilize another property of the RGMM algorithm, recently proven by [148]. Informally, this result shows that during oracle calls for RGMM, a vertex is visited proportionally to its degree, implying that low-degree vertices are visited only a small number of times.

Challenge (4): outer oracle creates biased inner oracle queries. The final main challenge we discuss here is that the simple $\widetilde{O}(n\sqrt{n})$ bound, which we informally proved in the previous paragraph, relies on the tacit assumption that the inner oracle queries generated by the outer oracle correspond to $\widetilde{O}(\sqrt{n})$ uniformly random calls to the inner oracle. Indeed, the running time of the algorithm of [34] is analyzed for a uniformly random query vertex; however, there may exist a vertex v in the graph for which calling the inner oracle takes significantly more than $\widetilde{O}(d)$ time. Consequently, if all outer oracle calls end up querying v, the running time could be significantly worse than $\widetilde{O}(n\sqrt{n})$. To overcome this issue, we use the result of [148] along with the fact that the maximum degree of $G[V \setminus V(M)]$ is $\widetilde{O}(\sqrt{n})$. We show that for any vertex v, the outer oracle queries the inner oracle for v at most $\widetilde{O}(\deg_{G[V \setminus V(M)]}(v)/\sqrt{n})$ times in expectation. This enables us to formally prove that the total running time of the algorithm remains at most $\widetilde{O}(n\sqrt{n})$.

General graphs and the adjacency matrix model. There are additional challenges when dealing with general graphs as opposed to bipartite graphs, such as the fact that the sizes of the maximal matching and the b-matching alone are insufficient to achieve a good approximation ratio. For general graphs, our algorithm estimates the maximum matching in the union of the maximal matching and the b-matching, which requires using the $(1-\varepsilon)$-approximate local computation algorithm (LCA) by [144] on the subgraph formed by this union, to which we only have oracle access. We encourage readers to refer to Section 3.2.3 for more details about the techniques used there.

Additionally, for more information on the extension of the algorithm that operates in the adjacency matrix model, we recommend readers to check Section 3.2.5.

3.2.2 Algorithm for Bipartite Graphs

We begin by describing our algorithm for bipartite graphs. We focus on implementing an algorithm with a multiplicative-additive approximation guarantee. Also, we assume that we have access to the adjacency list of the graph. These assumptions will help us avoid certain complications and challenges that arise when working with general graphs, the adjacency matrix model, or when trying to obtain a multiplicative approximation guarantee. To lift these assumptions, we can leverage strong tools and methods from the literature, which, with slight modifications, can be applied here. This section contains the main novelties of our approach and proofs. Our algorithm for bipartite graphs can be seen as a translation and implementation of a two-pass streaming algorithm, which we discuss in the next subsection.

Two-Pass Streaming Algorithm for Bipartite Graphs

Our starting point is the two-pass streaming algorithm which is described in Algorithm 10. This algorithm, or its variations, has appeared in previous works on designing streaming or dynamic algorithms for maximum matching [141, 60, 32]. In words, the first pass of the algorithm only finds a maximal matching M. In the second pass, the algorithm finds a maximal b-matching B in $G[V(M), V \setminus V(M)]$, where $V(M)$ is the set of vertices matched by M. The capacities of vertices in $V(M)$ and in $V \setminus V(M)$ for the b-matching are k and kb, respectively. Moreover, in the second pass of the algorithm, when an edge (u, v) arrives in the stream, we add multiple copies of the edge to the subgraph B, as long as doing so does not violate the capacity constraints.

Algorithm 10: Two-pass Streaming Algorithm for Bipartite Graphs

Parameter: let $b = 1 + \sqrt{2}$ and k be an integer larger than $\frac{1}{b\varepsilon^3}$.
First Pass: $M \leftarrow$ maximal matching of G. ▷ Finding maximal matching
Second Pass: ▷ Finding b-matching
Let $B = \emptyset$.
for $(u, v) \in G[V(M), \overline{V(M)}]$ *where* $u \in V(M)$ **do**
  while $\deg_B(u) < k$ *and* $\deg_B(v) < \lceil kb \rceil$ **do**
    $B \leftarrow B \cup (u, v)$. ▷ We allow multi edges
return $(1 - 1/b) \cdot |M| + 1/(kb) \cdot |B|$.

Intuitively, the algorithm tries to find length-3 augmenting paths using the b-matching that it finds in the second pass. The following lemma shows the approximation guarantee of Algorithm 10.

Lemma 3.2.3 (Lemma 3.3 in [60]). *For any $\varepsilon \in (0, 1)$, the output of Algorithm 10 is a $(2 - \sqrt{2} - \varepsilon)$-approximation for maximum matching of G.*

Sublinear Time Implementation of Algorithm 10

In this section, we demonstrate how to implement a modification of Algorithm 10 in the sublinear time model, as outlined in Section 3.2.1.

Sparsification: In order to be able to use two levels of recursive oracle calls, we need to sparsify the graph. We first sample $\widetilde{O}(n\sqrt{n})$ edges and construct a maximal matching on the sampled edges to sparsify

the induced subgraph of unmatched vertices. This sparsification step is formalized in Algorithm 11. In Lemma 3.2.5, we show that if we sample enough edges, then vertices that remain unmatched after this phase have an induced degree of $\sqrt{n}$. This step is very similar to the algorithm of [68, Appendix A] for approximating a maximal matching.

Algorithm 11: Sparsification of the Induced Subgraph of Unmatched Vertices

```
1  Parameter: let c = 2√n · log n be the sparsification parameter that controls the number of edges
     that the algorithm samples.
2  M ← ∅
3  for v ∈ V do
4      if v ∉ V(M) then
5          Sample c vertices u_1, ..., u_c from N(v).
6          for i ← 1 ... c do
7              if u_i ∉ V(M) then
8                  M ← M ∪ {(v, u_i)}
9                  break;
10 return M
```

Claim 3.2.4. *Algorithm 11 runs in $\widetilde{O}(n\sqrt{n})$ time.*

Proof. For each vertex v in the graph, the algorithm samples $\widetilde{O}(\sqrt{n})$ vertices from the adjacency list of the vertex v if it is not matched by the time the algorithm processes the vertex in Line 3. Thus, the total running time is at most $\widetilde{O}(n\sqrt{n})$. □

Lemma 3.2.5. *With high probability, we have $\Delta(G[V \setminus V(M)]) \leq \sqrt{n}$.*

Proof. We will show that for every $v \in V$, the probability that $v \in V \setminus V(M)$ and the degree of v in $G[V \setminus V(M)]$ is larger than $\sqrt{n}$ is at most $1/n^2$. The lemma then follows by a union bound over all $v \in V$.

Consider the moment before v is processed. Assume that at this time, still $v \in V \setminus V(M)$ and the degree of v in $G[V \setminus V(M)]$ is larger than $\sqrt{n}$. Then, each of the c samples has a probability of at least $\sqrt{n}/n$ to be one of the unmatched neighbors, in which case v will become matched. Thus the probability that v remains unmatched after it is processed is at most

$$\left(1 - \frac{1}{\sqrt{n}}\right)^c = \left(1 - \frac{1}{\sqrt{n}}\right)^{2\sqrt{n}\log n} \leq \left(\frac{1}{e}\right)^{2\log n} = \frac{1}{n^2}.$$

□

Augmenting M using nested oracles: Now we are ready to present our sublinear algorithm. After sparsifying the graph by finding a partially maximal matching M, we try to augment M in two different ways that we have outlined in Section 3.2.1 and which are formalized in Algorithm 12. See also Figure 3.6.

For simplicity, we pretend that $kb \in \mathbb{Z}$ throughout the analysis. Since k is an arbitrarily large constant, using kb instead of $\lceil kb \rceil$ leads to an arbitrarily small error in the calculations. We also note that a maximal b-matching can be viewed as maximal matching if we duplicate vertices multiple times.

First, we try to augment the matching by designing a maximal matching oracle for $G[V \setminus V(M)]$ vertices and then another oracle for finding a b-matching between the vertices newly matched using the new oracle

and unmatched vertices. Let M' be the maximal matching of $G[V \setminus V(M)]$ that can be obtained by the oracle. We try to augment it with a b-matching B_1.

However, it is also possible that $|M'|$ is small compared to $|M|$, which implies that in the previous case, the b-matching does not help to find many augmenting paths, as the size of the maximal matching that we try to augment is too small. To account for this case, the algorithm also finds a b-matching B_2 between $V(M)$ and $V \setminus V(M)$.

Note that because the algorithm finds the initial matching M explicitly, checking whether a vertex belongs to $V(M)$ or not can be done in $O(1)$ time.

Algorithm 12: Sublinear Time Algorithm for Bipartite Graphs with Access to the Adjacency List (see Figure 3.6)

1 Run Algorithm 11 with $c = 2\sqrt{n}\log n$ and let M be its output.
2 Let $\mu_{M'}$ and μ_{B_1} be the estimate of the size of a random greedy maximal matching M' in $G[V \setminus V(M)]$ and the estimate of the size of a random greedy maximal b-matching B_1 in $G[V(M'), V \setminus V(M) \setminus V(M')]$ by running Algorithm 13. ▷ Case 1
3 Let $\mu_1 := |M| + (1 - \frac{1}{b})\mu_{M'} + \frac{1}{kb}\mu_{B_1}$. ▷ Case 1
4 Let μ_{B_2} be the estimate of the size of a random greedy maximal b-matching B_2 in $G[V(M), V \setminus V(M)]$ by running Algorithm 14. ▷ Case 2
5 Let $\mu_2 := (1 - \frac{1}{b})|M| + \frac{1}{kb}\mu_{B_2}$. ▷ Case 2
6 **return** $\max(\mu_1, \mu_2)$.

Algorithm 13: Algorithm for the First Case

1 Let $b = 1 + \sqrt{2}$ and k be an integer larger than $\frac{1}{b\varepsilon^3}$.
2 Let π be a random permutation over edges of $G[V \setminus V(M)]$ and let M' be its corresponding random greedy maximal matching.
3 Let $G_1 := G[A, B]$ where $A = V(M')$ and $B = V \setminus V(M) \setminus V(M')$.
4 Let G_1' be a bipartite graph obtained from G_1 by adding k copies of vertices in A and kb copies of vertices in B. Further, if there exists an edge between $u \in A$ and $v \in B$ in G_1, we add edges between all copies of u and v in G_1'.
5 $r \leftarrow 6\log^3 n$.
6 Run the algorithm of Proposition 2.2.4 for r random vertices and fixed permutation π in $G[V \setminus V(M)]$ and let X_i be the indicator if the i-th vertex is matched.
7 Let $X \leftarrow \sum_{i=1}^{r} X_i$ and $\mu_{M'} \leftarrow \frac{nX}{2r} - \frac{n}{2\log n}$.
8 Run nested oracles of Proposition 2.2.4 for r random vertices and fixed permutation π in G_1' and let Y_i be the indicator if the i-th vertex is matched.
9 Let $Y \leftarrow \sum_{i=1}^{r} Y_i$ and $\mu_{B_1} \leftarrow \frac{nY}{2r} - \frac{n}{2\log n}$.
10 **return** $\mu_{M'}$ and μ_{B_1}.

Algorithm 14: Algorithm for the Second Case

1 Let $b = 1 + \sqrt{2}$ and k be an integer larger than $\frac{1}{b\varepsilon^3}$.
2 Let $G_2 := G[A, B]$ where $A = V(M)$ and $B = V \setminus V(M)$.
3 Let G_2' be a bipartite graph obtained from G_2 by adding k copies of vertices in A and kb copies of vertices in B. Further, if there exists an edge between $u \in A$ and $v \in B$ in G_2, we add edges between all copies of u and v in G_2'.
4 $r \leftarrow 6 \log^3 n$.
5 Run the algorithm of Proposition 2.2.4 for r random vertices and permutations in G_2' and let Z_i be the indicator that shows if the i-th vertex is matched.
6 Let $Z \leftarrow \sum_{i=1}^{r} Z_i$ and $\mu_{B_2} \leftarrow \frac{nZ}{2r} - \frac{n}{2 \log n}$.
7 **return** μ_{B_2}.

Implementation details of the algorithm: There are some technical details in the implementation of the algorithm that are not included in the pseudocode:

- **Access to the adjacency list of an induced subgraph:** Both in Algorithm 13 and Algorithm 14, we run the algorithm of Proposition 2.2.4 for some induced subgraph of G (for example, line 5 of Algorithm 14). However, Proposition 2.2.4 works with access to the adjacency list of the input graph. To address this issue, we leverage an important property of the algorithm in Proposition 2.2.4, namely that it only needs to find a random neighbor of a given vertex at each step of its execution. Now, whenever the algorithm requires a random neighbor of vertex v in a subgraph H, it queries random neighbors in the original graph G until it finds one that belongs to H. This increases the running time of the algorithm, as it may take $\omega(1)$ time to locate a valid neighbor in H, which we will formally bound in our runtime analysis.

- **Nested oracles in line 8 of Algorithm 13:** Unlike M, we do not explicitly construct the maximal matching M' in Algorithm 13. Moreover, the edges of the subgraph G_1' connect vertices matched by M' with those that remain unmatched in either M or M'. Hence, to verify whether an edge belongs to G_1', we need to determine whether its endpoints are matched or unmatched in M' by accessing the algorithm of Proposition 2.2.4. This again increases the algorithm's runtime, which we will also formally bound in our runtime analysis.

Analysis of the Approximation Ratio

The following lemma, an analogue of Observation 3.1 in [60], substantiates the soundness of the estimates μ_1 and μ_2 produced in Algorithm 12.

Lemma 3.2.6. *Let M, M', B_1 and B_2 be as in the description of Algorithm 12. Then*

- $\mu(G) \geq \mu(M \cup M' \cup B_1) \geq |M| + (1 - \frac{1}{b})|M'| + \frac{1}{kb}|B_1|$,
- $\mu(G) \geq \mu(M \cup B_2) \geq (1 - \frac{1}{b})|M| + \frac{1}{kb}|B_2|$.

Proof. Since G is bipartite, by integrality of the bipartite matching polytope, it is enough to exhibit a fractional matching x of the appropriate value $\sum_e x_e$. For case 1, we set

$$x_e = \begin{cases} 1 & e \in M, \\ 1 - \frac{1}{b} & e \in M', \\ \frac{1}{kb} & e \in B_1. \end{cases}$$

Note that M, M' and B_1 are pairwise disjoint, and B_1 has no edge to $V(M)$. Therefore it is easy to verify that the degree constraints for x are satisfied. For case 2, we similarly set

$$x_e = \begin{cases} 1 - \frac{1}{b} & e \in M, \\ \frac{1}{kb} & e \in B_2. \end{cases}$$

□

The following lemma states the $(2 - \sqrt{2})$-approximation guarantee of the "maximal matching plus b-matching" approach obtained in prior work, for both bipartite and general graphs. We will invoke it for appropriate subgraphs of G to obtain our guarantee.

Lemma 3.2.7. *Let G' be a graph, M' be any maximal matching in G', and B be a maximal b-matching in $G'[V(M'), V(G') \setminus V(M')]$ for vertex capacities k for vertices in $V(M')$ and kb for vertices in $V(G') \setminus V(M')$, where $k > \frac{1}{b\varepsilon^3}$ and $b = 1 + \sqrt{2}$. Then:*

- *for bipartite G', we have $\mu(M' \cup B) \geq (1 - \frac{1}{b})|M'| + \frac{1}{kb}|B| \geq (2 - \sqrt{2} - \varepsilon)\mu(G')$,*
- *for general G', if B is a* random greedy *maximal b-matching, we still have $\mathbf{E}[\mu(M' \cup B)] \geq (2 - \sqrt{2} - \varepsilon)\mu(G')$.*

Proof. The first statement is the same as Lemma 3.2.3, and shown as Lemma 3.3 in [60]. The second statement is shown as Claim 5.5 in [26]. □

The following lemma is the crux of our approximation ratio analysis.

Lemma 3.2.8. *In a bipartite graph G, let M, M', B_1 and B_2 be as in the description of Algorithm 12. Then*

$$\max\left[|M| + (1 - \frac{1}{b})|M'| + \frac{1}{kb}|B_1|, (1 - \frac{1}{b})|M| + \frac{1}{kb}|B_2|\right] \geq 0.5109 \cdot \mu(G).$$

Proof. Fix a maximum matching M^* in G, and partition its edges as follows (see Figure 3.7):

- M_2^* are those with both endpoints in $V(M)$,
- $M_2^{'*}$ are those with one endpoint in $V(M)$ and the other in $V(M')$,
- $M_2^{''*}$ are those with both endpoints in $V(M')$,
- M_1^* are those with one endpoint in $V(M)$ and the other in $V \setminus V(M) \setminus V(M')$,

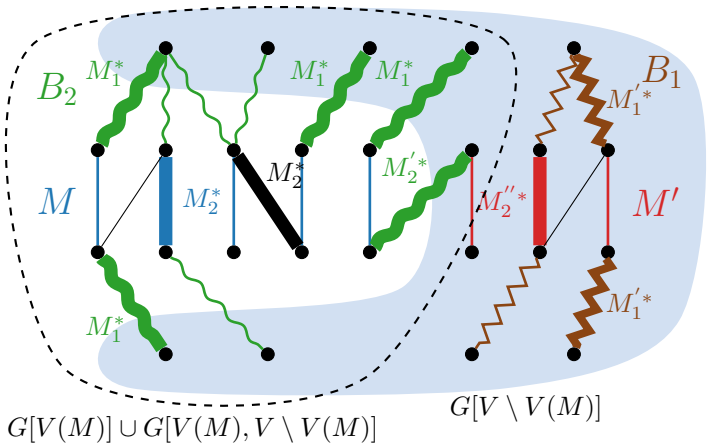


Figure 3.7: Illustration for the proof of Lemma 3.2.8. The thick edges belong to a fixed maximum matching M^*. Each of them is labeled with its partition (M_2^*, $M_2'^*$, $M_2''^*$, M_1^*, or $M_1'^*$). The two subgraphs for which we invoke Lemma 3.2.7 are marked (case 1 – highlighted in light blue, case 2 – dashed line).

- $M_1'^*$ are those with one endpoint in $V(M')$ and the other in $V \setminus V(M) \setminus V(M')$.

Since $M \cup M'$ is maximal, M^* cannot have edges with no endpoints in $V(M) \cup V(M')$. We thus have

$$\mu(G) = |M^*| = |M_1^*| + |M_1'^*| + |M_2^*| + |M_2'^*| + |M_2''^*|. \tag{3.13}$$

Also, by maximality of M^* and simple counting,

$$|M| = |M_2^*| + \frac{1}{2}|M_1^*| + \frac{1}{2}|M_2'^*|. \tag{3.14}$$

We will use Lemma 3.2.7 to analyze both cases in Algorithm 12. For case 1, we can use $G' = G[V \setminus V(M)]$; in this graph, M' is a maximal matching, and B_1 is a b-matching as in the statement of Lemma 3.2.7 (see Figure 3.7), thus we have

$$(1 - \frac{1}{b})|M'| + \frac{1}{kb}|B_1| \geq (2 - \sqrt{2} - \varepsilon)\mu(G[V \setminus V(M)]) \geq (2 - \sqrt{2} - \varepsilon)(|M_1'^*| + |M_2''^*|)$$

since $M_1'^* \cup M_2''^*$ is a matching in $G' = G[V \setminus V(M)]$. With (3.14) this gives

$$|M| + (1 - \frac{1}{b})|M'| + \frac{1}{kb}|B_1| \geq |M_2^*| + \frac{1}{2}|M_1^*| + \frac{1}{2}|M_2'^*| + (2 - \sqrt{2} - \varepsilon)(|M_1'^*| + |M_2''^*|). \tag{3.15}$$

For case 2, we can instead use $G' = G[V(M)] \cup G[V(M), V \setminus V(M)]$; in this graph, M is a maximal matching, and B_2 is a b-matching as in the statement of Lemma 3.2.7 (see Figure 3.7), thus

$$(1 - \frac{1}{b})|M| + \frac{1}{kb}|B_2| \geq (2 - \sqrt{2} - \varepsilon)\mu(G') \geq (2 - \sqrt{2} - \varepsilon)(|M_2^*| + |M_2'^*| + |M_1^*|) \tag{3.16}$$

since $M_2^* \cup M_2'^* \cup M_1^*$ is a matching in G'. Using (3.15) and (3.16), we can bound the left-hand side of this

lemma's statement as

$$\begin{aligned}\max[...] &\geq \max[|M_2^*| + \frac{1}{2}|M_1^*| + \frac{1}{2}|M_2^{'*}| + (2-\sqrt{2}-\varepsilon)(|M_1^{'*}| + |M_2^{''*}|),\\ &\qquad (2-\sqrt{2}-\varepsilon)(|M_2^*| + |M_2^{'*}| + |M_1^*|)]\\ &\geq ...\end{aligned}$$

We bound the maximum by a weighted average with weights β and $1-\beta$ for some $\beta \in [0,1]$ to be determined ($\max(x,y) \geq \beta x + (1-\beta)y$):

$$\begin{aligned}... &\geq |M_2^*|(\beta + (2-\sqrt{2}-\varepsilon)(1-\beta))\\ &\quad + (|M_1^*| + |M_2^{'*}|)\left(\frac{\beta}{2} + (2-\sqrt{2}-\varepsilon)(1-\beta)\right)\\ &\quad + (|M_1^{'*}| + |M_2^{''*}|)(2-\sqrt{2}-\varepsilon)\beta\\ &\geq (|M_2^*| + |M_1^*| + |M_2^{'*}|)\left(\frac{\beta}{2} + (2-\sqrt{2}-\varepsilon)(1-\beta)\right) + (|M_1^{'*}| + |M_2^{''*}|)(2-\sqrt{2}-\varepsilon)\beta\\ &\overset{(*)}{\geq} (|M_2^*| + |M_1^*| + |M_2^{'*}|)\gamma + (|M_1^{'*}| + |M_2^{''*}|)\gamma\\ &\overset{(3.13)}{=} \gamma \cdot \mu(G).\end{aligned}$$

We want $(*)$ to hold for some γ (the approximation ratio) as large as possible. For $(*)$ to hold, we need to satisfy:

$$\begin{aligned}\frac{\beta}{2} + (2-\sqrt{2}-\varepsilon)(1-\beta) &\geq \gamma,\\ (2-\sqrt{2}-\varepsilon)\beta &\geq \gamma.\end{aligned}$$

By solving for $\frac{\beta}{2} + (2-\sqrt{2})(1-\beta) = (2-\sqrt{2})\beta$ we get $\beta = \frac{12+2\sqrt{2}}{17}$ and $\gamma = \frac{4(5-2\sqrt{2})}{17} - O(\varepsilon) > 0.5109$. □

Lemma 3.2.9. *Let* $\max(\mu_1, \mu_2)$ *be the output of Algorithm 12. With high probability, it holds that*

$$0.5109 \cdot \mu(G) - o(n) \leq \max(\mu_1, \mu_2) \leq \mu(G).$$

The proof of Lemma 3.2.9 is routine.

Proof. By Lemma 3.2.8, we have

$$0.5109 \cdot \mu(G) \leq \max\left[|M| + (1-\frac{1}{b})\,\mathbf{E}\,|M'| + \frac{1}{kb}\,\mathbf{E}\,|B_1|, (1-\frac{1}{b})|M| + \frac{1}{kb}\,\mathbf{E}\,|B_2|\right] \leq \mu(G). \tag{3.17}$$

Let X_i, Y_i, and Z_i be as defined in Algorithm 13 and Algorithm 14. By the definition of X_i, Y_i, and Z_i we have

$$\begin{aligned}\mathbf{E}[X_i] = \Pr[X_i = 1] &= \frac{2\,\mathbf{E}\,|M'|}{n},\\ \mathbf{E}[Y_i] = \Pr[Y_i = 1] &= \frac{2\,\mathbf{E}\,|B_1|}{n},\end{aligned}$$

$$\mathbf{E}[Z_i] = \Pr[Z_i = 1] = \frac{2\,\mathbf{E}\,|B_2|}{n}.$$

Thus,

$$\begin{aligned}\mathbf{E}[X] &= \frac{2r \cdot \mathbf{E}\,|M'|}{n},\\ \mathbf{E}[Y] &= \frac{2r \cdot \mathbf{E}\,|B_1|}{n},\\ \mathbf{E}[Z] &= \frac{2r \cdot \mathbf{E}\,|B_2|}{n}.\end{aligned}$$

Then, using Chernoff bound, we obtain

$$\begin{aligned}\Pr[|X - \mathbf{E}[X]| \geq \sqrt{6\,\mathbf{E}[X]\log n}] &\leq 2\exp\left(\frac{6\,\mathbf{E}[X]\log n}{3\,\mathbf{E}[X]}\right) = \frac{2}{n^2},\\ \Pr[|Y - \mathbf{E}[Y]| \geq \sqrt{6\,\mathbf{E}[Y]\log n}] &\leq 2\exp\left(\frac{6\,\mathbf{E}[Y]\log n}{3\,\mathbf{E}[Y]}\right) = \frac{2}{n^2},\\ \Pr[|Z - \mathbf{E}[Z]| \geq \sqrt{6\,\mathbf{E}[Z]\log n}] &\leq 2\exp\left(\frac{6\,\mathbf{E}[Z]\log n}{3\,\mathbf{E}[Z]}\right) = \frac{2}{n^2}.\end{aligned}$$

Therefore, with a probability of $1 - 2/n^2$,

$$\begin{aligned}\mu_{M'} = \frac{nX}{2r} - \frac{n}{2\log n} &\in \frac{n(\mathbf{E}[X] \pm \sqrt{6\,\mathbf{E}[X]\log n})}{2r} - \frac{n}{2\log n}\\ &= \mathbf{E}\,|M'| - \frac{n}{2\log n} \pm \sqrt{\frac{3n\,\mathbf{E}\,|M'|\log n}{2r}}\\ &= \mathbf{E}\,|M'| - \frac{n}{2\log n} \pm \sqrt{\frac{n\,\mathbf{E}\,|M'|}{4\log^2 n}} && \text{(Since } r = 6\log^3 n\text{)}\\ &= \mathbf{E}\,|M'| - \frac{n}{2\log n} \pm \frac{n}{2\log n}.\end{aligned}$$

By repeating the same argument for Y and Z, we get

$$\begin{aligned}\mu_{B_1} &\in \mathbf{E}\,|B_1| - \frac{n}{2\log n} \pm \frac{n}{2\log n},\\ \mu_{B_2} &\in \mathbf{E}\,|B_2| - \frac{n}{2\log n} \pm \frac{n}{2\log n}.\end{aligned}$$

Plugging (3.17) in the bounds obtained above implies

$$\begin{aligned}\max(\mu_1, \mu_2) &= \max\left[|M| + (1 - \frac{1}{b})\mu_{M'} + \frac{1}{kb}\mu_{B_1}, (1 - \frac{1}{b})|M| + \frac{1}{kb}\mu_{B_2}\right]\\ &\leq \max\left[|M| + (1 - \frac{1}{b})\,\mathbf{E}\,|M'| + \frac{1}{kb}\,\mathbf{E}\,|B_1|, (1 - \frac{1}{b})|M| + \frac{1}{kb}\,\mathbf{E}\,|B_2|\right]\\ &\leq \mu(G).\end{aligned}$$

On the other hand, we have

$$\begin{aligned}
\max(\mu_1,\mu_2) &= \max\left[|M| + (1-\frac{1}{b})\mu_{M'} + \frac{1}{kb}\mu_{B_1}, (1-\frac{1}{b})|M| + \frac{1}{kb}\mu_{B_2}\right] \\
&\geq \max\left[|M| + (1-\frac{1}{b})(\mathbf{E}\,|M'| - \frac{n}{\log n}) + \frac{1}{kb}(\mathbf{E}\,|B_1| - \frac{n}{\log n}), (1-\frac{1}{b})|M| + \frac{1}{kb}(\mathbf{E}\,|B_2| - \frac{n}{\log n})\right] \\
&\geq \max\left[|M| + (1-\frac{1}{b})\,\mathbf{E}\,|M'| + \frac{1}{kb}\,\mathbf{E}\,|B_1|, (1-\frac{1}{b})|M| + \frac{1}{kb}\,\mathbf{E}\,|B_2|\right] - \frac{2n}{\log n} \\
&\geq 0.5109 \cdot \mu(G) - \frac{2n}{\log n}
\end{aligned}$$

which completes the proof. □

Running Time Analysis

For the analysis of the running time, we use a crucial property of random greedy maximal matching algorithm that was proved recently in [148].

Proposition 3.2.10 (Lemma 5.14 of [148])**.** *Let $Q(v)$ be the expected number of times that the oracle queries an adjacent edge of v if we start the oracle calls from a random vertex, for a random permutation over the edges of the graph G when running random greedy maximal matching. It holds that $Q(v) = \widetilde{O}(\deg_G(v)/|V(G)|)$.*

Proposition 3.2.11 (Corollary 5.15 of [148])**.** *Let $T(v)$ be the expected time needed to return a random neighbor of vertex v. Then, the expected time to run the random greedy maximal matching oracle for a random vertex and a random permutation in graph G is $\sum_{v\in V(G)} \widetilde{O}(T(v)\cdot \deg_G(v)/|V(G)|)$.*

Lemma 3.2.12. *Algorithm 13 runs in $\widetilde{O}(\bar{d}(G)\cdot\sqrt{n})$ time in expectation.*

Proof. Without loss of generality, we assume that $|V \setminus V(M)| = \Omega(n)$; otherwise, Algorithm 12 does not need to execute Algorithm 13, as the estimate from Algorithm 14 suffices.

The proof consists of two parts: first we show that line 6 of the algorithm can be implemented in $\widetilde{O}(\bar{d}(G))$ expected time, and then we prove that line 8 of the algorithm can be implemented in $\widetilde{O}(\bar{d}(G)\cdot\sqrt{n})$ expected time.

For the first part, let $G'' = G[V \setminus V(M)]$ and let v be a vertex in G''. In the adjacency list of v in the original graph, which contains $\deg_G(v)$ elements, only $\deg_{G''}(v)$ of them are neighbors in G''. Thus, to find a neighbor in G'', we need to randomly sample $T(v) = \deg_G(v)/\deg_{G''}(v)$ elements from v's adjacency list in expectation. Consequently, using Proposition 3.2.11, the expected time required to execute the random greedy maximal matching oracle for a randomly chosen vertex and permutation in G'' is

$$\begin{aligned}
\sum_{v\in V(G'')} \widetilde{O}\left(\frac{T(v)\cdot\deg_{G''}(v)}{|V(G'')|}\right) &= \sum_{v\in V(G'')} \widetilde{O}\left(\frac{(\deg_G(v)/\deg_{G''}(v))\cdot\deg_{G''}(v)}{|V(G'')|}\right) \\
&= \sum_{v\in V(G'')} \widetilde{O}\left(\frac{\deg_G(v)}{|V(G'')|}\right) = \widetilde{O}\left(\frac{|E(G)|}{|V(G'')|}\right) = \widetilde{O}\left(\frac{|E(G)|}{|V(G)|}\right) = \widetilde{O}(\bar{d}(G)).
\end{aligned}$$

In the algorithm, we choose a permutation π and run the random greedy maximal matching for $r = \widetilde{O}(1)$ randomly chosen vertices. By linearity of expectation, the expected running time of the first part is $\widetilde{O}(\bar{d}(G))$.

We prove the second part in a few steps. As a first step, for simplicity, assume that we have access to the adjacency list of G'_1 and there is no need to run nested oracles. Then, using Proposition 2.2.4, the expected running time of line 8 is bounded by $\widetilde{O}(\bar{d}(G'_1))$ for $r = \widetilde{O}(1)$ random permutations and random vertices.

Now, suppose that we have access to the adjacency list of $G'' = G[V \setminus V(M)]$ instead of G'_1. By Proposition 3.2.11, the total expected runtime of the outer (b-matching) oracle is

$$\widetilde{O}\left(\sum_{v \in G'_1} T_{\text{outer}}(v) \cdot \deg_{G'_1}(v)/|V(G'_1)|\right),$$

where $T_{\text{outer}}(v)$ is the expected time needed to find a random neighbor of v in G'_1. To find such a neighbor, we sample neighbors w of v in G'' and check whether w is matched in M' using the inner oracle. The expected number of these checks until a vertex w with $(v,w) \in E(G'_1)$ is found is $\widetilde{O}(\deg_{G''}(v)/\deg_{G'_1}(v))$. The expected cost of each check is $\mathbf{E}_{w:(v,w)\in E(G'')}[\text{cost}_{\text{inner}}(w)]$, where $\text{cost}_{\text{inner}}(w)$ is the runtime of invoking the inner oracle to check if w is matched in M'. Putting this together, the expected total runtime is

$$\begin{aligned}
\widetilde{O}\left(\frac{\sum_{v\in G'_1} T_{\text{outer}}(v)\cdot \deg_{G'_1}(v)}{|V(G'_1)|}\right) &= \widetilde{O}\left(\frac{1}{|V(G'_1)|}\sum_{v\in G'_1}\frac{\deg_{G''}(v)}{\deg_{G'_1}(v)}\cdot\frac{\sum_{w:(v,w)\in E(G'')}\text{cost}_{\text{inner}}(w)}{\deg_{G''}(v)}\cdot\deg_{G'_1}(v)\right)\\
&= \widetilde{O}\left(\frac{1}{|V(G'_1)|}\sum_{w\in G''}\text{cost}_{\text{inner}}(w)\cdot\deg_{G''}(w)\right)\\
&= \widetilde{O}\left(\sum_{w\in G''}\frac{\text{cost}_{\text{inner}}(w)}{|V(G'')|}\cdot\deg_{G''}(w)\right)\\
&\overset{Lemma\ 3.2.5}{=} \widetilde{O}\left(\sqrt{n}\cdot\mathbf{E}_{w\in G''}[\text{cost}_{\text{inner}}(w)]\right)\\
&\overset{Proposition\ 3.2.11}{=} \widetilde{O}\left(\sqrt{n}\cdot\sum_{v\in G''}\deg_{G''}(v)\cdot T_{\text{inner}}(v)\cdot\frac{1}{|V(G'')|}\right)
\end{aligned}$$

where $T_{\text{inner}}(v)$ is the expected time needed to find a random neighbor of v in G''. Under our assumption of having access to the adjacency list of G'', $T_{\text{inner}}(v) = O(1)$ and thus we can finish bounding with $\widetilde{O}(\sqrt{n}\cdot\bar{d}(G'')) = \widetilde{O}(\sqrt{n}\cdot\bar{d}(G))$.

Now we lift the assumption of having access to the adjacency list of G''. This means that, similarly as in the first part, we need to sample $T_{\text{inner}}(v) = \deg_G(v)/\deg_{G''}(v)$ vertices from the adjacency list of v in expectation (checking if a vertex is matched in M is done in $O(1)$ time). Then the total bound becomes

$$\widetilde{O}\left(\sqrt{n}\cdot\sum_{v\in G''}\deg_{G''}(v)\cdot\frac{\deg_G(v)}{\deg_{G''}(v)}\cdot\frac{1}{|V(G'')|}\right) = \widetilde{O}\left(\sqrt{n}\cdot\frac{|E(G)|}{|V(G'')|}\right) = \widetilde{O}(\sqrt{n}\cdot\bar{d}(G)).$$

Also $T_{\text{outer}}(v)$ increases by $\deg_G(v)/\deg_{G'_1}(v)$, which increases the total runtime only by

$$\widetilde{O}\left(\frac{1}{|V(G'_1)|}\sum_{v\in G'_1}\frac{\deg_G(v)}{\deg_{G'_1}(v)}\cdot\deg_{G'_1}(v)\right) = \widetilde{O}(\bar{d}(G)).$$

□

Lemma 3.2.13. *Algorithm 14 runs in $\widetilde{O}(\bar{d}(G))$ time in expectation.*

Proof. It is enough to show that it takes $\widetilde{O}(\bar{d}(G))$ time to run the algorithm of Proposition 2.2.4 (the b-matching oracle) for each sampled vertex and permutation, as we have $r = \widetilde{O}(1)$. We repeat the same arguments as in the proof of Lemma 3.2.12.

Let v be a vertex in the graph $G_2' = G[V(M), V \setminus V(M)]$. Note that in the adjacency list of v in the original graph, which contains $\deg_G(v)$ elements, only $\deg_{G_2'}(v)$ of the elements are neighbors in G_2'. Thus, we need to randomly sample $T(v) = \deg_G(v)/\deg_{G_2'}(v)$ elements of the adjacency list of v to find a neighbor in G_2'. Recall that we can check whether a vertex is matched in M in $O(1)$ time. Therefore, using Proposition 3.2.11, the expected time needed to run the random greedy maximal matching for a random vertex and permutation in G_2' is

$$\sum_{v \in V(G_2')} \widetilde{O}\left(\frac{T(v) \cdot \deg_{G_2'}(v)}{|V(G_2')|}\right) = \sum_{v \in V} \widetilde{O}\left(\frac{(\deg_G(v)/\deg_{G_2'}(v)) \cdot \deg_{G_2'}(v)}{n}\right)$$
$$= \sum_{v \in V} \widetilde{O}\left(\frac{\deg_G(v)}{n}\right) = \widetilde{O}\left(\frac{|E(G)|}{n}\right) = \widetilde{O}(\bar{d}(G)).$$

□

Lemma 3.2.14. *Algorithm 12 runs in $\widetilde{O}(n\sqrt{n})$ time with high probability.*

Proof. By Claim 3.2.4, the sparsification step takes $\widetilde{O}(n\sqrt{n})$ time. Also, by Lemmas 3.2.12 and 3.2.13, Algorithms 13 and 14 run in $\widetilde{O}(\bar{d}(G) \cdot \sqrt{n})$ and $\widetilde{O}(\bar{d}(G))$ expected time, respectively. To establish a high probability bound on the time complexity, we execute $O(\log n)$ instances of the algorithm in parallel and halt as soon as the first one completes. By applying Markov's inequality, we deduce that each individual instance terminates within $\widetilde{O}(n\sqrt{n})$ time with constant probability. As a result, with high probability, at least one of these instances finishes within $\widetilde{O}(n\sqrt{n})$ time. This concludes the proof. □

Now we are ready to prove the final theorem of this section.

Theorem 3.2.15. *There exists an algorithm that, given access to the adjacency list of a bipartite graph, estimates the size of the maximum matching with a multiplicative-additive approximation factor of $(0.5109, o(n))$ and runs in $\widetilde{O}(n\sqrt{n})$ time with high probability.*

Proof. By Lemma 3.2.9, Algorithm 12 achieves multiplicative-additive approximation of $(0.5109, o(n))$. Moreover, the running time of Algorithm 12 is $\widetilde{O}(n\sqrt{n})$ with high probability by Lemma 3.2.14. □

3.2.3 Algorithm for General Graphs

The following is an analogue of Lemma 3.2.8 for general graphs.

Lemma 3.2.16. *In a general graph G, let M, M', B_1 and B_2 be as in the description of Algorithm 12. Then*

$$(1-\varepsilon)\max\left[|M| + \mathbf{E}[\mu(M' \cup B_1)], \mathbf{E}[\mu(M \cup B_2)]\right] \geq 0.5109 \cdot \mu(G).$$

Proof. The proof proceeds as that of Lemma 3.2.8, with the following modifications:

- we invoke the second rather than the first part of Lemma 3.2.7 to obtain analogues of inequalities (3.15) and (3.16), i.e., instead of lower-bounding $|M| + (1 - \frac{1}{b})|M'| + \frac{1}{kb}|B_1|$, we lower-bound $|M| + \mathbf{E}[\mu(M' \cup B_1)]$, and instead of lower-bounding $(1 - \frac{1}{b})|M| + \frac{1}{kb}|B_2|$, we lower-bound $\mathbf{E}[\mu(M \cup B_2)]$,
- the $(1 - \varepsilon)$ factor on the left-hand side of the statement can be folded into the $O(\varepsilon)$ term in γ at the end of the proof, for small enough ε.

□

In this section, we show how to extend our algorithm to work for general graphs. The main difference between bipartite and general graphs is that the estimate based on the sizes of $|M|$, $|M'|$, $|B_1|$, and $|B_2|$ is insufficient to achieve a 0.5109 approximation guarantee. In Lemma 3.2.16, we show that we can achieve this approximation ratio by estimating $\mu(M \cup B_2)$ and $\mu(M' \cup B_1)$. More formally, to produce μ_1 and μ_2 in Algorithm 12, we use $|M| + \mu(M' \cup B_1)$ and $\mu(M \cup B_2)$, respectively. Also, Algorithm 13 and Algorithm 14 output $\mu(M' \cup B_1)$ and $\mu(M \cup B_2)$, respectively.

In both Algorithm 13 and Algorithm 14 we have access to oracles that can return whether a vertex is matched in M', B_1, or B_2 for a fixed permutation π. These oracles can also be used to return the edge of the matching if the vertex is matched, which is a corollary of Proposition 2.2.4 since the algorithm of Proposition 2.2.4 can be also used to return the edge of the matching.

Lemma 3.2.17. *For a vertex v, there exists an algorithm that returns the edges of v in M' and B_1 in $\widetilde{O}(\bar{d}(G) \cdot \sqrt{n})$ expected time.*

Proof. The proof follows from Lemma 3.2.12 and the fact that the algorithm in Proposition 2.2.4 can return the edge of the matched vertex in the same running time. □

Lemma 3.2.18. *For a vertex v, there exists an algorithm that returns the edges of v in M and B_2 in $\widetilde{O}(\bar{d}(G))$ expected time.*

Proof. If v is matched in M, we can return the edge in $O(1)$ time since we have access to this maximal matching. The rest of the proof follows from Lemma 3.2.13 and the fact that the algorithm in Proposition 2.2.4 can return the edge of the matched vertex in the same running time. □

Local computation algorithms (LCA) is a model of computation, also motivated by large data sets, in which the algorithm is not expected to produce the entire output at once. Instead, the algorithm is queried for parts of the output, and must produce a consistent and approximately optimal output. We use the following local computation algorithm (LCA) by [144] to design our algorithm.

Proposition 3.2.19 ([144])**.** *There exists a $(1 - \varepsilon)$-approximate local computation algorithm for maximum matching of graph G in $\widetilde{O}(\Delta(G)^{1/\varepsilon^2})$ time with access to the adjacency list of G.*

Now we prove the main technical part of this section that can be used to estimate $\mu(M' \cup B_1)$ and $\mu(M \cup B_2)$.

Lemma 3.2.20. *Let H be a subgraph of graph G. Suppose that H is the union of a constant number of random greedy maximal matching on different subsets of vertices. Also, we have oracle access to edges of random greedy maximal matching. We can query a vertex to obtain the matching edge of vertex v in $\widetilde{O}(T)$ expected time. Moreover, the maximum degree of H is constant. Then, there exists a $(1 - \varepsilon)$-approximate algorithm that estimates the size of the maximum matching of H in $\widetilde{O}(T)$ expected time.*

Proof. We run the algorithm of Proposition 3.2.19. Each time the algorithm visits a new vertex u, we first query all the constant number of oracles for random greedy maximal matching to get the adjacency list of u in H. If the algorithm in Proposition 3.2.19 made uniform queries to the oracle, then we could conclude the proof since $\Delta(H)$ and ε are constant. However, note that queries to the oracles are not independent and we have an expected time guarantee on the running time of the oracles. So it is possible that the way that the algorithm of Proposition 3.2.19 works might result in making biased queries to oracles.

Let $\mathcal{L}(u,\pi)$ be the set of vertices the algorithm of Proposition 3.2.19 visits when we run the algorithm for vertex u and we use permutation π for random greedy maximal matchings of the subgraph H. Let $F(u,\pi)$ be the running time of the oracle for vertex u and permutation π. Also, let $L(u,\pi)$ be the running time of the algorithm of Proposition 3.2.19 on vertex u using permutation π for random greedy maximal matchings. Hence, we have

$$\mathbf{E}_{u,\pi}[F'(u,\pi)] = \sum_{\pi}\sum_{u}\sum_{v\in\mathcal{L}(u,\pi)} \frac{\mathbf{E}[F(v,\pi)]}{n\cdot|E(G)|!}$$

Further, the algorithm in Proposition 3.2.19 explores the local neighborhood of a vertex to answer each query. Therefore, if two vertices w and z are at a distance greater than $\Delta(H)^{1/\varepsilon^3}$, the algorithm does not visit z when answering a query for w. Thus,

$$\begin{aligned}
\mathbf{E}_{u,\pi}[F'(u,\pi)] &\le \sum_{\pi}\sum_{v} \Delta(H)^{\left(\Delta(H)^{1/\varepsilon^3}\right)}\cdot\frac{\mathbf{E}[F(v,\pi)]}{n\cdot|E(G)|!}\\
&= \Delta(H)^{\left(\Delta(H)^{1/\varepsilon^3}\right)}\cdot\sum_{\pi}\sum_{v}\frac{\mathbf{E}[F(v,\pi)]}{n\cdot|E(G)|!}\\
&\le \widetilde{O}(T),
\end{aligned}$$

where the first inequality follows by the fact that each vertex has at most $\Delta(H)^{\left(\Delta(H)^{1/\varepsilon^3}\right)}$ neighbors in H within a distance of $\Delta(H)^{1/\varepsilon^3}$, and the last inequality because $\Delta(H)$ and ε are constants. Therefore, even with the biased queries, we can implement Proposition 3.2.19 on subgraph H in $\widetilde{O}(T)$ time, which completes the proof. □

Lemma 3.2.21. *There exists an algorithm that outputs a $(1-\varepsilon)$-approximate estimation of the value of $\mu(M\cup B_2)$ in $\widetilde{O}(\bar{d}(G))$ expected time.*

Proof. The proof follows by plugging Lemma 3.2.18 into Lemma 3.2.20 and running the algorithm for r samples. □

Lemma 3.2.22. *There exists an algorithm that outputs a $(1-\varepsilon)$-approximate estimation of the value of $\mu(M'\cup B_1)$ in $\widetilde{O}(\bar{d}(G)\cdot\sqrt{n})$ expected time.*

Proof. The proof follows by plugging Lemma 3.2.17 into Lemma 3.2.20 and running the algorithm for r samples. □

Theorem 3.2.23. *There exists an algorithm that, given access to the adjacency list of a graph, estimates the size of the maximum matching with a multiplicative-additive approximation factor of $(0.5109, o(n))$ and runs in $\widetilde{O}(n\sqrt{n})$ time with high probability.*

Proof. By Lemma 3.2.16 and a Chernoff bound similar to Lemma 3.2.9, we achieve a multiplicative-additive approximation of $(0.5109, o(n))$. Moreover, the expected running time is $\widetilde{O}(n\sqrt{n})$ by Claim 3.2.4, Lemma 3.2.21, and Lemma 3.2.22. To establish a high probability bound on the time complexity, we execute $O(\log n)$ instances of the algorithm in parallel and halt as soon as the first one completes. By applying Markov's inequality, we deduce that each individual instance terminates within $O(n\sqrt{n})$ time with constant probability. As a result, with high probability, at least one of these instances finishes within $O(n\sqrt{n})$ time. This concludes the proof. □

3.2.4 Multiplicative Approximation

In this section, we show that we can achieve a multiplicative approximation guarantee by slightly increasing the number of samples in Algorithm 13 and Algorithm 14. First, we prove a simple lower bound for the size of the maximum matching of a graph based on its maximum and average degree.

Claim 3.2.24. *For any graph G, it holds that $\mu(G) \geq n\bar{d}(G)/(4\Delta(G))$.*

Proof. Any graph with a maximum degree of $\Delta(G)$ can be colored greedily using $2\Delta(G)$ colors. Furthermore, the edges of each color create a matching. Thus, we have $\mu(G) \geq |E(G)|/(2\Delta(G))$. Combining with $|E(G)| = n\bar{d}(G)/2$, we can conclude the proof. □

The goal is to obtain a multiplicative approximation guarantee of $(0.5109 - \varepsilon)\mu(G)$. It is important to note that if any of $|M|$, $|M'|$, $|B_1|$, or $|B_2|$ is not a constant fraction of the others, it can be omitted from the equation in the statement of Lemma 3.2.8 without affecting the approximation by more than a function of ε. Thus, without loss of generality, we can assume that $|M| = \Omega(\mu(G))$, $|M'| = \Omega(\mu(G))$, $|B_1| = \Omega(\mu(G))$, and $|B_2| = \Omega(\mu(G))$. Consequently, using Claim 3.2.24 and as an application of Chernoff bound, we can use $\widetilde{O}_\varepsilon(\Delta(G)/\bar{d}(G))$ samples in Algorithm 13 and Algorithm 14 to obtain multiplicative estimation of $\mu_{M'}$, μ_{B_1}, and μ_{B_2}.

By Lemma 3.2.12, Algorithm 13 runs in $\widetilde{O}(\bar{d}(G)\cdot\sqrt{n})$ time when we have $r = \widetilde{O}(1)$ samples. By increasing the number of samples to $\widetilde{O}_\varepsilon(\Delta(G)/\bar{d}(G))$, the running time of Algorithm 13 increases to $\widetilde{O}(\Delta(G)\cdot\sqrt{n})$. Moreover, by Lemma 3.2.13, Algorithm 14 runs in $\widetilde{O}(\bar{d}(G))$ time when we have $r = \widetilde{O}(1)$ samples. By increasing the number of samples to $\widetilde{O}_\varepsilon(\Delta(G)/\bar{d}(G))$, the running time of Algorithm 14 increases to $\widetilde{O}(\Delta(G))$. Therefore, the total running time of the algorithm is within $\widetilde{O}(n\sqrt{n})$.

Finally, we can obtain the degree of each vertex in the graph using binary search. Therefore, we can assume that we have access to $\Delta(G)$ and $\bar{d}(G)$ by spending $\widetilde{O}(n)$ time. Thus we get:

Theorem 3.2.1. *There exists an algorithm that, given access to the adjacency list of a graph, estimates the size of the maximum matching with a multiplicative approximation factor of* 0.5109 *and runs in $\widetilde{O}(n\sqrt{n})$ time with high probability.*

3.2.5 Algorithm with Access to the Adjacency Matrix

In this section, we use a simple reduction to show that with a small modification, our algorithm can be adapted to the setting where we have access to the graph's adjacency matrix. A slightly different version of this kind of reduction appeared in previous works on sublinear time algorithms for maximum matching [34, 43].

It is important to note that obtaining a constant-factor multiplicative approximation is impossible when the algorithm only has access to the adjacency matrix of the graph. This is because if the graph is guaranteed to contain either a single edge or be completely empty, any algorithm would require $\Omega(n^2)$ adjacency matrix queries to distinguish between these two cases. Consequently, we allow the algorithm to have an additive error of $o(n)$ in addition to the multiplicative approximation ratio.

We build an auxiliary graph H with the following vertex set and edge set:

- **Vertex set:** $V(H)$ contains $n+2$ disjoint sets of n vertices V_1, V_2, and $U_1, \ldots, U_n$. Each V_i is a copy of the vertices of the original graph. Each U_i contains $n \log^2 n$ vertices.
- **Edge set:** For each vertex $v \in V_1$, the i-th neighbor of v is the i-th vertex in V_1 if $(v, i) \in E(G)$, and otherwise it is the i-th vertex in V_2. Similarly, for each vertex $v \in V_2$, the i-th neighbor of v is i-th vertex in V_2 if $(v, i) \in E(G)$, and otherwise it is the i-th vertex in V_1. Also, each $v \in V_2$ is connected to all vertices of U_v. As a result, the degree of each vertex in $U_1 \cup U_2 \cup \ldots \cup U_n$ is 1, the degree of each vertex in V_2 is n, and the degree of each vertex in V_2 is $n + n \log^2 n$. Therefore, we have $\Delta(H) = \widetilde{O}(n)$.

Because of the way we constructed the graph, it is not hard to see that we can find the i-th neighbor of the adjacency list of each vertex in H using only a single query to the adjacency matrix of G.

Observation 3.2.25. *For each vertex v in graph H, the i-th neighbor of H can be found using at most a single adjacency matrix query in G.*

Proof. If $i > \deg_H(v)$, we can simply certify this since the degree of each vertex in H is determined by the vertex set to which v belongs. If $v \in U_j$, then the only neighbor of v is j in V_2. If $v \in V_1$, we make an adjacency matrix query for (v, i), and based on the result, the i-th neighbor is either vertex i in V_1 or in V_2. A similar procedure applies for $v \in V_2$ when $i \leq n$. If $i > n$ for $v \in V_2$, we return the $(i-n)$-th vertex in U_v. □

Modification to the algorithm: Since the graph contains $\widetilde{\Theta}(n^2)$ vertices, we cannot afford to apply the sparsification step to all vertices. However, vertices in $U_1 \cup \ldots \cup U_n$ have degree 1. Therefore, we apply the sparsification step only to vertices in V_1 and V_2. Since we have $|V_1| + |V_2| = 2n$, we can apply the sparsification for these sets in $\widetilde{O}(n\sqrt{n})$ time. We first iterate over the vertices in V_2 and apply the sparsification step, and then we apply it to the vertices in V_1. This ordering ensures that most vertices in V_2 get matched to vertices in $U_1 \cup \ldots \cup U_n$ in this step, which is desirable for our application.

Claim 3.2.26. *After the sparsification step, each vertex in V_2 is matched by M with high probability. Moreover, at most $n/\log n$ vertices in V_2 are matched to vertices in $V_1 \cup V_2$ with high probability.*

Proof. Note that we first process the vertices in V_2. At the time we process a vertex $v \in V_2$, it has at least $n \log^2 n$ neighbors due to its neighbors in U_v. Since, after sparsification, each vertex must have a degree of $\widetilde{O}(\sqrt{n})$ with high probability by Lemma 3.2.5, all vertices in V_2 will be matched with high probability.

Additionally, at the time we process a vertex $v \in V_2$, it has $n \log^2 n$ unmatched neighbors in U_v. On the other hand, it has at most n neighbors in the rest of the graph. Thus, with a probability of at most $1/\log^2 n$, it gets matched to a vertex outside U_v. Therefore, in expectation, at most $n/\log^2 n$ vertices in V_2 are matched to vertices in $V_1 \cup V_2$. Using the Chernoff bound, we can conclude that at most $n/\log n$ vertices in V_2 are matched to vertices in $V_1 \cup V_2$ with high probability. □

Equipped with this reduction, we can now simply run the rest of the algorithm for vertices in V_1. The only difference is that we exclude the edges of M that lie between V_2 and $U_1 \cup \ldots \cup U_n$ in the estimation. Additionally, in the final estimation, the algorithm returns the previous estimate minus $n/\log n$, accounting for the vertices in V_2 that are not matched within $V_1 \cup V_2$, which introduces an additional $o(n)$ additive error. Thus we obtain:

Theorem 3.2.2. *There exists an algorithm that, given access to the adjacency matrix of a graph, estimates the size of the maximum matching with a multiplicative-additive approximation factor of* $(0.5109, o(n))$ *and runs in* $\widetilde{O}(n\sqrt{n})$ *time with high probability.*

3.3 A Slightly Better Than 2/3-Approximation Algorithm Sublinear Time

In this section, we present two algorithms that run in strictly sublinear time of $n^{2-\Omega(1)}$ with larger approximation ratio. Our first result is an algorithm that works on general (i.e., not necessarily bipartite) graphs and obtains an (almost) 2/3-approximation. This significantly improves prior close-to-1/2 approximations of previous sections and result of [34].

Theorem 3.3.1. *For any fixed* $\varepsilon > 0$*, there are algorithms for approximating the maximum matching size of any (general) n-vertex graph that take* $n^{2-\Omega_\varepsilon(1)}$ *time and obtain*

- *a multiplicative* $(2/3 - \varepsilon)$*-approximation in the adjacency list model, and*
- *a multiplicative-additive* $(2/3 - \varepsilon, o(n))$*-approximation in the adjacency matrix model.*

Our next result gives a subquadratic time algorithm that indeed breaks 2/3-approximation provided that the input graph is bipartite.

Theorem 3.3.2. *There are algorithms for approximating the maximum matching size of any bipartite n-vertex graph that take* $n^{2-\Omega(1)}$ *time and obtain*

- *a multiplicative* $(2/3 + \Omega(1))$*-approximation in the adjacency list model, and*
- *a multiplicative-additive* $(2/3 + \Omega(1), o(n))$*-approximation in the adjacency matrix model.*

3.3.1 Technical Overview

An (Almost) 2/3-Approximation via EDCS: The *edge-degree constrained subgraph* (EDCS) introduced by Bernstein and Stein [51, 50] has been a powerful tool in obtaining an (almost) 2/3-approximation of maximum matching in various settings including dynamic algorithms [51, 50], communication complexity [16], stochastic matchings [16], and (random order) streaming [47, 15]. In this work, we use it for the first time in the sublinear time model. In particular, both Theorems 3.3.1 and 3.3.2 build on EDCS.

For a parameter β (think of it as a large constant), a subgraph H of a graph G is a β-EDCS of G if $(P1)$ all edges (u, v) in H satisfy $\deg_H(u) + \deg_H(v) \leq \beta$ and $(P2)$ all edges $(u, v) \in G \setminus H$ satisfy $\deg_H(v) + \deg_H(v) \geq (1-\varepsilon)\beta$. The main property of EDCS, proved first by [51, 50] and further strengethened

in [16, 33], is that for any $\beta = \Omega(1/\varepsilon)$, any β-EDCS includes a $(1-O(\varepsilon))2/3$ approximate maximum matching of its base graph G.

It is not hard to see that *finding* any β-EDCS of the whole input graph G requires $\Omega(n^2)$ queries.[5] Therefore, instead, we first sub-sample some $p = 1/n^\delta$ fraction of the edges (or pairs in the adjacency matrix model) of G for some fixed $\delta > 0$ in $O(n^2 p) = n^{2-\Omega(1)}$ time and construct an EDCS H over those edges. Building on an approach of Bernstein [47] in the random-order streaming setting, this can be done in a way such that at most $\widetilde{O}(\mu(G)/p) = n^{2-\Omega(1)}$ edges in the whole graph G remain *underfull*, i.e., those that do not satisfy property $(P2)$. The union of the set U of underfull edges and the EDCS H can be shown to include an (almost) 2/3-approximation. The next challenge is that we do not have the set U. However, to estimate the maximum matching of $H \cup U$, it suffices to design an oracle that upon querying a vertex v, determines whether it belongs to an approximately optimal maximum matching of $H \cup U$ and run this oracle on a few randomly sampled vertices. To do so, we build on a local computation algorithm (LCA) of Levi, Rubinfeld, and Yodpinyanee [144] (which itself builds on a result of Yoshida, Yamamoto, and Ito [176]) that takes $\text{poly}_\varepsilon(\Delta)$ time to return if a given vertex v belongs to some $(1-\varepsilon)$-approximate matching of its input graph, where here Δ is the maximum degree. Modifying $H \cup U$ by getting rid of its high-degree vertices, we show the algorithm of [144] can be used to $(1-\varepsilon)$-approximate the size of the maximum matching of $H \cup U$ in $O(n^{1+\delta/\varepsilon^2})$ time. Picking δ sufficiently small, we arrive at an algorithm that obtains a $(2/3-\varepsilon)$-approximation in $n^{2-\Omega_\varepsilon(1)}$ time.

Going Beyond 2/3-Approximations: It is known that the (almost) 2/3-approximation analysis for EDCS is tight. That is, there are instances on which the EDCS does not include a better than 2/3-approximation. However, a characterization of such tight instances of EDCS was recently given in the work of Behnezhad [32] that we use in beating 2/3-approximation. Consider a β-EDCS H, for sufficiently large constant β, that does not include a strictly better than 2/3 approximation of $(2/3+\delta)$-approximation for some small $\delta > 0$. The characterization divides the vertices into two subsets V_{mid} and V_{low} depending on their degrees in H. Then shows that there must be an (almost) 2/3-approximate matching M in the induced bipartite subgraph $G[V_{mid}, V_{low}]$ that is far from being maximal. Namely, it leaves an (almost) 1/3-approximate matching in $G[V \setminus M]$ that can be directly added to M. Interestingly, this characterization coincides with our lower bound construction! The set V_{mid}, here, corresponds to the set $A \cup B$ in the lower bound and the set V_{low} corresponds to S. This essentially reduces the problem to showing that our lower bound construction can indeed be solved in $n^{2-\Omega(1)}$ time for all choices of the degree d.

Here we show why our lower bound construction fundamentally cannot lead to a better than $n\sqrt{n}$ time lower bound. Indeed the ideas behind our algorithm for Theorem 3.3.2 are very close. First, suppose that $d < \sqrt{n}$. In this case, we can random sample a vertex in A and examine the A/B value of each of its d neighbors in total $O(nd) = O(n\sqrt{n})$ time to see if it has any A neighbors, thereby solving the instance. So let us suppose that $d \geq \sqrt{n}$. Now in this case, we can first sample a vertex v in A and list all of its $A \cup B$ neighbors in $O(n)$ time. Since the vast majority (i.e., all but one) of the neighbors of v go to B, we have essentially found d vertices in B in $O(n)$ time. Repeating this $\Theta(n/d)$ times, we find all of the B vertices in total $O(n^2/d) = O(n\sqrt{n})$ time. We note that the final running time of our algorithm in Theorem 3.3.2 is only $n^{2-\Omega_\varepsilon(1)}$ and not $O(n\sqrt{n})$ since the bottleneck is in finding the EDCS and estimating the maximum

[5] In fact, finding the edges of any constant approximate matching requires $\Omega(n^2)$ time. Our focus is on estimating only the size of the maximum matching.

matching size of $H \cup U$ as discussed above. For a more detailed high level overview of the algorithm, see Section 3.3.3.

3.3.2 An (Almost) 2/3-Approximation

Throughout this section, we introduce an algorithm that achieves an almost 2/3 approximation in sublinear time in both the adjacency list and matrix models, thereby proving Theorem 3.3.1. Specifically, we prove the following theorems, which further formalize Theorem 3.3.1 of the introduction.

Theorem 3.3.3. *For any constant $\varepsilon > 0$, there exists an algorithm that estimates the size of the maximum matching in $\widetilde{O}_\varepsilon(n^{2-\varepsilon^3})$ time up to a multiplicative-additive factor of $(\frac{2}{3} - \varepsilon, o(n))$ with high probability in both adjacency matrix and adjacency list model.*

Theorem 3.3.4. *For any constant $\varepsilon > 0$, there exists an algorithm that estimates the size of the maximum matching in $\widetilde{O}_\varepsilon(n^{2-\varepsilon^3})$ time up to a multiplicative factor of $(\frac{2}{3} - \varepsilon)$ with high probability in the adjacency list model.*

One key component of our algorithm is the *edge-degree constrained subgraph* (EDCS) of Bernstein and Stein [51]. An EDCS is a maximum matching sparsifier that has been used extensively in the literature on maximum matching in different settings such as dynamic and streaming. It is known that an EDCS contains an almost 2/3-approximation of the maximum matching of the original graph. Formally, we can define EDCS as follows:

Definition 3.3.5 (EDCS)**.** *For any $\lambda < 1$ and $\beta \geq 2$, subgraph H is a $(\beta, (1-\lambda)\beta)$-EDCS of G if*

- *(Property P1:) for all edges $(u, v) \in H$, $\deg_H(u) + \deg_H(v) \leq \beta$, and*
- *(Property P2:) for all edges $(u, v) \in G \setminus H$, $\deg_H(u) + \deg_H(v) \geq (1-\lambda)\beta$.*

We use a more relaxed version of EDCS in our algorithm that first appeared in the literature on random-order streaming matching [47]. Let $H = (V, E_H)$ be a subgraph of G such that it satisfies the first property of the EDCS (P1) in Definition 3.3.5 for some $\beta = \widetilde{O}(1)$, and E_U be the set of all edges of G that violate the second property of EDCS (P2). It is possible to show that the union of E_H and E_U contains an almost 2/3-approximation of the maximum matching of the original graph.

Definition 3.3.6 (Bounded edge-degree)**.** *A graph H has a bounded edge-degree β if for each edge $(u, v) \in H$, we have $\deg_H(u) + \deg_H(v) \leq \beta$.*

Proposition 3.3.7 (Lemma 3.1 of [47], (Relaxed EDCS))**.** *Let $\varepsilon \in [0, 1/2)$ and λ, β be parameters such that $\lambda \leq \frac{\varepsilon}{128}$, $\beta \geq \frac{16\log(1/\lambda)}{\lambda^2}$. Let $H = (V, E_H)$ be a subgraph of G with bounded edge-degree β. Furthermore, let $\widetilde{H} = (V, E_U)$ be a subgraph that contains all edges (u, v) in G such that $\deg_H(u) + \deg_H(v) < (1-\lambda)\beta$. Then we have $\mu(H \cup \widetilde{H}) \geq (3/2 - \varepsilon) \cdot \mu(G)$.*

An Informal Overview of the Algorithm: In the next few paragraphs, we give an informal overview so the readers know what to expect. Algorithm 15 consists of two main parts. Let H initially be an empty subgraph. First, in $n\beta^2 + 1$ rounds (β is the parameter of EDCS), we sample $n^{1-\varepsilon^3}$ pairs of vertices in the graph (in the adjacency list model, we sample δ edges). After sampling in each round, we update the

Table 3.1: Subgraphs considered by algorithms in Sections 3.3.2 and 3.3.3.

Variable	Definition	Intuition
H	See Algorithm 15	Relaxed EDCS
$G^{Unsampled}$	Edges not sampled by Algorithm 15	$\mu(G^{Unsampled}) \geq (1-2\varepsilon)\mu(G)$
U	See Algorithm 15	Unsampled low H-degree edges (violating P1)
G'	$G' = (V, E_H \cup E_U)$	We can estimate $\mu(G')$, and also $\mu(G')$ is a 2/3-approx of $\mu(G)$
G''	$G'' = G'[V_{low}, V_{mid}]$	Subgraph that contains 2/3-approx matching and is far from being maximal
$G[A]$	$A = V_{mid} \setminus V(M)$	Edges in $G[A]$ can augment $\mu(G'')$
$G[V_{mid} \setminus V(M^i_{AB})]$	M^i_{AB} defined in Algorithm 17	$G[V_{mid} \setminus V(M^i_{AB})]$ is non-trivially sparse
$G_{M_{AB}}$	$G_{M_{AB}} = (V_{mid}, \bigcup_i^k M^i_{AB})$ defined in Algorithm 17	Helps to augment G'' or remove vertices of $V_{mid} \setminus A$ in case 3

bounded edge-degree subgraph H. For each edge (u, v), if the $\deg_H(u) + \deg_H(v) < (1 - \lambda)\beta$, we add it to H. This change can affect the bounded edge-degree property of H. In order to avoid this issue, we iterate over incident edges of (u, v) to remove edges with degree more than β. Note that at most one neighbor of each u and v will be deleted after this iteration. Since $\beta = \widetilde{O}(1)$, this part can be done in $\widetilde{O}(n^{2-\varepsilon^3})$ time. Furthermore, since we have chosen enough random edges, the number of *unsampled* edges in the graph that violate Property (P2) of the EDCS is relatively small. Note that when we remove edges to fix Property (P1), it is possible that an edge that was previously sampled but rejected due to the high H-degree, now has a lower H-degree and thus violates Property (P2). However, in the analysis, it will be sufficient to consider only unsampled edges that violate Property P2. More specifically, we show that the total number of unsampled violating edges is $\widetilde{O}(n^{1+\varepsilon^3})$.

Note that the union of approximate EDCS H and unsampled violating edges preserves an almost 2/3-approximate maximum matching of the original graph since most of the edges of the graph are unsampled and the unsampled subgraph itself, contains a large matching. Hence, we can apply Proposition 3.3.7 to the union of H and all unsampled violating edges. Also, graph $G' = (V, E_H, E_U)$ has a low average degree. Hence, it remains to estimate a $(1 - \varepsilon)$-approximate matching of graph G' that has a low average degree. For this part, we use the $\widetilde{O}_\varepsilon(\Delta^{1/\varepsilon^2})$ time *local computation algorithms* (LCA) of Levi, Rubinfeld, and Yodpinyanee [144] which itself builds on the sublinear time algorithm of Yoshida, Yamamoto, and Ito [176]. In local computation algorithms, the goal is to compute a queried part of the output in sublinear time. One challenge here is that we do not have access to the adjacency list of graph G' to use the algorithm of [144] as a black box. We slightly modify this algorithm to run in $\widetilde{O}_\varepsilon(n\Delta^{1/\varepsilon^2})$ with access to the adjacency matrix or the adjacency list of the original graph G. Furthermore, since this algorithm works with the maximum degree, we eliminate some high-degree vertices by losing an additive error. In what follows, we formalize the intuition given in previous paragraphs.

Proposition 3.3.8 ([144])**.** *There exists a randomized $(1 - \varepsilon)$-approximation local computation algorithm for maximum matching with running time $\widetilde{O}_\varepsilon(\Delta^{1/\varepsilon^2})$ using access to adjacency list.*

Lemma 3.3.9. *Let G' be a subgraph of graph G. Also, let $\mathcal{A}$ be a local computation algorithm for maximum matching in G' with running time $O(T)$ using access to the adjacency list of G'. There exists an algorithm with exactly the same approximation ratio that runs in $O(nT)$ time with access to the adjacency matrix or the adjacency list of G.*

Proof. Since algorithm $\mathcal{A}$ runs in $O(T)$ time, it visits at most $O(T)$ vertices in the graph. For each of these vertices, we can simply query all their edges in G' using $O(n)$ time having access to the adjacency matrix or the adjacency list of G. Therefore, for each vertex that $\mathcal{A}$ visits, we can construct the adjacency list of the vertex by spending $O(n)$ time. □

Corollary 3.3.10. *Let G' be a subgraph with maximum degree Δ of graph G. There exists a local computation algorithm for that for a given vertex v, determines if it is in the output of a $(1 - \varepsilon)$-approximate maximum matching of G' with a running time $\widetilde{O}_\varepsilon(n\Delta^{1/\varepsilon^2})$ with access to the adjacency matrix or the adjacency list of G.*

Proof. The proof follows by combining Proposition 3.3.8 and Lemma 3.3.9. □

Algorithm 15: A 2/3-Approximate Matching Algorithm in Adjacency List and Matrix

1 **Parameter:** ε.
2 $\delta \leftarrow n^{1-\varepsilon^3}$, $H \leftarrow \emptyset$, $\lambda \leftarrow \frac{\varepsilon}{128}$, $\beta \leftarrow \frac{16\log(1/\lambda)}{\lambda^2}$, $T \leftarrow n\beta^2+1$, $r \leftarrow 36\log^3 n$.
3 **for** i *in* $1 \dots T$ **do**
4 Sample δ pairs of vertices and query if there is an edge between them (in the adjacency list model, sample δ edges). Let E_S be the set of sampled edges.
5 Run the Algorithm 16 on E_S for one epoch to update H.
6 **if** *subgraph H did not change in this iteration* **then break**;
7 Let $E_U = \{(u,v) | (u,v) \in E, \deg_H(u) + \deg_H(v) < (1-\lambda)\beta\}$.
8 Sample r vertices $v_1, \dots v_r$ from V with replacement.
9 Let X_i be the indicator if vertex v_i is in the solution of maximum matching of graph $G' = (V, E_H \cup E_U)$ up to a multiplicative-additive factor of $(1-\varepsilon)$ using the algorithm of Lemma 3.3.14.
10 Let $X \leftarrow \sum_{i=1}^{r} X_i$ and $\widetilde{\mu} \leftarrow \frac{nX}{2r} - \frac{n}{2\log n}$.
11 **return** $\widetilde{\mu}$.

Algorithm 16: Algorithm of [47] on E_S for One Epoch to Update H.

1 **for** (u,v) *in* E_S **do**
2 **if** $\deg_H(u) + \deg_H(v) < (1-\lambda)\beta$ **then**
3 $H \leftarrow H \cup (u,v)$
4 **for** (u,w) *in* $N_H(u)$ **do**
5 **if** $\deg_H(u) + \deg_H(w) > \beta$ **then**
6 Remove edge (u,w) from H.
7 **break.**
8 **for** (v,w) *in* $N_H(v)$ **do**
9 **if** $\deg_H(u) + \deg_H(w) > \beta$ **then**
10 Remove edge (u,w) from H.
11 **break.**

We use the same argument as [47] to show that the average degree of violating unsampled edges is low. The only difference here is that in each epoch, we have $\widetilde{O}(n^{1-\varepsilon^3})$ sampled edges but the algorithm of [47] use $\widetilde{O}(n)$ sampled edges which causes us to get weaker bound on the average degree. First, we rewrite a useful lemma from [47] that also holds in our case which shows that in one of T epochs of Algorithm 15 the subgraph H does not change and the algorithm breaks the loop in Line 3.

Lemma 3.3.11 (Lemma 4.2 of [47])**.** *Let $\beta > 2$ and $H = (V_H, E_H)$ be a subgraph with no edges at the beginning. An adversary inserts and deletes edges from H with the following rules:*

- *delete an edge (u,v) if $\deg_H(u) + \deg_H(v) > \beta$,*
- *insert an edge (u,v) if $\deg_H(u) + \deg_H(v) < (1-\lambda)\beta$.*

Then after at most $n\beta^2$ insertions and deletions, no legal move remains.

Note that in the original version of this lemma in [47], the constraint for insertion rule is $\deg_H(u) + \deg_H(v) < \beta - 1$, but essentially the same proof carries over without any change. Therefore, the same $n\beta^2$ bound still holds for this weaker version. By Lemma 3.3.11, in at least one of T rounds of Algorithm 15, the subgraph H remains unchanged and the algorithm exits the loop in Line 3. Next, we prove that after the algorithm exits in Line 3 because of no change in H, the number of remaining unsampled edges that violate the second property of EDCS (P2) is small. (The proof of this lemma is similar to Lemma 4.1 of [47].)

Next, after the algorithm exits the loop in line 3 because of no change in H, the number of remaining unsampled edges that vaiolate the second property of EDCS (P2) is small. Assume that the algorithm sample p fraction of edges of the original graph at random and constructs the bounded edge-degree subgraph H. Behnezhad and Khanna [38] (see Claim 4.16 of the paper) proved that the number of unsampled violating edges is at most $O(\mu(G) \cdot \beta^2 \cdot \log n/p)$. In our case, $p = O(\beta^2/n^{\varepsilon^3})$ by the choice of δ and T. We restate the lemma with this specific p.

Lemma 3.3.12. *Let $\widetilde{H} = (V, E_U)$ be the subgraph consisting of violating unsampled edges in Line 7 of Algorithm 15. Then we have $|E_U| = O(\mu(G) \cdot n^{\varepsilon^3} \cdot \log n)$ with probability at least $1 - 1/n^5$.*

Claim 3.3.13. *Let E_S be the set of sampled edges in Line 4 of Algorithm 15. Then running the Algorithm 16 for one epoch to update H can be done in $O_\varepsilon(n^{1-\varepsilon^3})$ time. Moreover, the whole process of constructing H takes $O_\varepsilon(n^{2-\varepsilon^3})$ time in both adjacency lists and matrix models.*

Proof. Since Algorithm 15 samples at most $\delta = n^{1-\varepsilon^3}$ pairs of vertices, we have $|E_S| = O(n^{1-\varepsilon^3})$. Algorithm 16 iterates over the edges one by one in each epoch and adds edge (u, v) to H if $\deg_H(u) + \deg_H(v) < (1 - \lambda)\beta$. Adding this edge can cause incident edges of u and v to have a degree higher than β in H. In order to remove those edges, we iterate over edges of u and v in H and remove the first edge with a degree higher than β. This can be done in $O(\beta)$ since each vertex has at most β neighbors. Thus, the running time of each epoch is $O(n^{1-\varepsilon^3}\beta) = O_\varepsilon(n^{1-\varepsilon^3})$. Since we have $O(n\beta^2)$ epochs, the total running time is $O(n^{2-\varepsilon^3}\beta^3) = O_\varepsilon(n^{2-\varepsilon^3})$ by our choice of β. □

Lemma 3.3.14. *Let $G' = (V, E_H \cup E_U)$ be as defined in Algorithm 15. Then there exists a $(1 - \varepsilon)$-approximation LCA algorithm for maximum matching of G' with $\widetilde{O}_\varepsilon(n^{2-\varepsilon^3})$ preprocessing time and $\widetilde{O}_\varepsilon(n^{1+\varepsilon})$ additional time per query.*

Proof. Combining Lemma 3.3.12 and our choice of β, the average degree of graph G' is $O((\mu(G) \cdot n^{\varepsilon^3} \cdot \log n)/n)$. Note that the algorithm of [144] works with maximum degree. In order to use this algorithm as a subroutine in Line 9 of our algorithm, we ignore vertices of high degree and find a $(1 - \varepsilon)$-approximate maximum matching in the remaining graph by losing an additive error which depends on $\mu(G)$. However, we do not have access to the degree of vertices in G' since we do not have access to the adjacency list of E_U.

For each vertex v, we sample $k = 100n^{1-\varepsilon^3}\log^2 n$ vertices to estimate the degree of v. Let X be the number of neighbors that are in G' and $\widetilde{\deg}_{G'}(v) = nX/k$ be our estimate for the degree of v in G'. Using Chernoff bound, we get

$$\Pr\left[|\widetilde{\deg}_{G'}(v) - \deg_{G'}(v)| > n^{\varepsilon^3}\right] \leq 2\exp\left(-\frac{10000\log^4 n}{1500\log^3 n}\right) < \frac{2}{n^6}.$$

A union bound over all n vertices yields that with a probability of $1 - 1/n^4$, we have an additive error of at most n^{ε^3} for the degree of all vertices.

Now we ignore the vertices with estimated degrees larger than $n^{\varepsilon^3}\log^2 n$. Combining the additive error bound for degree estimation and the fact that the average degree of the graph is at most $O((\mu(G)\cdot n^{\varepsilon^3}\cdot \log n)/n)$, the total number of ignored vertices is at most $o(\mu(G))$.

For the remaining vertices, the maximum degree is at most $\widetilde{O}(n^{\varepsilon^3})$. Moreover, for an edge in G we can check if the degree of the edge is smaller than $(1-\lambda)\beta$ or not (the edge belongs to G' or not). Thus, we can run the algorithm of Corollary 3.3.10 in $\widetilde{O}_\varepsilon(n^{1+\varepsilon})$ since the maximum degree is $\widetilde{O}(n^{\varepsilon^3})$ and either we have access to adjacency matrix or adjacency list of G. Also, note that since the additive error is $o(\mu(G))$, the approximation ratio cannot be worse than $(1-\varepsilon)\mu(G') - o(\mu(G)) \geq (1-2\varepsilon)\mu(G')$. Proof of claim follows from using $\varepsilon/2$ as the parameter. □

Claim 3.3.15. $(\frac{2}{3}-\varepsilon)\cdot\mu(G) \leq \mu(G') \leq \mu(G)$.

Proof. The fact that G' is a subgraph of G implies the second inequality. Let $G^{Unsampled}$ be the subgraph consisting of unsampled edges. By Lemma 2.2 of [47], we have $\mu(G^{Unsampled}) \geq (1-2\varepsilon)\mu(G)$ with high probability. To see this, we sample $\tilde{O}(m/n^{\varepsilon^3})$ fraction of edges, however, in [47], they sample εm edges. Hence, the graph $G^{Unsampled}$ has more unsampled edges and therefore a larger matching compared to unsampled edges of [47]. Moreover, by Proposition 3.3.7, we have $\mu(G') \geq (\frac{2}{3}-\varepsilon)\mu(G^{Unsampled})$. Therefore, we obtain

$$\mu(G') \geq (\frac{2}{3}-\varepsilon)\mu(G^{Unsampled}) \geq (\frac{2}{3}-\varepsilon)(1-2\varepsilon)\mu(G) \geq (\frac{2}{3}-3\varepsilon)\mu(G).$$

Proof of claim follows from using $\varepsilon/3$ as the parameter. □

Lemma 3.3.16. *let $\widetilde{\mu}$ be the output of Algorithm 15 on graph G. With high probability,*

$$(\frac{2}{3}-\varepsilon)\mu(G) - o(n) \leq \widetilde{\mu} \leq \mu(G).$$

Proof. Let $\hat{M}$ be the $(1-\varepsilon)$-approximate matching of Lemma 3.3.14 on graph G' and X_i be the indicator if vertex i is matched in $\hat{M}$ or not. By Claim 3.3.15, we have

$$(1-\varepsilon)(\frac{2}{3}-\varepsilon)\mu(G) \leq \mathbf{E}\,|\hat{M}| \leq \mu(G). \tag{3.18}$$

Because the number of matching edges is half of the matched vertices,

$$\mathbf{E}[X_i] = \Pr[X_i = 1] = \frac{2\,\mathbf{E}\,|\hat{M}|}{n}.$$

Thus,

$$\mathbf{E}[X] = \frac{2r\,\mathbf{E}\,|\hat{M}|}{n}. \tag{3.19}$$

Using Chernoff bound and the fact that X is the sum of r independent Bernoulli random variables, we have

$$\Pr[|X - \mathbf{E}[X]| \geq \sqrt{18\,\mathbf{E}[X]\log n}] \leq 2\exp\left(-\frac{18\,\mathbf{E}[X]\log n}{3\,\mathbf{E}[X]}\right) = \frac{2}{n^6}.$$

Hence, with probability $1 - 2/n^6$,

$$\begin{aligned}
\widetilde{\mu} = \frac{nX}{2r} - \frac{n}{2\log n} &\in \frac{n(\mathbf{E}[X] \pm \sqrt{18\,\mathbf{E}[X]\log n})}{2r} - \frac{n}{2\log n} \\
&= \mathbf{E}\,|\hat{M}| \pm \sqrt{\frac{9n\,\mathbf{E}\,|\hat{M}|\log n}{r}} - \frac{n}{2\log n} && \text{(By (3.19))} \\
&= \mathbf{E}\,|\hat{M}| \pm \sqrt{\frac{n\,\mathbf{E}\,|\hat{M}|}{4\log^2 n}} - \frac{n}{2\log n} && \text{(Since } r = 36\log^3 n\text{)} \\
&= \mathbf{E}\,|\hat{M}| \pm \frac{n}{2\log n} - \frac{n}{2\log n} && \text{(Since } \mathbf{E}\,|\hat{M}| \le n\text{).}
\end{aligned}$$

Plugging (3.18) in the above range implies

$$(1-\varepsilon)(\frac{2}{3} - \varepsilon)\mu(G) - \frac{n}{\log n} \le \widetilde{\mu} \le \mu(G).$$

Since $\frac{n}{\log n} = o(n)$ and $(1-\varepsilon)(\frac{2}{3} - \varepsilon) > (\frac{2}{3} - 3\varepsilon)$, we get

$$(\frac{2}{3} - 3\varepsilon)\mu(G) - o(n) \le \widetilde{\mu} \le \mu(G).$$

Proof of lemma follows from using $\varepsilon/3$ as the parameter. □

Now we are ready to complete the analysis of the 2/3-approximate maximum matching algorithm.

Theorem 3.3.3. *For any constant $\varepsilon > 0$, there exists an algorithm that estimates the size of the maximum matching in $\widetilde{O}_\varepsilon(n^{2-\varepsilon^3})$ time up to a multiplicative-additive factor of $(\frac{2}{3} - \varepsilon, o(n))$ with high probability in both adjacency matrix and adjacency list model.*

Proof. We run Algorithm 15. By Lemma 3.3.16, we obtain the claimed approximation ratio. The proof of running time follows from combining Claim 3.3.13, Lemma 3.3.14, and $r = \widetilde{O}(1)$. □

Multiplicative Approximation for Adjacency List: In the adjacency list model, we can assume that we are given the degree of each vertex. This assumption is without a loss of generality since we can use binary search for each vertex to find its exact degree. Thus, we know the total number of edges in the graph and if this number is not larger than $n^{1.99}$, then we can use linear time $1-\varepsilon$ approximation for maximum matching. Equipped with this observation, we assume that $\mu(G) \ge n^{0.99}/2$, otherwise, the number of edges cannot be more than $2n \cdot \mu(G)$. Moreover, the size $\mu(G')$ is at least $(2/3 - \varepsilon)\mu(G)$, which implies that we can assume $\mu(G') = \Omega(n^{0.99})$. With this assumption and using standard Chernoff bound, it is not hard to see that we can estimate the size of the maximum matching of G' with a multiplicative factor by sampling $r = \widetilde{O}(n^{0.01})$ vertices. By the running time of Claim 3.3.13, preprocessing time of Lemma 3.3.14, and query time of Lemma 3.3.14 combined with r queries, results in multiplicative $(2/3 - \varepsilon)$-approximation algorithm with $\widetilde{O}_\varepsilon(n^{2-\varepsilon^3})$ running time.

Theorem 3.3.4. *For any constant $\varepsilon > 0$, there exists an algorithm that estimates the size of the maximum matching in $\widetilde{O}_\varepsilon(n^{2-\varepsilon^3})$ time up to a multiplicative factor of $(\frac{2}{3} - \varepsilon)$ with high probability in the adjacency list model.*

3.3.3 Beating 2/3-Approximation in Bipartite Graphs

In this section, we design a new algorithm that gets slightly better than 2/3 approximation in sublinear time in both the adjacency list and matrix models for bipartite graphs. Our starting point is to use the tight case characterization of the instance that our algorithm in the previous part cannot obtain better than a 2/3 approximation. Then we use that characterization to design a new algorithm to go beyond 2/3. We prove the following theorem, which further formalizes the statement of Theorem 3.3.2 in the introduction.

Theorem 3.3.17. *For an absolute constant α, there exists an algorithm that estimates the size of the maximum matching in $\widetilde{O}(n^{2-\Omega(1)})$ time up to a multiplicative-additive factor of $(\frac{2}{3}+\alpha, o(n))$ with high probability in both adjacency list and matrix models.*

Theorem 3.3.18. *For an absolute constant α, there exists an algorithm that estimates the size of the maximum matching in $\widetilde{O}(n^{2-\Omega(1)})$ time up to a multiplicative factor of $(\frac{2}{3}+\alpha)$ with high probability in the adjacency list model.*

To break 2/3-approximation, we build on a characterization of tight instances of EDCS due to a recent work of Behnezhad [32]. In our context, the characterization asserts that if the output of our Algorithm 15 is not already a $(2/3+\alpha)$-approximate matching of G for some constant $\alpha \geq 10^{-24}$ (we did not try to optimize the constant), then there exists a 2/3-approximate matching in G' (defined in Algorithm 15) such that it is far from being maximal in the original graph G.

Notation Let G be the input bipartite graph, and let $G' = (V, E_H \cup E_U)$ and $H = (V, E_H)$ be as defined in Algorithm 15. Let V_{low} and V_{mid} be the set of vertices v such that $\deg_H(v) \in [0, 0.2\beta]$ and $\deg_H(v) \in [0.4\beta, 0.6\beta]$, respectively. Let $G'' \coloneqq G'[V_{low}, V_{mid}]$. Also, fix M to be a $(1-\varepsilon)$-approximate matching of $G'' \coloneqq G'[V_{low}, V_{mid}]$. We let $A \coloneqq V_{mid} \setminus V(M)$ and $B \coloneqq V_{mid} \cap V(M)$ for the rest of this section.

Lemma 3.3.19 ([32])**.** *Let $\varepsilon < 1/120$ and α be some constant. If $\mu(G') \leq (\frac{2}{3}+\alpha)\mu(G)$, then both of the following guarantees hold:*

- **(G1 - G'' contains a large matching M)** $\mu(G'') \geq (\frac{2}{3} - 120\sqrt{\alpha}) \cdot \mu(G)$,
- **(G2 - M can be augmented in $G[A]$)** *For any matching M in G'', we have $\mu(G[V_{mid} \setminus V(M)]) \geq (\frac{1}{3} - 800\alpha) \cdot \mu(G)$.*

An Informal Overview of the Algorithm: Let M be a $(1-\varepsilon)$-approximate matching of $G'' \coloneqq G'[V_{low}, V_{mid}]$. According to this characterization, if Algorithm 15 does not obtain better than a $(2/3+\alpha)$ approximation, then matching M preserves the size of matching of G' and also, it can be augmented using a matching with edges of $G[V_{mid} \setminus V(M)]$. By Lemma 3.3.19, if the matching that we estimate in Algorithm 15 does not obtain better than 2/3 approximation, then there exists a large matching in G'' that can be augmented using a matching between vertices of A.

Therefore, if we want to obtain a better than 2/3 approximation, all we need is to estimate the size of a constant fraction of a matching in $G[A]$. The first challenge is that we do not know which vertices of V_{mid} are in A and which are in B. With a similar proof as proof of Lemma 3.3.14, we can show that it is possible

to query if a vertex is matched in M in $\widetilde{O}_\varepsilon(n^{1+\varepsilon})$ time, since G'' is a subgraph of G' and its average degree is smaller than G'. Moreover, for an edge in G, we can check if the H-degree of the edge is smaller than $(1-\lambda)\beta$ (the edge belongs to G' or not), and one endpoint is in V_{mid} and the other one in V_{low} (the edge belongs to G'' or not) in $O(1)$. Thus, we can run the algorithm of Corollary 3.3.10. We restate the lemma for subgraph G''.

Lemma 3.3.20. *There exists a $(1-\varepsilon)$-approximation LCA algorithm for maximum matching of $G'' := G'[V_{low}, V_{mid}]$ with $\widetilde{O}_\varepsilon(n^{2-\varepsilon^3})$ preprocessing time and $\widetilde{O}_\varepsilon(n^{1+\varepsilon})$ additional time per query.*

Since $G'[V_{low}, V_{mid}]$ is a subgraph of G', by Lemma 3.3.14, we can query if a vertex is matched in M in $\widetilde{O}_\varepsilon(n^{1+\varepsilon})$. So the time to classify if a vertex is in A or B is $\widetilde{O}_\varepsilon(n^{1+\varepsilon})$ since we can query if it is matched in matching M or not.

Unfortunately, the subgraph $G[A \cup B]$ can be a dense graph (possibly with an average degree of $\Theta(n)$), so it is not possible to run an adaptive sublinear algorithm that adaptively queries if a vertex is in A or not while looking for a matching. So we first try to sparsify the graph by sampling edges and constructing several maximal matchings. More specifically, we query δk pairs of vertices (for $\delta = n^{1+\gamma}$ and $k = n^{\gamma/2}$) and partition the queries to k equal buckets. Then we build a greedy maximal matching M^i_{AB} among those sampled edges of bucket i. By this construction and the sparsification property of greedy maximal matching, we are able to show that the maximum degree of subgraph $G[V_{mid} \setminus V(M^i_{AB})]$ is $O(n^{1-\gamma})$ with high probability for each i.

We split the rest of the analysis into three possible cases.

(Case 1) A constant fraction of vertices of A are matched in at least one of M^i_{AB}: in this case we immediately beat 2/3 since we found a constant fraction of a maximum matching edges of $G[A]$.

(Case 2) A constant fraction of maximum matching of $G[A]$ has both endpoints unmatched in at least one of M^i_{AB}: Let i^* be an index of matching $M^{i^*}_{AB}$ that a constant fraction of maximum matching of $G[A]$ has both endpoints unmatched. In this case, we prove that it is possible to estimate a constant approximation of the size of the matching between unmatched vertices of A in sublinear time.

Let $G^{i^*}_{AB}$ be the same as subgraph $G[V_{mid} \setminus V(M^{i^*}_{AB})]$, except that we connect each vertex in $B \setminus V(M^{i^*}_{AB})$ to $\kappa = \widetilde{O}(n^{1-\gamma})$ dummy singleton vertices. The reason to connect singleton vertices to B is that if we run a random greedy maximal matching on $G^{i^*}_{AB}$, most of the vertices in B will match to singleton vertices with some additive error. Thus, if there exists a large matching between unmatched A vertices, random greedy maximal matching will find at least half its edges. We sample $\widetilde{O}(1)$ vertices and test if they are matched in the random greedy maximal matching of $G^{i^*}_{AB}$. To do this, we use the algorithm of [34] which has a running time of $\widetilde{O}(n^{1-\gamma})$ for a random vertex. However, for each vertex that this algorithm recursively makes a query, we have to classify if it is in A or B since we do not explicitly construct $G^{i^*}_{AB}$. As we discussed at the beginning of this overview, this task can be done in $\widetilde{O}_\varepsilon(n^{1+\varepsilon})$ time. For each maximal matching, the running time for this case is $\widetilde{O}_\varepsilon(n^{2+\varepsilon-\gamma})$. Therefore, the total running time is $\widetilde{O}_\varepsilon(n^{2+\varepsilon-\gamma/2})$ for all $n^{\gamma/2}$ maximal matchings which is sublinear if we choose γ sufficiently larger than ε.

Note that if we are not in one of the above cases, almost all edges of $n^{\gamma/2}$ maximal matchings have at most one endpoint in A (as a result of not being in Case 1), and almost all edges of the maximum matching of $G[A]$ have at least one endpoint matched by all $n^{\gamma/2}$ maximal matchings (as a result of not being in Case

2). Thus, if we consider a vertex in A, almost all of its incident maximal matching edges go to set B. For this case, we now describe how it is possible to estimate the matching of $G[A]$.

(Case 3) Almost every edge of maximum matching of $G[A]$ has at least one of its endpoints matched by almost all M^i_{AB}: In this case we will show that we can efficiently remove many B-vertices, and then repeat the algorithm and and analysis of Cases 1 and 2 (using fresh samples of M^i_{AB}. Because we can't keep removing B-vertices indefinitely, eventually we have to end up at either Case 1 or 2.

Our goal is henceforth to efficiently identify many B-vertices. Suppose that we sample a vertex and check if it is in A or B. If the sampled vertex is in A and is one of the endpoints of maximum matching of A, most of its incident edges in $M^1_{AB}, M^2_{AB}, \ldots, M^k_{AB}$ are connected to vertices of B. Therefore, we are able to find $k = \Theta(n^{\gamma/2})$ vertices of the set B by spending $\widetilde{O}_\varepsilon(n^{1+\varepsilon})$ to find a vertex in A. Hence, if we sample $\widetilde{\Theta}(n^{1-\gamma/2})$ such vertices, we are expecting to see a constant fraction of vertices of B; we can remove these vertices and run the same algorithm on the remaining graph again. One challenge that arises here is that some of the neighbors of the sampled vertex might be in A. To resolve this issue, we choose a threshold η and only remove vertices that have at least η maximal matching edges that are connected to sampled A vertices. This will help us to avoid removing many A vertices in each round. Since initially the size of B and A is roughly the same (up to a constant factor), after a few rounds, we can estimate the size of the maximum matching of A. The total running time of this case is also $\widetilde{O}_\varepsilon(n^{2+\varepsilon-\gamma/2})$ which is the same as the previous case.

In what follows, we present the formal algorithm (Algorithm 17) and formalize the technical overview given in previous paragraphs.

Analysis of Running Time

We analyze each part of Algorithm 17 separately in this section and at the end, we put everything together. Before starting the analysis of the running time of Algorithm 17, we first restate the following result on the time complexity of estimating the size of the maximal matching by Behnezhad [34].

Lemma 3.3.21 (Lemma 4.1 of [34])**.** *There exists an algorithm that draws a random permutation over the edges of the graph and for an arbitrary vertex v in graph G, it determines if v is matched in the random greedy maximal matching of G in $\widetilde{O}(\bar{d})$ time with high probability, using adjacency list of G, where $\bar{d}$ is the average degree of G.*

Claim 3.3.22. *Let M^i_{AB} be as defined in Algorithm 17. The running time for constructing M^i_{AB} for all $1 \le i \le k$, takes $O(n^{1+3\gamma/2})$ time for all T iterations of the algorithm.*

Proof. Fix an iteration in the algorithm. Since we sample $\delta k = n^{1+3\gamma/2}$ pairs of vertices and we have k buckets of equal size, each bucket can have at most δ edges. For each of the buckets, in order to construct the maximal matching we need to iterate over the edges of the bucket one by one which takes $O(\delta k)$ time for all buckets. The proof follows from $T = O(1)$. □

Claim 3.3.23. *The total preprocessing time before all iterations of Algorithm 17 is $\widetilde{O}_\varepsilon(n^{2-\varepsilon^3})$.*

Proof. Algorithm 17 uses subroutine of Lemma 3.3.20 to query if a vertex is matched in $(1-\varepsilon)$-approximate matching of G'' during its execution. Thus, by Lemma 3.3.20, the total preprocessing time needed for the algorithm is $\widetilde{O}_\varepsilon(n^{2-\varepsilon^3})$. □

Algorithm 17: Better Than 2/3-Approximate Matching Algorithm in Adjacency List and Matrix

Parameter: ε, γ.
$r_1 \leftarrow 72\log^3 n, r_2 \leftarrow 288\log^3 n, r_3 \leftarrow 10n^{1-\gamma/2}\log n$.
$T \leftarrow 200, \alpha \leftarrow 10^{-10}, \kappa \leftarrow 48n^{1-\gamma}\log^2 n, \delta \leftarrow n^{1+\gamma}, k \leftarrow n^{\gamma/2}$, $c \leftarrow 10^{16}$.
Let $\widetilde{\mu}$ be the output of Algorithm 15 on G with parameter ε. Also, let β, H, E_U, and $G' = (V, E_H \cup E_U)$ be as defined in Algorithm 15.
$\eta \leftarrow \frac{n\log n}{100\widetilde{\mu}_1}$.
Let $V_{low} = \{v | \deg_H(v) \in [0, 0.2\beta]\}$, $V_{mid} = \{v | \deg_H(v) \in [0.4\beta, 0.6\beta]\}$, and $G'' \coloneqq G'[V_{low}, V_{mid}]$.
Let $\widetilde{\mu}_1$ be an estimate with an additive error of $o(n)$ for the size of $(1-\varepsilon)$-approximate matching of G'' by sampling $\widetilde{O}(1)$ vertices and running the algorithm of Lemma 3.3.20.
for j *in* $1 \dots T$ **do**
    Sample $k\delta$ pairs of vertices of V_{mid} and partition them into k buckets of equal size. Let M^i_{AB} be the greedy maximal matching on the sampled edges of bucket i.
    for i *in* $1 \dots k$ **do**
        // **Beginning of case 1**
        Sample r_1 random edges $e_1, \dots, e_{r_1}$ of M^i_{AB}.
        Let X_ℓ be the indicator if neither endpoint of e_ℓ is matched in the $(1-\varepsilon)$-approximate matching of G'' by running algorithm of Lemma 3.3.20.
        Let $X \leftarrow \sum_{\ell=1}^{r_1} X_\ell$ and $\widetilde{\mu}_2 = \frac{nX}{r_1} - \frac{n}{2\log n}$.
        if $\widetilde{\mu}_2 \geq c\alpha\widetilde{\mu}_1$ **then return** $\widetilde{\mu}_1 + \widetilde{\mu}_2$;
        // **End of case 1**
        // **Beginning of case 2**
        Let G^i_{AB} be the same as subgraph $G[V_{mid} \setminus V(M^i_{AB})]$, except that we connect each vertex in $B \setminus V(M^i_{AB})$ to κ singleton vertices.
        Sample r_2 random vertices $v_1, \dots, v_{r_2}$ of vertex set $A \setminus V(M^i_{AB})$.
        Let Y_ℓ be the indicator if v_ℓ is matched in the random greedy maximal matching of G^i_{AB} (algorithm of [34]).
        Let $Y \leftarrow \sum_{\ell=1}^{r_2} Y_\ell$ and $\widetilde{\mu}_3 = \frac{nY}{2r_2} - \frac{n}{2\log n}$.
        if $\widetilde{\mu}_3 \geq c\alpha\widetilde{\mu}_1$ **then return** $\widetilde{\mu}_1 + \widetilde{\mu}_3$;
        // **End of case 2**
    // **Beginning of case 3**
    Sample r_3 vertices $u_1, u_2, \dots u_{r_3}$ of vertex set A.
    Define $G_{M_{AB}} = (V_{mid}, \bigcup_i^k M^i_{AB})$.
    Let $C \subseteq V_{mid}$ be the set of vertices that has at least η neighbors in $G_{M_{AB}}$ among r_3 sampled vertices of A. ▷ C vertices likely in B
    $V_{mid} \leftarrow V_{mid} \setminus C$. ▷ Remove those vertices
    // **End of case 3**
return $\widetilde{\mu}$

Claim 3.3.24. *Every time that Algorithm 17 calculates $\widetilde{\mu}_2$, it takes $\widetilde{O}_\varepsilon(n^{1+\varepsilon})$ time. Furthermore, the total running time for computing $\widetilde{\mu}_2$ in the whole process of algorithm is $\widetilde{O}_\varepsilon(n^{1+\varepsilon+\gamma/2})$ time.*

Proof. By Lemma 3.3.20, each query to see if a vertex is matched in $(1-\varepsilon)$-approximate matching of G'', takes $\widetilde{O}_\varepsilon(n^{1+\varepsilon})$ time. Since we sample $\widetilde{O}(1)$ vertices, we get the claimed time complexity. □

Lemma 3.3.25. *Subgraph $G[V_{mid} \setminus V(M^i_{AB})]$ has a maximum degree of $6n^{1-\gamma}\log n$ with high probability.*

Proof. Consider a vertex v that is not matched in M^i_{AB} and has a degree larger than $6n^{1-\gamma}\log n$ in $G[V_{mid} \setminus V(M^i_{AB})]$. The only way that v remains unmatched is that none of its incident edges in $G[V_{mid} \setminus V(M^i_{AB})]$ get sampled in $n^{1+\gamma}$ pairs of vertices we sampled. The probability that we sample one of these edges is at least $6n^{1-\gamma}\log n/n^2$, since there are at most n^2 possible pairs of vertices. Thus, the probability that none of these edges sampled in $n^{1+\gamma}$ samples when constructing M^i_{AB} is at most

$$\left(1-\frac{6n^{1-\gamma}\log n}{n^2}\right)^{n^{1+\gamma}} = \left(1-\frac{6\log n}{n^{1+\gamma}}\right)^{n^{1+\gamma}} \leq n^{-6}.$$

Taking a union bound over all vertices of the graph completes the proof. □

Claim 3.3.26. *Every time that Algorithm 17 calculates $\widetilde{\mu}_3$, it takes $\widetilde{O}_\varepsilon(n^{2+\varepsilon-\gamma})$ time. Furthermore, the total running time for computing $\widetilde{\mu}_3$ in the whole process of algorithm is $\widetilde{O}_\varepsilon(n^{2+\varepsilon-\gamma/2})$ time.*

Proof. By Lemma 3.3.25, the maximum degree of subgraph G^i_{AB} is $\widetilde{O}(n^{1-\gamma})$ with high probability. Thus, by Lemma 3.3.21, indicating if a vertex is matched in the random greedy maximal matching of G^i_{AB} takes $\widetilde{O}(n^{1-\gamma})$ time. Moreover, for each vertex in the process of running a random greedy maximal matching local query algorithm, we need to distinguish if it is a vertex in A or B since we build subgraph G^i_{AB} on the fly (we cannot afford to build the whole graph at the beginning which takes $\Theta(n^2)$ time). By Lemma 3.3.20, each of these queries takes $\widetilde{O}_\varepsilon(n^{1+\varepsilon})$ time. Therefore, it takes $\widetilde{O}_\varepsilon(n^{2+\varepsilon-\gamma})$ time to calculate $\widetilde{\mu}_3$ each time. Combining with the fact that $r_2 = \widetilde{O}(1)$, we have $O(n^{\gamma/2})$ maximal matchings, the size of set A is a constant fraction of V_{mid}, and $T = O(1)$ implies the claimed running time. □

Claim 3.3.27. *Updating vertex set V_{mid} at the end of case 3 in Algorithm 17 takes $\widetilde{O}_\varepsilon(n^{2+\varepsilon-\gamma})$ time. Furthermore, the total running time for computing V_{mid} in the whole process of algorithm is $\widetilde{O}_\varepsilon(n^{2+\varepsilon-\gamma})$ time.*

Proof. Each time that we sample a vertex, we need to test if it is a vertex in A. This takes $\widetilde{O}_\varepsilon(n^{1+\varepsilon})$ by Lemma 3.3.20. Since we need to sample $r_3 = \widetilde{O}(n^{1-\gamma/2})$ vertices of A and the size of set A is a constant fraction of V_{mid} by Lemma 3.3.19, we need to sample at most $\widetilde{O}(n^{1-\gamma})$ vertices of V_{mid}. Hence, it takes $\widetilde{O}_\varepsilon(n^{2+\varepsilon-\gamma/2})$ time to find vertices $u_1, u_2, \ldots, u_k \in A$. Each of the vertices has at most k neighbors in all maximal matchings $M^1_{AB}, M^2_{AB} \ldots, M^k_{AB}$ which implies that we need to spend $\widetilde{O}(n)$ time to find and remove vertices of set C which completes the proof since $T = O(1)$. □

Now we are ready to complete the analysis of the running time of Algorithm 17.

Lemma 3.3.28. *The total running time of Algorithm 17 is $\widetilde{O}_\varepsilon(\max(n^{2+\varepsilon-\gamma/2}, n^{2-\varepsilon^3}, n^{1+3\gamma/2}, n^{1+\varepsilon+\gamma/2}))$.*

Proof. The proof follows from combining Lemma 3.3.20, Claim 3.3.22, Claim 3.3.23, Claim 3.3.24, Claim 3.3.26, and Claim 3.3.27. □

Analysis of Approximation Ratio

In this section, we assume that $\widetilde{\mu} \leq (\frac{2}{3}+\alpha)\cdot\mu(G)$ and we prove with this assumption, the algorithm outputs a better than 2/3 approximation in one of T iterations. The proof for the other case where $\widetilde{\mu} > (\frac{2}{3}+\alpha)\cdot\mu(G)$ is trivial since the algorithm outputs an estimation which is at least $\widetilde{\mu}$.

Approximation analysis of Case 1: Many A-vertices matched in some M^i_{AB}

Lemma 3.3.29. *If $\widetilde{\mu}_2 \geq c\alpha\widetilde{\mu}_1$, then $\widetilde{\mu}_1+\widetilde{\mu}_2$ is an estimate for $\mu(G)$ up to a multiplcative-additive factor of $(\frac{2}{3}+\alpha, o(n))$ with high probability.*

Proof. First, we prove that $\widetilde{\mu}_1+\widetilde{\mu}_2 \leq \mu(G)$. Let X be as defined in Algorithm 17. Note that $\frac{n\,\mathbf{E}[X]}{r_1} \leq \mu(G[A])$ since it counts the number of edges in M^i_{AB} that have both of their endpoints in A. Since X is the sum of r_1 independent Bernoulli random variables, using Chernoff bound we have

$$\Pr[|X-\mathbf{E}[X]| \geq \sqrt{18\,\mathbf{E}[X]\log n}] \leq 2\exp\left(-\frac{18\,\mathbf{E}[X]\log n}{3\,\mathbf{E}[X]}\right) = \frac{2}{n^6}.$$

Thus, with a probability of at least $1-2/n^6$,

$$\begin{aligned}
\widetilde{\mu}_2 \leq \frac{nX}{r_1} - \frac{n}{2\log n} &\leq \frac{n\left(\mathbf{E}[X]+\sqrt{18\,\mathbf{E}[X]\log n}\right)}{r_1} - \frac{n}{2\log n} \\
&\leq \mu(G[A]) + \sqrt{\frac{18n\cdot\mu(G[A])\log n}{r_1}} - \frac{n}{2\log n} && (\text{Since } \frac{n\,\mathbf{E}[X]}{r_1} \leq \mu(G[A])) \\
&\leq \mu(G[A]) + \sqrt{\frac{n\cdot\mu(G[A])}{4\log^2 n}} - \frac{n}{2\log n} && (\text{Since } r_1 = 72\log^3 n) \\
&\leq \mu(G[A]) && (\text{Since } \mu(G[A]) \leq n).
\end{aligned}$$

Moreover, we have that $\widetilde{\mu}_1 \leq \mu(G'')$. Since $G[A]$ only contains vertices that are not matched in the matching that we found in G'', we get $\widetilde{\mu}_1+\widetilde{\mu}_2 \leq \mu(G)$. On the other hand, by Lemma 3.3.19, we have $\widetilde{\mu}_1 \geq (1-\varepsilon)\cdot(\frac{2}{3}-120\sqrt{\alpha})\mu(G) - o(n)$. Thus,

$$\begin{aligned}
\widetilde{\mu}_1+\widetilde{\mu}_2 \geq (1+c\alpha)\widetilde{\mu}_1 &\geq (1+c\alpha)\left((1-\varepsilon)\left(\frac{2}{3}-120\sqrt{\alpha}\right)\cdot\mu(G) - o(n)\right) \\
&\geq (1+c\alpha)\cdot\left(\frac{2}{3}-120\sqrt{\alpha}-\frac{2}{3}\varepsilon\right)\cdot\mu(G) - o(n) \\
&\geq \left(\frac{2}{3}-120\sqrt{\alpha}-\frac{2}{3}\varepsilon+\frac{2}{3}c\alpha-120c\alpha\sqrt{\alpha}-\frac{2}{3}c\alpha\varepsilon\right)\cdot\mu(G) - o(n) \\
&\geq \left(\frac{2}{3}+\alpha\right)\mu(G) - o(n),
\end{aligned}$$

where the last inequality holds by our choice of c, α, and if we choose ε sufficiently small enough, which leads to our claimed approximation ratio. □

Approximation analysis of Case 2: Large matching in $G[A \setminus M^i_{AB}]$ (for some M^i_{AB})

Lemma 3.3.30. *If $\widetilde{\mu}_3 \geq c\alpha\widetilde{\mu}_1$, then $\widetilde{\mu}_1 + \widetilde{\mu}_3$ is an estimate for $\mu(G)$ up to a multiplcative-additive factor of $(\frac{2}{3} + \alpha, o(n))$ with high probability.*

Proof. Note that since we connect each vertex of $B \setminus V(M^i_{AB})$ to $\kappa = \widetilde{O}(n^{1-\gamma})$ singleton vertices, most of the vertices of $B \setminus V(M^i_{AB})$ will be matched to singleton vertices when running random greedy maximal matching. Let $v \in B \setminus V(M^i_{AB})$. The first incident edge of v in the permutation of edges is not an edge to singleton vertices with a probability of at most $(6n^{1-\gamma}\log n)/(\kappa + 6n^{1-\gamma}\log n)$ since the maximum degree of subgraph $G[V_{mid} \setminus V(M^i_{AB})]$ is at most $6n^{1-\gamma}\log n$ by Lemma 3.3.25. We denote the vertices of $B \setminus V(M^i_{AB})$ that are not matched with singleton vertices by R. Hence, we have the following bounds,

$$\mathbf{E}\,|R| \leq n \cdot \left(\frac{6n^{1-\gamma}\log n}{\kappa + 6n^{1-\gamma}\log n}\right) \leq n \cdot \left(\frac{6n^{1-\gamma}\log n}{\kappa}\right) = \frac{n}{8\log n},$$
$$\mathbf{E}\,|R| \geq n \cdot \left(\frac{6n^{1-\gamma}\log n}{\kappa + 6n^{1-\gamma}\log n}\right) \geq n \cdot \left(\frac{6n^{1-\gamma}\log n}{2\kappa}\right) = \frac{n}{16\log n}.$$

Therefore, using a Chernoff bound, we can show that $|R| \leq \frac{n}{4\log n}$ with a high probability.

Now, we prove that $\widetilde{\mu}_1 + \widetilde{\mu}_3 \leq \mu(G)$. Let Y be as defined in Algorithm 17, then we have $\frac{n\,\mathbf{E}[Y]}{2r_2} \leq \mu(G[A]) + |R|$, since it counts the number of edges in output of random greedy maximal matching with both endpoints in A and there are at most $|R|$ vertices of B can match to vertices of A. Also, each edge is counted at most twice since it has two endpoints in A. Thus,

$$\Pr[|Y - \mathbf{E}[Y]| \geq \sqrt{18\,\mathbf{E}[Y]\log n}] \leq 2\exp\left(-\frac{18\,\mathbf{E}[Y]\log n}{3\,\mathbf{E}[Y]}\right) = \frac{2}{n^6},$$

since Y is the sum of independent Bernoulli random variables. Therefore, with a probability of at least $1 - 2/n^6$,

$$\begin{aligned}
\widetilde{\mu}_3 &\leq \frac{nY}{2r_2} - \frac{n}{2\log n} \\
&\leq \frac{n\left(\mathbf{E}[Y] + \sqrt{18\,\mathbf{E}[Y]\log n}\right)}{2r_2} - \frac{n}{2\log n} \\
&\leq \mu(G[A]) + |R| + \sqrt{\frac{9n\cdot(\mu(G[A]) + |R|)\log n}{r_2}} - \frac{n}{2\log n} && \text{(Since } \frac{n\,\mathbf{E}[Y]}{2r_2} \leq \mu(G[A]) + |R|\text{)} \\
&\leq \mu(G[A]) + |R| + \sqrt{\frac{n\cdot(\mu(G[A]) + |R|)}{32\log^2 n}} - \frac{n}{2\log n} && \text{(Since } r_2 = 288\log^3 n\text{)} \\
&\leq \mu(G[A]), && \text{(Since } \mu(G[A]) \leq n \text{ and } |R| \leq \frac{n}{4\log n}\text{)}
\end{aligned}$$

which implies that $\widetilde{\mu}_1 + \widetilde{\mu}_3 \leq \mu(G)$ with the same argument as proof of Lemma 3.3.29. Now we are ready to finish the proof since

$$\begin{aligned}
\widetilde{\mu}_1 + \widetilde{\mu}_3 \geq (1 + c\alpha)\widetilde{\mu}_1 &\geq (1 + c\alpha)\left((1 - \varepsilon)\left(\frac{2}{3} - 120\sqrt{\alpha}\right)\cdot\mu(G) - o(n)\right) \\
&\geq (1 + c\alpha)\cdot\left(\frac{2}{3} - 120\sqrt{\alpha} - \frac{2}{3}\varepsilon\right)\cdot\mu(G) - o(n)
\end{aligned}$$

$$\geq \left(\frac{2}{3} - 120\sqrt{\alpha} - \frac{2}{3}\varepsilon + \frac{2}{3}c\alpha - 120c\alpha\sqrt{\alpha} - \frac{2}{3}c\alpha\varepsilon\right) \cdot \mu(G) - o(n)$$
$$\geq \left(\frac{2}{3} + \alpha\right)\mu(G) - o(n),$$

□

Approximation analysis of Case 3: Almost every edge of maximum matching of $G[A]$ has at least one of its endpoints matched by almost all M^i_{AB}

Lemma 3.3.31. *Before the start of the i-th iteration of Algorithm 17 for $i \in [1, T]$: if the algorithm has not terminated yet, then it holds that $\mu(G[A]) \geq \left(\frac{1}{4} - (i-1)\cdot(10^{20}\alpha)\right)\cdot\mu(G)$.*

Proof. In the beginning of the section we assumed that $\widetilde{\mu} \leq (\frac{2}{3} + \alpha)\cdot\mu(G)$. Note that A contains vertices that are not matched by the matching of estimation $\widetilde{\mu}_1$. Therefore, before the first iteration of the algorithm, by Lemma 3.3.19, set A must contain a matching of size larger than $(\frac{1}{3} - 800\alpha)\cdot\mu(G) \geq \mu(G)/4$ where the inequality follows from our choice of α.

We use induction to prove the lemma. Suppose that the algorithm has not terminated before the i-th iteration for $i > 1$. Note that since the algorithm has not terminated in the previous iteration for any of the maximal matching M^i_{AB}, the total number of maximal matching edges between vertices of A is at most $c\alpha\widetilde{\mu}_1 n^{\gamma/2} \leq c\alpha\mu(G)n^{\gamma/2}$, which implies that the average degree of a vertex in $G_{M_{AB}}[A]$ is at most

$$\begin{aligned}
\frac{c\alpha\mu(G)n^{\gamma/2}}{|A|} &\leq \frac{c\alpha\mu(G)n^{\gamma/2}}{(\frac{1}{4} - (i-2)(10^5\alpha))\cdot\mu(G)} && \text{(By induction hypothesis)}\\
&\leq \frac{c\alpha\mu(G)n^{\gamma/2}}{1/6\cdot\mu(G)} && (i \leq T \text{ and } 10^{20}T\alpha \leq \frac{1}{12})\\
&= 6c\alpha n^{\gamma/2}.
\end{aligned}$$

Let Z be the number of edges of $G[A]$ that are in one of the maximal matchings. Since we sample $r_3 = 10n^{1-\gamma/2}\log n$ vertices of A, we have

$$\mathbf{E}[Z] \leq 60c\alpha n\log n.$$

Since each vertex of A have at most $n^{\gamma/2}$ neighbors in A (because we have $k = n^{\gamma/2}$ maximal matchings), then by using Hoeffding's inequality, we obtain

$$\Pr[|Z - \mathbf{E}[Z]| \geq 30c\alpha n\log n] \leq 2\exp\left(-\frac{2\cdot(10n^{1-\gamma/2}\log n)\cdot(12\alpha n^{\gamma/2})}{n^{\gamma}}\right) \leq \frac{1}{n^{10}}.$$

Therefore, with high probability, there are at most $90c\alpha n\log n$ edges of maximal matchings of sampled vertices with both endpoints in A (*).

Now we show that at most $10^{20}\alpha\mu(G)$ vertices of A are removed in Line 28 (of Algorithm 17) in the previous iteration. For the sake of contradiction, assume that more than $10^{20}\alpha\mu(G)$ are removed in the previous iteration. Then, we have at least

$$10^{20}\alpha\mu(G)\cdot\eta = 10^{20}\alpha\mu(G)\cdot\left(\frac{n\log n}{100\widetilde{\mu}_1}\right)$$

$$\geq 10^{20}\alpha\mu(G)\cdot\left(\frac{n\log n}{100\mu(G)}\right) \qquad \text{(Since } \widetilde{\mu}_1 \leq \mu(G)\text{)}$$
$$= 100c\alpha n\log n \qquad \text{(Since } c = 10^{16}\text{)}$$

edges of maximal matching between vertices of A since each removed vertex must have at least η neighbors in the sampled vertices, which is a contradiction to (*). By induction, before iteration $i-1$, we have $\mu(G[A]) \geq (\frac{1}{4} - (i-1)\cdot(10^{20}\alpha))\cdot\mu(G)$. Also, at most $10^{20}\alpha\mu(G)$ vertices of A can be removed in the previous step which implies the claimed bound for iteration i. □

Corollary 3.3.32. *Before the start of the i-th iteration of Algorithm 17 for $i \in [1,T]$: if the algorithm has not terminated yet, then it holds that $|A| \geq \frac{1}{5}\mu(G)$*

Proof. By Lemma 3.3.31, we have that

$$\mu(G[A]) \geq (\frac{1}{4} - (i-2)(10^{20}\alpha))\cdot\mu(G) \geq \frac{1}{5}\mu(G),$$

where the last inequality follows from our choice of α and T. We finish the proof by the fact that the number of vertices of a graph is larger than its matching size. □

Observation 3.3.33. *Suppose that Algorithm 17 does not terminate in iteration i. Then, it holds $|B| \geq \mu(G)/5$ before iteration i.*

Proof. Consider maximal matching M^1_{AB}. Since the algorithm has not terminated in iteration i, at least $\mu(G[A]) - 2c\alpha\widetilde{\mu}_1$ edges of $\mu(G[A])$ are blocked by maximal matching edges since the algorithm does not find more than $2c\alpha\widetilde{\mu}_1$ edges of matching $\mu(G[A])$ in case 1 and 2. This means that at least one endpoint of these edges is matched with a vertex in B. Thus, we have

$$|B| \geq \mu(G[A]) - 2c\alpha\widetilde{\mu}_1 \geq (\frac{1}{4} - (i-1)(10^{20}\alpha) - 8\alpha)\cdot\mu(G) \geq \frac{1}{5}\cdot\mu(G),$$

where the second inequality follows from Lemma 3.3.31 and $\widetilde{\mu}_1 \leq \mu(G)$, and the last inequality follows by the choice of T, α, and c. □

Lemma 3.3.34. *Suppose that the Algorithm 17 does not terminate in iteration i. Then, at the end of this iteration, the algorithm removes at least $0.01\cdot|B|$ vertices of B.*

Proof. Since the algorithm does not terminate in iteration i for any of $k = n^{\gamma/2}$ maximal matchings, at least $(\mu(G[A]) - 2c\alpha\mu(G))\cdot n^{\gamma/2}$ edges of $\mu(G[A])$ are blocked by each of maximal matchings. This means that at least one endpoint of these edges is matched with a vertex in B. Thus, there are at least

$$(\mu(G[A]) - 2c\alpha\cdot\mu(G))\cdot n^{\gamma/2} \geq (\frac{1}{4} - (i-1)(10^{20}\alpha) - 2c\alpha)\cdot\mu(G)n^{\gamma/2} \geq \frac{1}{5}\cdot\mu(G)n^{\gamma/2}$$

edges with one endpoint in A and one endpoint in B in all maximal matchings. Hence, the expected average degree of a vertex in B in the subgraph $G_{M_{AB}}$ is at least

$$\frac{\frac{1}{5}\cdot\mu(G)n^{\gamma/2}}{|B|} \leq \frac{1}{5}\cdot n^{\gamma/2},$$

where the inequality is because $|B| \leq \mu(G'') \leq \mu(G)$. Let Z be the number of edges of $G_{M_{AB}}$ with one endpoint in A that is among r_3 sampled vertices and one endpoint in B. Since we sample $10n^{1-\gamma/2}\log n$ vertices of A, we have

$$\mathbf{E}[Z] \geq 2n\log n.$$

With the same argument as proof of Lemma 3.3.31, each vertex of A can have at most $n^{\gamma/2}$ neighbors in B, then by using Hoeffding's inequality, we get

$$\Pr[|Z - \mathbf{E}[Z]| \geq n\log n] \leq 2\exp\left(-\frac{2\cdot(10n^{1-\gamma/2}\log n)\cdot(\frac{1}{10}n^{\gamma/2})}{n^{\gamma}}\right) \leq \frac{1}{n^{10}},$$

which implies that with high probability, there are at least $n\log n$ edges of maximal matchings with one endpoint in sampled A vertices and one in B.

Next, we prove for each vertex in B, the number of sampled A vertices in $G_{M_{AB}}$ is small with high probability. Note that each vertex in A is sampled with a probability of $\frac{10n^{1-\gamma/2}}{|A|}$. Moreover, each vertex in B can have at most $n^{\gamma/2}$ neighbors in $G_{M_{AB}}$. Let W_v be the expected number of sampled A vertices that are connected to v in $G_{M_{AB}}$ for $v \in B$. Therefore,

$$\mathbf{E}[W_v] \leq \frac{10n\log n}{|A|} \leq \frac{50n\log n}{\mu(G)}.$$

where the last inequality holds because of Corollary 3.3.32. Then, by Chernoff bound

$$\Pr\left[|W_v - \mathbf{E}[W_v]| \geq \frac{25n\log n}{\mu(G)}\right] \leq 2\exp\left(-\frac{625n\log n}{150\mu(G)}\right) \leq \frac{2}{n^8}.$$

Using a union bound on all vertices of B implies that with high probability, each of the vertices of B has at most $75n\log n/\mu(G)$ neighbors in $G_{M_{AB}}$ among sampled A vertices.

For the sake of contradiction, suppose that the algorithm removes less than $0.01|B|$ vertices of B at the end of the iteration. This implies that the maximum number of edges between vertices of B and sampled A vertices is

$$\begin{aligned}
0.01|B|\cdot\left(\frac{75n\log n}{\mu(G)}\right) + 0.99|B|\cdot\left(\frac{n\log n}{100\widetilde{\mu}_1}\right) &\leq 0.01|B|\cdot\left(\frac{75n\log n}{\widetilde{\mu}_1}\right) + 0.99|B|\cdot\left(\frac{n\log n}{100\widetilde{\mu}_1}\right)\\
&\leq \frac{0.76|B|\cdot n\log n}{\widetilde{\mu}_1}\\
&\leq 0.76n\log n,
\end{aligned}$$

where the first term is the total number of edges of removed vertices of B and the second term is the total number of edges of vertices of B that are not removed. Since we proved that with high probability, there are at least $n\log n$ edges, then the above upper bound on the number of edges is a contradiction that completes the proof. □

Putting it all together: Beating two-thirds

Lemma 3.3.35. *If $\widetilde{\mu} \leq (\frac{2}{3}+\alpha)\cdot\mu(G)$, then the Algorithm 17 terminates in one of T iterations.*

Proof. Suppose that the algorithm does not terminate in any of T rounds. By Lemma 3.3.34, in each iteration, the algorithm will remove $0.01|B|$ vertices of B. Thus, after $T = 200$ iterations, none of the vertices of B will remain in the graph. However, plugging $T = 200$ in Lemma 3.3.31, we have $\mu(G[A]) \geq \frac{1}{5}\mu(G)$. Therefore, either $\widetilde{\mu}_2 \geq c\alpha\widetilde{\mu}_1$ or $\widetilde{\mu}_3 \geq c\alpha\widetilde{\mu}_1$ in iteration 200, since there is no vertices of B to block the matching of $G[A]$. □

Lemma 3.3.36. *The output of Algorithm 17 is a $(\frac{2}{3} + \alpha, o(n))$ estimate for maximum matching of G.*

Proof. If $\widetilde{\mu}$ is not a $(\frac{2}{3}+\alpha, o(n))$ estimate for maximum matching of G, then by Lemma 3.3.35, the algorithm terminates in one of the rounds. Therefore, by Lemma 3.3.29 and Lemma 3.3.30, the output is a $(\frac{2}{3}+\alpha, o(n))$ estimate for maximum matching of G. □

Theorem 3.3.17. *For an absolute constant α, there exists an algorithm that estimates the size of the maximum matching in $\widetilde{O}(n^{2-\Omega(1)})$ time up to a multiplicative-additive factor of $(\frac{2}{3} + \alpha, o(n))$ with high probability in both adjacency list and matrix models.*

Proof. By Lemma 3.3.36, we obtain the claimed approximation ratio. Also, since we choose ε to be smaller than γ in Algorithm 17, the running time is $\widetilde{O}_\varepsilon(n^{2-\varepsilon^3})$ by Lemma 3.3.28. □

Multiplicative Approximation for Adjacency List: With the same argument as the multiplicative approximation for the adjacency list in the previous section, we can assume $\mu(G') = \Omega(n^{0.99})$. Hence, if $\widetilde{\mu} < (2/3 + \alpha)\mu(G)$, we have $\mu(G[A]) = \Omega(n^{0.99})$ by Lemma 3.3.19. Therefore, with the exact same argument as previous section, using a standard Chernoff bound, for getting multiplicative approximation, we can use $r_1 = \widetilde{O}(n^{0.01})$ and $r_2 = \widetilde{O}(n^{0.01})$ samples for estimation of $\widetilde{\mu}_2$ and $\widetilde{\mu}_3$. Note that the running time of Claim 3.3.24 and Claim 3.3.26 will increase to $\widetilde{O}_\varepsilon(n^{1+\varepsilon+\gamma/2+0.01})$ and $\widetilde{O}_\varepsilon(n^{2+\varepsilon-\gamma/2+0.01})$, respectively because of the larger number of samples. However, if we choose γ larger than $0.02 + 2\varepsilon + 2\varepsilon^3$, the running time of the algorithm will remain $\widetilde{O}(n^{2-\varepsilon^3})$ by Lemma 3.3.28.

Theorem 3.3.18. *For an absolute constant α, there exists an algorithm that estimates the size of the maximum matching in $\widetilde{O}(n^{2-\Omega(1)})$ time up to a multiplicative factor of $(\frac{2}{3} + \alpha)$ with high probability in the adjacency list model.*

Chapter 4

Lower Bounds for Sublinear Time Maximum Matching

In this chapter, we study the sublinear matching problem from a lower bound perspective. In Section 4.2, we establish the first superlinear (in n) lower bound for this problem, showing that at least $n^{1.2-o(1)}$ queries in the adjacency list model are required to obtain a $\left(\frac{2}{3}+\Omega(1)\right)$-approximation of the maximum matching size.

In Section 4.3, we turn to the bounded degree regime and prove that any LCA computing a matching that is within an additive error of εn from the maximum matching in an n vertex graph of maximum degree Δ must take at least $\Delta^{\Omega(1/\varepsilon)}$ time. As a corollary, for sublinear time algorithms, our techniques imply that any algorithm estimating the size of the maximum matching up to an additive error of εn must also take $\Delta^{\Omega(1/\varepsilon)}$ time.

This bounded degree lower bound forms the backbone of the main result of this chapter: in Section 4.4, we prove that for any fixed $\delta > 0$, there exists a corresponding $\varepsilon = \varepsilon(\delta) > 0$ such that estimating the size of the maximum matching within an additive error of εn requires $\Omega(n^{2-\delta})$ time in the adjacency list model. Finally, in Section 4.5, we focus on the adjacency matrix model and extend the result of Section 4.4 to this setting as well.

4.1 Introduction

4.2 First Super Linear Lower Bound

In this section, we give the first super-linear in n lower bound for the sublinear matching problem. More specifically, we show that any better than 2/3-approximation requires at least $n^{1.2-o(1)}$ time.

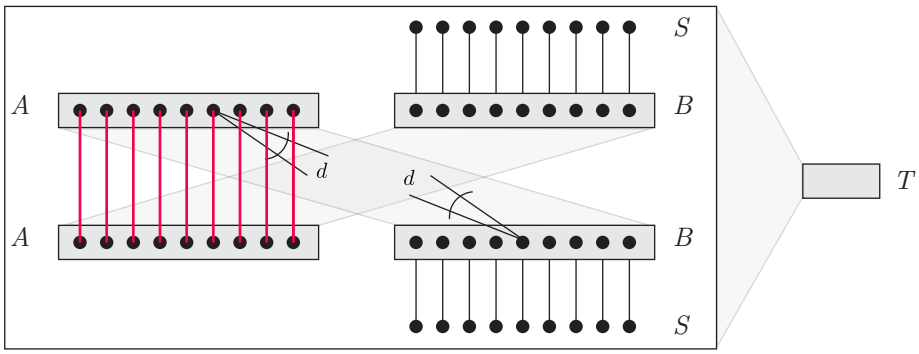
Theorem 4.2.1. *For any fixed $\alpha > 0$, any (possibly randomized) algorithm obtaining a $(2/3+\alpha)$-approximation of the size of maximum needs to make at least $n^{1.2-o(1)}$ adjacency list queries to the graph. This holds even if the graph is bipartite and has a matching of size $\Theta(n)$.*

Our proof of Theorem 4.2.1 relies crucially on *correlation decay*. To our knowledge, this is the first application of correlation decay in proving sublinear time lower bounds. We give an informal overview of our techniques in proving Theorem 4.2.1 in Section 4.2.1, and discuss why correlation decay is extremely helpful for us. The formal proof of Theorem 4.2.1 is then presented.

4.2.1 Technical Overview

An insightful, but broken, input distribution: Let us start with the lower bound. Consider the following input distribution, illustrated in the figure below. While we emphasize that this is not the final input distribution that we prove the lower bound with, it provides the right intuition and also highlights some of the challenges that arise in proving the lower bound.

There are four sets of vertices, A, B, S, T with $|A| = |B| = |S| = N$ and $T = \varepsilon N$. The T vertices are "dummy" vertices, they are adjacent to every other vertex in $A \cup B \cup S$. While this increases the degree of every vertex in $A \cup B \cup S$ to at least εN, the T vertices cannot contribute much to the output since $|T| = \varepsilon N$. The important edges, that determine the output are among the rest of the vertices. In particular, there is always a perfect matching between the B vertices and the S vertices. Additionally, there is a random Erdös-Renyi graph between A and B for a suitable expected degree $d = N^{\Theta(1)}$. Now in the YES distribution, we also add a perfect matching among the A vertices (the red matching) but in the NO distribution, we do not add this matching. Observe that in the YES case, we can match all of B to S and all of A together, obtaining a matching of size at least $3N$. However, in the NO case, it can be verified that the maximum matching is only at most $(2+\varepsilon)N$. Thus, any algorithm that beats 2/3-approximation by a margin more than ε, should be able to distinguish whether we are in the YES distribution or in the NO distribution.



Suppose now that we give away which vertices belong to S and T for free with no queries, but keep it secret whether a vertex $v \in A \cup B$ belongs to A or B. To examine whether a vertex v belongs to A or B, the naive approach is to go over all of its neighbors, and see whether we see an S neighbor or not. But because the degree of each vertex is $\Omega(\varepsilon N)$ due to the edges to T, this requires at least $\Omega(\varepsilon N) = \Omega(N)$ time. But this is not enough. To separate the two cases, one has to actually determine if there are any edges among the A vertices or not. Here, the naive approach is to first find a vertex $v \in A$ which can be done in $O(N)$ time, and then go over all neighbors u of v in $A \cup B$, and examine whether u belongs to A. Since v has $d = N^{\Omega(1)}$ neighbors in $A \cup B$, this naive approach takes $N^{1+\Omega(1)}$ time. We would like to argue that this naive approach is indeed the best possible, and that any algorithm distinguishing the two distributions must make $N^{1+\Omega(1)} = n^{1+\Omega(1)}$ queries to the graph. Unfortunately, this is not the case with the distribution above as we describe next.

Consider the following algorithm. We first start by finding an A vertex v in $O(n)$ time. Then we go over the neighbors of v in a random order until we find the first neighbor u that belongs to $A \cup B$. Note that this takes $O(n/d)$ time. We then do the same for u, finding a random neighbor to another vertex in $A \cup B$. Since each step of this random walk takes $O(n/d)$ time, we can continue it for $2d$ steps in $O(n)$ total time. Let w be the last vertex of the walk. We now examine whether w belongs to A or B in $O(n)$ time. We argue that by doing so, there is a constant probability that we can distinguish the YES case from the NO case. To see this, observe that in the NO distribution, every A vertex goes to a B vertex and every B vertex goes to an A vertex with probability one. As such, since the walk continues for an even number of steps, the last vertex w must belong to A with probability 1. But in the YES distribution, there is a constant probability that we go through an A-A edge exactly once. In this case, the last vertex w of the random walk must belong to B. Repeating this process a constant number of times allows us to distinguish the two distributions with probability 0.99 in merely $O(n)$ time!

Correlation Decay to the Rescue: To get around this challenge, we add εd edges also among the B vertices (while modifying the size of A and B slightly to ensure that the degrees in A and B do not reveal any information — see Figure 4.1). Observe that this completely destroys the algorithm above. In particular, it no longer holds that any B vertex goes to an A vertex with probability 1. Rather, there is a probability ε that we go from B to B. Hence, intuitively, whether the last vertex w of the random walk belongs to A or B does not immediately reveal any information about whether an A-A edge was seen or not. We show that indeed, this can be turned into a formal lower bound against all algorithms via correlation decay. First, we show that for suitable d, the queried subgraph will only be a tree. We then show that conditioning on the queries conducted far away from a vertex v, the probability of v being an A or B vertex will not change, and use this to prove that the algorithm cannot distinguish the YES and NO distributions.

4.2.2 The Lower Bound

In this section, we prove the lower bound of Theorem 4.2.1. To prove Theorem 4.2.1, we first give an input distribution. We then prove that any *deterministic* algorithm which with probability at least .51 (taken over the randomization of the input distribution) returns a $(2/3+\alpha)$-approximation of the size of maximum matching of the input must make at least $n^{1.2-o(1)}$ queries to the graph. By the 'easy' direction of Yao's minimax theorem [175], this also implies that any randomized algorithm with success probability .51 over

worst-case inputs must make at least $n^{1.2-o(1)}$ queries to obtain a $(2/3+\alpha)$-approximation.

4.2.3 The Input Distribution

We start by formalizing the input distribution. Let $\varepsilon := \alpha/100$, where α is as in Theorem 4.2.1, let N be a parameter which controls the number of vertices and let $d = N^{1/5}$. In our construction, we assume N, εN, d, and εd are all integers; note that this holds for infinitely many choices of N. For $n := 6N$, we construct an n-vertex bipartite graph $G = (V, U, E)$. We first categorize the vertices in G, then specify its edges.

The vertex set: The vertex set V is composed of four distinct subsets A_V, B_V, S_V, T_V and similarly U is composed of A_U, B_U, S_U, T_U. Throughout, we may write A, B, S, T to respectively refer to sets $A_V \cup A_U$, $B_V \cup B_U$, $S_V \cup S_U$, and $T_V \cup T_U$. In our construction, we will have

$$|A_V| = |A_U| = (1-\varepsilon)N, \qquad |B_V| = |B_U| = |S_V| = |S_U| = N, \qquad |T_U| = |T_V| = \varepsilon N.$$

We randomly permute all the vertices in A_V and use $v_i(A_V)$ to denote the i-th vertex of A_V in this permutation. We do the same for A_U, B_V, B_U, S_V, S_U.

The edge set: For the edge-set E, we give two distributions: in distribution $\mathcal{D}_{\mathsf{YES}}$ the maximum matching of G is sufficiently large, and in distribution $\mathcal{D}_{\mathsf{NO}}$ the maximum matching of G is sufficiently small. The final input distribution $\mathcal{D} := (\frac{1}{2}\mathcal{D}_{\mathsf{YES}} + \frac{1}{2}\mathcal{D}_{\mathsf{NO}})$ draws its input from $\mathcal{D}_{\mathsf{YES}}$ with probability $1/2$ and from $\mathcal{D}_{\mathsf{NO}}$ with probability $1/2$. The following edges are common in both disributions $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$:

- All vertices in T_U (resp. T_V) are adjacent to all of A_V, B_V, S_V (resp. A_U, B_U, S_U).
- For any $i \in [N]$, we add edges $(v_i(B_U), v_i(S_V))$ and $(v_i(B_V), v_i(S_U))$ to the graph. In words, there are perfect matchings between B_U and S_V and between B_V and S_U.
- Let $\mathcal{R}(B_V, B_U)$ be the set of all $(\varepsilon d - 1)$-regular graphs H between B_V and B_U such that for all $i \in [N]$, $(v_i(B_V), v_i(B_U)) \notin H$. We draw one regular graph $R(B_V, B_U)$ from $\mathcal{D}(B_V, B_U)$ uniformly at random and add its edges to G.
- Let $\mathcal{R}(A_V, B_U)$ be the set of all graphs H between A_V and B_U such that $\deg_H(v) = d$ for all $v \in A_V$, $\deg_H(u) = (1-\varepsilon)d$ for all $u \in B_U$, and for all $i \in [(1-\varepsilon)N]$, $(v_i(A_V), v_i(B_U)) \notin H$. We draw one regular graph $R(A_V, B_U)$ from $\mathcal{R}(A_V, B_U)$ uniformly at random and add its edges to G.
- Let $\mathcal{R}(B_V, A_U)$ be the set of all graphs H between B_V and A_U such that $\deg_H(v) = d$ for all $v \in A_U$, $\deg_H(u) = (1-\varepsilon)d$ for all $u \in B_V$, and for all $i \in [(1-\varepsilon)N]$, $(v_i(B_V), v_i(A_U)) \notin H$. We draw one regular graph $R(B_V, A_U)$ from $\mathcal{R}(B_V, A_U)$ uniformly at random and add its edges to G.
- For all $i \in \{(1-\varepsilon)N + 1, \ldots, N\}$ we add edge $(v_i(B_V), v_i(B_U))$ to G.

The following edges are specific to $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$ respectively:

- In $\mathcal{D}_{\mathsf{YES}}$, we additionally add edges $(v_i(A_V), v_i(A_U))$ and $(v_i(B_V), v_i(B_U))$ for all $i \in [(1-\varepsilon)N]$.
- In $\mathcal{D}_{\mathsf{NO}}$, we additionally add edges $(v_i(A_V), v_i(B_U))$ and $(v_i(B_V), v_i(A_U))$ for all $i \in [(1-\varepsilon)N]$.

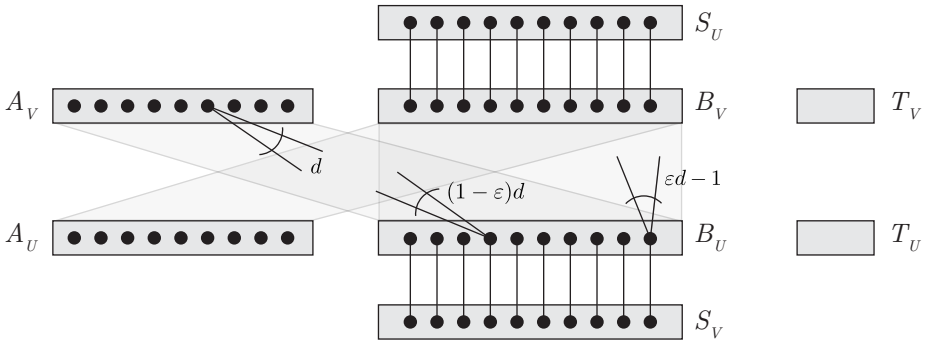
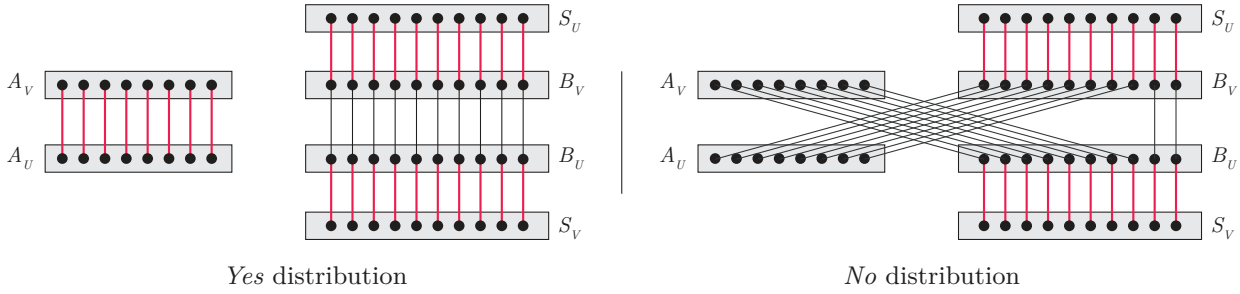


Figure 4.1: The common edges in both $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$ distributions. For simplicity, we have not illustrated the edges of T_V and T_U (which are adjacent to all vertices on the opposite side). See Figure 4.2 for edges specific to the two distributions.



Yes distribution *No* distribution

Figure 4.2: In addition to the common edges in distributions $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$ that were illustrated in Figure 4.1, we have a special perfect matching between vertices $A_V \cup B_V$ and $A_U \cup B_U$. In distribution $\mathcal{D}_{\mathsf{YES}}$, this perfect matching matches all of A_V to A_U. But in distribution $\mathcal{D}_{\mathsf{NO}}$, none of the edges of this matching go from A_V to A_U. This ensures that the maximum matching of G in the YES case is (almost) 1.5 times that of G in the NO case.

This concludes the construction of graph G. We also emphasize that the adjacency list of each vertex includes its neighbors in a random order chosen uniformly and independently.

Finally, we note that we give away the bipartition V, U of the graph for free. Additionally, we also give away which of the sets $S, T, A \cup B$ any vertex belongs to. What is crucially hidden from the algorithm, however, is whether a vertex $v \in A \cup B$ belongs to A or B.

4.2.4 Basic Properties of the Input Distribution

The following bounds on vertex degrees follows immediately from the construction.

Observation 4.2.2. *For any graph G drawn from $\mathcal{D}_{\mathsf{YES}}$ or $\mathcal{D}_{\mathsf{NO}}$ with probability 1 it holds that:*

1. $\deg(v) = \varepsilon N + d + 1$ *for all* $v \in A_V, A_U, B_V, B_U$.
2. $\deg(v) = \varepsilon N + 1$ *for all* $v \in S_V, S_U$.
3. $\deg(v) = (3 - \varepsilon)N$ *for all* $v \in T_V, T_U$.

Since for any vertex v, we know which of the sets $S_V, S_U, T_V, T_U, A_V \cup B_V, A_U \cup B_U$ it belongs to, Observation 4.2.2 implies we know the degree of every vertex in the graph before making any queries. Thus, there is no point for the algorithm to make any degree queries.

Lemma 4.2.3. *Let $G_{\mathsf{YES}} \sim \mathcal{D}_{\mathsf{YES}}$ and $G_{\mathsf{NO}} \sim \mathcal{D}_{\mathsf{NO}}$. Then it holds with probability 1 that*

$$\mu(G_{\mathsf{YES}}) \geq (3-\varepsilon)N \qquad \text{and} \qquad \mu(G_{\mathsf{NO}}) \leq (2+2\varepsilon)N.$$

This, in particular, implies that any algorithm satisfying the condition of Theorem 4.2.1 must be able to distinguish whether a graph G drawn from distribution $(\frac{1}{2}\mathcal{D}_{\mathsf{YES}} + \frac{1}{2}\mathcal{D}_{\mathsf{NO}})$ belongs to the support of $\mathcal{D}_{\mathsf{YES}}$ or $\mathcal{D}_{\mathsf{NO}}$ with probability at least .51.

Proof. For G_{YES}, we take edges $(v_i(A_V), v_i(A_U))$ for $1 \leq i \leq (1-\varepsilon)N$, $(v_i(B_V), v_i(S_V))$ for $1 \leq i \leq N$, and $(v_i(B_U), v_i(S_U))$ for $1 \leq i \leq N$. Since none of the edges have the same endpoint, the union creates a matching with size $(3-\varepsilon)N$ which implies $\mu(G_{\mathsf{YES}}) \geq (3-\varepsilon)N$.

For G_{NO}, we take $B_U \cup B_V \cup T_U \cup T_V$ as a vertex cover. Note that there is no edge in the induced subgraph between vertices of $A_U \cup A_V \cup S_U \cup S_V$. The proof follows by König's Theorem since there exists a vertex cover with $(2+2\varepsilon)N$ vertices.

Furthermore, if $\alpha > 0$ be a constant, since $(2+2\varepsilon)/(3-\varepsilon) \leq 2/3 + \alpha$ (if we choose ε sufficiently smaller than α), any algorithm satisfying the condition of Theorem 4.2.1, must be able to distinguish whether a graph G drawn from distribution $(\frac{1}{2}\mathcal{D}_{\mathsf{YES}} + \frac{1}{2}\mathcal{D}_{\mathsf{NO}})$ belongs to the support of $\mathcal{D}_{\mathsf{YES}}$ or $\mathcal{D}_{\mathsf{NO}}$ with probability at least .51. □

Remark 2. *Note that our construction can be slightly modified to have a perfect matching in $\mathcal{D}_{\mathsf{YES}}$. We can add a perfect matching between vertices of T_U and T_V to have a matching of size $3N$. Therefore, our lower bound also holds for the case that the $\mathcal{D}_{\mathsf{YES}}$ has a perfect matching.*

4.2.5 Queried Edges Form a Rooted Forest

Here and throughout the rest of the proof, we use G' to denote the induced subgraph of G excluding all the dummy vertices in $T_U \cup T_V$. Our main result of this section is that any algorithm that makes at most $o(n^{6/5})$ queries to graph G, only discovers a rooted forest in G':

Lemma 4.2.4. *Let $\mathcal{A}$ be any algorithm making at most $K = o(n^{6/5})$ queries to graph $G \sim \mathcal{D}$. Let F_0 be the empty graph on the vertex set of G', and for $t \geq 1$ let F_t be the subgraph of G' that $\mathcal{A}$ discovers after t queries. The following property holds throughout the execution of $\mathcal{A}$ with probability $1 - o(1)$: For any t, if the t'th query is to the adjacency list of a vertex v and returns edge (v,u) then either $(v,u) \notin G'$ or if $(v,u) \in G'$ then u is singleton in F_{t-1}. Equivalently, this implies that each F_t can be thought of as a rooted forest where $F_t \setminus F_{t-1}$ may only include one edge (v,u) and if $\mathcal{A}$ discovered (v,u) by querying v, then u becomes a leaf of v in F_t.*

Remark 3. *Our proof of Lemma 4.2.4 crucially relies on the fact that the adjacency list of each vertex is randomly permuted in the input distribution. However, we emphasize here that our proof holds even if the internal permutation of the G'-neighbors of a vertex v in the adjacency list of v is adversarial. This implies, in particular, that if we condition on the high probability event of Lemma 4.2.4, the internal permutation of the G'-neighbors will still be uniform.*

To prove Lemma 4.2.4, we first bound the number of edges of G' that we see with $K = o(n^{6/5})$ queries.

Claim 4.2.5. *Any algorithm $\mathcal{A}$ that makes $K = o(n^{6/5})$ queries to G, discovers at most $o(n^{2/5})$ edges of G' with probability $1 - 1/\operatorname{poly}(n)$.*

Proof. Since we assume $\mathcal{A}$ makes at most $K = o(n^{6/5})$ queries, there will be at most $o(n^{1/5})$ vertices for which the algorithm makes more than $\varepsilon N/2 = \Omega(n)$ adjacency list queries. For each of these $o(n^{1/5})$ vertices, we assume that we discover all their neighbors in G' that is at most $d+1 = O(n^{1/5})$, which in total is $o(n^{2/5})$ edges.

Now let V' be the set of vertices for which $\mathcal{A}$ makes at most $\varepsilon N/2$ queries. For each new query to a vertex $v \in V'$, since there are εN edges to $T_U \cup T_V$ in G and that we have already made at most $\varepsilon N/2$ queries to v, the new query goes to G' with probability at most $\deg_{G'}(v)/(\deg_{G'}(v) + \varepsilon N - \varepsilon N/2) = O(d/n)$. Next, let X_i be the indicator of the event that the i'th query to V' discovers a G' edge. From our earlier discussion, we get $\mathbf{E}[X_i] = O(d/n)$. Moreover, these X_i's are negatively correlated. Hence, denoting $X = \sum X_i$, we get $\mathbf{E}[X] \leq O(Kd/n)$ and can apply the Chernoff bound to obtain that with probability at least $1 - 1/\operatorname{poly}(n)$,

$$X \leq \mathbf{E}[X] + O(\sqrt{\mathbf{E}[X] \log n}) = O(Kd/n) = o(n^{2/5}),$$

concluding the proof. □

Next, we show that conditioning on the fact that we can discover at most $o(n^{2/5})$ edges, the probability of having an edge between any pairs of vertices is at most $O(1/n^{4/5})$. To give an intuition of why this claim holds, assume that instead of the regular graphs between B_U and B_V (similarly between A_U and B_V, and A_V and B_U), we have an Erdos-Renyi with the same expected degree as the regular graphs. Since the degree of each regular graph is $O(d)$, then the probability of having an edge between a pair of vertices must be $O(d/n) = O(1/n^{4/5})$ since the existences of edges are independent. However, since we draw a random $O(d)$-regular graph, edge realizations are not independent the same argument does not work. But using careful coupling, we prove that the same claim holds for this construction.

Claim 4.2.6. *Let us condition on the high probability event of Claim 4.2.5 that the algorithm has discovered at most $o(n^{2/5})$ edges in G'. Then for any pair of vertices u, v such that (u, v) is not among the discovered edges, the probability that (u, v) is an edge in G' is at most $O(1/n^{4/5})$.*

Proof. Let $A = A_U \cup A_V$ and $B = B_U \cup B_V$. Also, assume that by index of a vertex we mean the index in the permutation of its corresponding subset in the construction. There are three possible cases for the type of subsets that u and v belong to:

(Case 1) $u, v \in A$**:** note that since the algorithm makes at most $o(n^{6/5})$ queries, using Chernoff bound, it is easy to see that at least a constant fraction of perfect matching edges in the induced subgraph of A remains undiscovered. If u and v both belong to A, the probability of having an edge between the pair is at most $O(1/n)$ since we only have a perfect matching between vertices with the same indices in A_U and A_V for $\mathcal{D}_{\mathsf{YES}}$, and vertices are randomly permuted.

For the other two cases, we bound the probability of having an edge between u and v by considering two possible scenarios. First, the probability that both u and v have the same indices is equal to $1/n$. Next, assume that u and v have different indices. Let $\mathcal{G}_e$ be the set of all graphs in $\mathcal{D}$ such that all of them have

the edge $e = (u, v)$. Also, let $\overline{\mathcal{G}}_e$ be the set of all graphs in $\mathcal{D}$ with no edge between pair (u, v). In the next two cases, we use coupling to show that conditioned on discovered edges, we have $|\mathcal{G}_e|/|\overline{\mathcal{G}}_e| \leq O(1/n^{4/5})$ which implies that the probability of having an edge for a pair (u, v) is at most $O(1/n^{4/5})$.

(Case 2) $u, v \in B$**:** without loss of generality, assume that $u \in B_U$ and $v \in B_V$. First, note that with a probability of $O(1/n)$, the index of u and v is the same, which implies that there is an edge between u and v. Let i_x denote the index of vertex x. Now assume that $i_u \neq i_v$. We claim if there exists edge (u, v), then there are at least $N\varepsilon d/2$ edges (w, z) such that $z \in B_U$, $w \in B_V$, $\{i_u, i_v, i_w, i_z\} = 4$, and the induced subgraph of these four vertices only has two edges (u, v) and (w, z).

Since the number of incident edges of u to B_V is at most εd, then there are at least $N - \varepsilon d$ vertices in B_V that are not connected to u. Also, we exclude the vertex with index i_u in B_V from this set. Let Q be the set of such vertices in B_V. Hence, $|Q| \geq N - \varepsilon d - 1$. Furthermore, by the construction, each vertex $w \in Q$ has at least $\varepsilon d - 3$ neighbors in B_U with an index not in $\{i_u, i_v, i_w\}$. Hence, there are at least $(\varepsilon d - 3)(N - \varepsilon d - 1)$ edges between vertices of Q and B_U with endpoints having different labels. Moreover, at most $\varepsilon^2 d^2$ of these edges are incident to an edge that one of its endpoints is v. Therefore, there are at least $(\varepsilon d - 3)(N - \varepsilon d - 1) - \varepsilon^2 d^2 \geq N\varepsilon d/2$ edges (w, z) that satisfy the claimed properties where the inequality follows by the choice of d.

Now assume that we remove all edges that we discovered so far. Since we discover at most $o(n^{2/5})$ edges, by removing these edges we still have $O(n\varepsilon d)$ edges (w, z) with mentioned properties. Let (w, z) be such an edge where $w \in B_V$ and $z \in B_U$. We construct a graph by removing edge (u, v) and (z, w), and adding edges (u, w) and (v, z). If the original graph is in $\mathcal{D}_{\mathsf{YES}}$ ($\mathcal{D}_{\mathsf{NO}}$), the new graph is in $\mathcal{D}_{\mathsf{YES}}$ ($\mathcal{D}_{\mathsf{NO}}$) according to the construction since we only change the random regular bipartite graph between B_U and B_V without changing the degrees. We construct a bipartite graph where each vertex of the first part represents a graph in $\mathcal{G}_e$ and each vertex in the second part represents a graph in $\overline{\mathcal{G}}_e$. For each vertex in the first part, we connect it to the vertices of the second part that can be produced by the above operation. Thus, the degree of vertices in the first part is at least $O(n\varepsilon d)$. On the other hand, each vertex in the second part is connected to at most $O(\varepsilon^2 d^2)$ vertices of the first part since the degree of each u and v is at most εd in the induced subgraph of B. Counting the edges from both sides of the constructed bipartite graph yields

$$\frac{|\mathcal{G}_e|}{|\overline{\mathcal{G}}_e|} \leq \frac{O(\varepsilon^2 d^2)}{O(N\varepsilon d)} = O\left(\frac{\varepsilon d}{N}\right) = O_\varepsilon\left(\frac{1}{n^{4/5}}\right).$$

(Case 3) $u \in A, v \in B$ **or** $u \in B, v \in A$**:** without loss of generality, assume that $u \in A_U$ and $v \in B_V$. With the same argument as the previous case, the probability that both u and v have the same index is $O(1/n)$. Moreover, since the degree of the induced subgraph of $A_U \cup B_V$ differs from the previous case by a constant factor, with the same reasoning as the previous part, the probability that there exists an edge between u and v is at most $O(1/n^{4/5})$.

Therefore, conditioning on the discovered edges, the probability of having an edge (u, v), is at most $O(1/n^{4/5})$ □

Now we are ready to complete the proof of Lemma 4.2.4.

Proof of Lemma 4.2.4. First, we prove that for any $t \geq 1$, if two vertices u, v are non-singleton in F_t and $(u, v) \notin F_t$, then there is no edge between u and v in graph G'. This will imply that any time that we discover

a new edge of a non-singleton vertex in F_t, it must go to a vertex that is singleton in F_t. We prove this by induction on t. For F_0, the graph is empty and so the claim holds. Suppose that there are no undiscovered edges among the non-singleton vertices of F_{t-1}. We prove that this continues to hold for F_t. If $F_t \setminus F_{t-1} = \emptyset$, i.e., if we do not discover any new edge of G' at step t, then the claim clearly holds. So suppose that we query some vertex v and find an edge (v, u) at step t. It suffices to show that in this case, u will not have any non-singleton neighbors in F_{t-1} other than v. To see this, recall by Claim 4.2.5 that there are at most $o(n^{2/5})$ non-singleton vertices in F_{t-1}. Fixing any such vertex w with $w \neq v$, we get by Claim 4.2.6 that the conditional probability of (u, w) being an edge in G' is at most $O(1/n^{4/5})$. By a union bound over all $o(n^{2/5})$ choices of w, we get that the probability of u having an edge to any of the non-singleton vertices of F_{t-1} is $o(n^{2/5}) \cdot O(1/n^{4/5}) = o(1/n^{2/5})$. Finally, since by Claim 4.2.5 there are at most $O(n^{2/5})$ steps where we discover any edge of G', the failure probability over all the steps of the induction is $O(n^{2/5}) \cdot o(1/n^{2/5}) = o(1)$. Hence, the claim is true throughout with probability at least $1 - o(1)$.

We proved above that by querying an already non-singleton vertex v, its discovered neighbor u must be singleton w.h.p. We will now prove that this also holds if v itself is singleton. As discussed, the number of non-singleton vertices in F_t is $o(n^{2/5})$ by Claim 4.2.5. For each of these vertices u, the conditional probability of v having an edge to u is at most $O(1/n^{4/5})$ by Claim 4.2.6. Hence, the expected number of edges of v to non-singleton vertices is $o(1/n^{2/5})$. Since v has $\Omega(n)$ edges in G and its adjacency list is uniformly sorted, the probability of discovering one of such edges of v is at most $o(1/n^{2/5})/n = o(1/n^{7/5})$. A union bound over all $o(n^{6/5})$ queries of the algorithm implies that with probability $1 - o(1)$, any time that we query a singleton vertices of F_t, its discovered edge will not be to another non-singleton vertex of F_t. □

4.2.6 The Tree Model

In our proofs, we condition on the high probability event of Lemma 4.2.4 that F_t for any t forms a rooted forest. Under this event, we prove that the algorithm cannot distinguish the YES distribution from the NO distribution.

Recall that from our definition in Lemma 4.2.4, every vertex in F_t for any t is a vertex in set $A \cup B \cup S$. Given the forest F_t, we do not necessarily know for a given a vertex v in F_t whether it belongs to A or B (but if it belongs to S we know this since recall all the S vertices are given for free by the distribution). Inspired by this, we can see each vertex v in F_t as a random variable taking one of the values $\{A, B, S\}$. Importantly, depending on the value of v we have a different distribution on what values its children in the tree will take. This distribution is also different for the YES and NO distributions. For example, in the NO distribution all children of any vertex $v \in A$ will be B vertices whereas in the YES distribution there is a small chance of seeing an A child. We prove that although these distributions are different for the YES and NO distributions, no algorithm can distinguish (with a sufficiently large probability) whether the observed forest F_t was drawn from $\mathcal{D}_{\mathsf{YES}}$ or $\mathcal{D}_{\mathsf{NO}}$.

The proof consists of multiple steps. First, we prove a correlation decay property on the tree. Then equipped with the correlation decay property, we are able to show the probability of seeing the same forest by the algorithm in a $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$ is almost the same which implies that the algorithm cannot decide if the graph is drawn from $\mathcal{D}_{\mathsf{YES}}$ or $\mathcal{D}_{\mathsf{NO}}$.

4.2.7 Correlation Decay

Lemma 4.2.7 below is our main result in this section. Before stating the lemma, let us start with some definitions. Consider a tree $T(v)$ of any arbitrary depth rooted at vertex v. We say $T(v)$ is ℓ-S-free if no vertex in $T(v)$ with distance at most ℓ from the root belongs to S. We also use $Q(T(v), \ell)$ to denote the total number of vertices of distance at most ℓ from the root v in $T(v)$. With a slight abuse of notation, we may also use $T(v)$ to denote the event that the sub-tree of F_t rooted at vertex v by the end of the algorithm (i.e., when $t = K$) is exactly the same as $T(v)$.

Lemma 4.2.7. *Let v be any vertex, $\ell \geq 20(\log^2 n)/\varepsilon$, and let $T(v)$ be any ℓ-S-free tree. Let $P(v)$ and $P'(v)$ each be an arbitrary outcome of v from $\{A, B\}$ and the entire forest $F_t(v)$ excluding the sub-tree of v. Letting $Q := Q(T(v), \ell)$ and assuming that $Q = o(d)$, we have*

$$\Pr_{\mathcal{D}_{\mathsf{YES}}}[T(v) \mid P(v)] \leq \left(1 + \frac{1}{d}\right)^{O(Q)} \Pr_{\mathcal{D}_{\mathsf{YES}}}[T(v) \mid P'(v)],$$

and

$$\Pr_{\mathcal{D}_{\mathsf{NO}}}[T(v) \mid P(v)] \leq \left(1 + \frac{1}{d}\right)^{O(Q)} \Pr_{\mathcal{D}_{\mathsf{NO}}}[T(v) \mid P'(v)].$$

Proof. Call a vertex $u \in T(v)$ an *internal* vertex if it has distance at most ℓ from v. Note that Q is the number of internal vertices in $T(v)$. We say a subset of internal vertices $u_1, \ldots, u_k$ form an *internal path* in a tree T if for any $1 \leq i \leq k-1$, u_i is the parent of u_{i+1} in T, and additionally each u_i, for $i \in [k]$, only has one child in T.

We use the following two auxiliary Claims 4.2.8 and 4.2.9 to prove Lemma 4.2.7.

Claim 4.2.8. *Suppose that every vertex in $T(v)$ has at least one non-internal descendant. Then T must have an internal path of length at least $10 \log n/\varepsilon$.*

Proof. Suppose for contradiction that $T(v)$ does not have any internal path of length ℓ'. Construct a tree $T'(v)$ by contracting all maximal internal paths of $T(v)$ (of any length). Since every vertex in $T(v)$ has a non-internal descendant, which is of distance at least ℓ from the root by definition, and every contracted path has a length less than ℓ', every vertex in $T'(v)$ must have a descendant of distance at least ℓ/ℓ' from the root. Furthermore, every vertex in $T'(v)$ must have at least two children. As such, we get that $T'(v)$ must have $2^{\ell/\ell'} > n$ vertices, a contradiction. □

The proof of Claim 4.2.9 below is involved; we state it here and present the proof later.

Claim 4.2.9. *Let $(u_1, \ldots, u_{\ell'})$ be an internal path in $T(v)$ of length $\ell' \geq 10 \log n/\varepsilon$ such that each u_i is the parent of u_{i+1}. Let $T(u)$ be the sub-tree of $T(v)$ rooted at vertex u and let $P(u_1)$ and $P'(u_1)$ be any two events on the vertices outside u_1's sub-tree. Then*

$$\Pr_{\mathcal{D}_{\mathsf{YES}}}[T(u_{\ell'}) \mid P(u_1)] \leq \left(1 + \frac{1}{d}\right)^{O(\ell')} \Pr_{\mathcal{D}_{\mathsf{YES}}}[T(u_{\ell'}) \mid P'(u_1)],$$

and

$$\Pr_{\mathcal{D}_{\mathsf{NO}}}[T(u_{\ell'}) \mid P(u_1)] \leq \left(1 + \frac{1}{d}\right)^{O(\ell')} \Pr_{\mathcal{D}_{\mathsf{NO}}}[T(u_{\ell'}) \mid P'(u_1)].$$

We are now ready to present the proof of Lemma 4.2.7. Given the tree $T(v)$, to measure the probability of $T(v)$ actually happening we start by querying from v the same tree topology. That is, if some vertex u

has x children in $T(v)$, we reveal x children of u from the distribution and measure the probability that the resulting tree is exactly the same as $T(v)$. First, let us measure the probability that the resulting tree is indeed ℓ-S-free. That is, no internal vertex in the queried subtree is an S vertex. To do this, take an internal vertex u in $T(v)$. Conditioned on either of $P(u_1)$ or $P'(u_1)$, the probability of u being an S vertex is at most $O(1/d)$ since every vertex in G' has at most one S neighbor, has degree $d+1$, and its adjacency list is uniformly sorted. On the other hand, since the total number of internal vertices in $T(v)$ is Q, by a union bound, the probability of seeing at least one S vertex is at most $O(Q/d)$. Let us condition on the event that we see no internal S vertex. This only multiplies the final probability of $T(v)$ occuring by some $1 \pm O(Q/d) \leq (1 \pm 1/d)^{O(Q)}$ factor.

Conditioned on the observed tree being ℓ-S-free, note that if some internal vertex u in the tree has no non-internal descendants, then indeed its sub-tree exactly matches $T(v)$. So let us peel off all such sub-trees, arriving at a tree where every internal vertex has at least one non-internal descedntant. From Claim 4.2.8, we get that there must be a an internal path $u_1, \ldots, u_{\ell'}$ of length at least $\ell' \geq 10 \log n/\varepsilon$ in this tree. Applying Claim 4.2.9, the probability of $T(u_{\ell'})$ happening remains the same up to a factor of $(1+1/d)^{O(\ell')}$, no matter what we condition on above u_1. Thus, we can peel off the whole sub-tree of u_1 and repeat. Assuming that we repeat this process K times until we peel off all vertices, we get that the probability of $T(v)$ occuring under $P(v)$ and $P'(v)$ is the same up to a factor of $(1 \pm 1/d)^{O(Q)} \times (1 \pm 1/d)^{\ell' K} = (1 \pm 1/d)^{O(Q+\ell' K)}$. Finally, note that $\ell' K = O(Q)$ since every time we peel off the sub-tree of an internal path, we remove ℓ' internal vertices in the path, and so $K \times \ell'$ is at most the number of internal vertices. This completes the proof of Lemma 4.2.7. □

Correlation Decay on Internal Paths: Proof of Claim 4.2.9

In this section, we present the proof of Claim 4.2.9, which as discussed implies correctness of Lemma 4.2.7.

Proof of Claim 4.2.9. For any $i \in [\ell']$ we define

$$b_i = \Pr[u_i \in B \mid P(u)], \qquad a_i = \Pr[u_i \in A \mid P(u)].$$

We claim that for both distributions $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$, and for any $i > 1$,

$$b_i \in \left(1 \pm \frac{2}{\varepsilon d}\right)(\varepsilon b_{i-1} + a_{i-1}), \qquad a_i \in \left(1 \pm \frac{2}{\varepsilon d}\right)(1-\varepsilon) b_{i-1} \pm \frac{a_{i-1}}{d}. \tag{4.1}$$

To see this, let us first focus on b_i. We have

$$\begin{aligned} \Pr[u_i \in B \mid P(u)] &= \Pr[u_i \in B \mid u_{i-1} \in B, P(u)] \cdot \Pr[u_{i-1} \in B \mid P(u)] \\ &\quad + \Pr[u_i \in B \mid u_{i-1} \in A, P(u)] \cdot \Pr[u_{i-1} \in A \mid P(u)] \\ &= \Pr[u_i \in B \mid u_{i-1} \in B, P(u)] b_{i-1} + \Pr[u_i \in B \mid u_{i-1} \in A, P(u)] a_{i-1}. \end{aligned} \tag{4.2}$$

Let us now examine $\Pr[u_i \in B \mid u_{i-1} \in B, P(u)]$. Given that $u_{i-1} \in B$, vertex u_{i-1} has exactly $d+1$ neighbors in graph G'. Among them, either εd or $\varepsilon d - 1$ neighbors of u_i are in B. Additionally, since u_{i-1} has exactly one child (by definition of internal paths) then the conditional event $P(u)$ may only reveal the A/B-value of one neighbor of u_{i-1}, which would be its parent. Since the adjacency list of vertex u_{i-1} is

randomly sorted, its discovered child u_i belongs to B with probability at least $\frac{\varepsilon d-2}{d} \geq (1-\frac{2}{\varepsilon d})\varepsilon$ and at most $\frac{\varepsilon d}{d} = \varepsilon$. Thus,

$$\Pr[u_i \in B \mid u_{i-1} \in B, P(u)] \in (1 \pm \frac{2}{\varepsilon d})\varepsilon.$$

On the other hand, since every vertex in A has at least d and at most $d+1$ neighbors in A,

$$\Pr[u_i \in B \mid u_{i-1} \in A, P(u)] \in (1 \pm \frac{1}{d}).$$

Replacing these two bounds in (4.2), we indeed arrive at the recursion of (4.1) for b_i.

The calculation for a_i is similar. In particular, as in (4.2), we have

$$\Pr[u_i \in A \mid P(u)] = \Pr[u_i \in A \mid u_{i-1} \in B, P(u)]b_{i-1} + \Pr[u_i \in A \mid u_{i-1} \in A, P(u)]a_{i-1}.$$

We have $\Pr[u_i \in A \mid u_{i-1} \in B, P(u)] \in (1 \pm \frac{2}{\varepsilon d})(1-\varepsilon)$ since almost $(1-\varepsilon)$ fraction of neighbors of each B vertex go to A, and we have $\Pr[u_i \in A \mid u_{i-1} \in A, P(u)] \leq 1/d$ since each A vertex has at most one A neighbor among its $d+1$ neighbors. Replacing these into the inequality above, implies the recursion of (4.1) for a_i.

The following claim gives an explicit (i.e., non-recursive) bound for b_i and a_i as a function of just a_1 and b_1.

Claim 4.2.10. *Let*

$$a'_i := \frac{(1-\varepsilon)^2(a_1+b_1) - (\varepsilon-1)^i(a_1-(1-\varepsilon)b_1)}{(2-\varepsilon)(1-\varepsilon)},$$

and

$$b'_i := \frac{(1-\varepsilon)(a_1+b_1) + (\varepsilon-1)^i(a_1-(1-\varepsilon)b_1)}{(2-\varepsilon)(1-\varepsilon)}.$$

Then for any $i \geq 3$, we have

$$b_i \in \left(1 \pm \frac{2}{\varepsilon d}\right)^{2i} b'_i, \qquad a_i \in \left(1 \pm \frac{2}{\varepsilon d}\right)^{2i} a'_i.$$

Proof. First, we note a couple of useful properties of b'_i and a'_i that all can be verified from their definitions:

$$a'_1 = a_1, \qquad b'_1 = b_1, \qquad b'_i = \varepsilon b'_{i-1} + a'_{i-1}, \qquad a'_i = (1-\varepsilon)b'_{i-1}. \tag{4.3}$$

To prove the stated bounds on b_i and a_i we use induction on i. For the base case $i = 3$, directly applying (4.1) we get

$$\begin{aligned}
b_3 &\in \left(1 \pm \frac{2}{\varepsilon d}\right)(\varepsilon b_2 + a_2) \\
&\in \left(1 \pm \frac{2}{\varepsilon d}\right)\left[\varepsilon\left(\left(1 \pm \frac{2}{\varepsilon d}\right)(\varepsilon b_1 + a_1)\right) + \left(\left(1 \pm \frac{2}{\varepsilon d}\right)(1-\varepsilon)b_1 \pm \frac{a_1}{d}\right)\right] \\
&\in \left(1 \pm \frac{2}{\varepsilon d}\right)^2\left[\varepsilon^2 b_1 + \varepsilon a_1 + (1-\varepsilon)b_1 \pm \frac{a_1}{d}\right] \\
&\in \left(1 \pm \frac{2}{\varepsilon d}\right)^3\left[(1-\varepsilon+\varepsilon^2)b_1 + \varepsilon a_1\right]
\end{aligned}$$

$$= \left(1 \pm \frac{2}{\varepsilon d}\right)^3 b'_3 \in \left(1 \pm \frac{2}{\varepsilon d}\right)^6 b'_3.$$

Similarly, we have

$$\begin{aligned}
a_3 &\in \left(1 \pm \frac{2}{\varepsilon d}\right)(1-\varepsilon)b_2 \pm \frac{a_2}{d} \\
&\in \left(1 \pm \frac{2}{\varepsilon d}\right)(1-\varepsilon)\left(\left(1 \pm \frac{2}{\varepsilon d}\right)(\varepsilon b_1 + a_1)\right) \pm \frac{1}{d}\left(\left(1 \pm \frac{2}{\varepsilon d}\right)(1-\varepsilon)b_1 \pm \frac{a_1}{d}\right) \\
&\in \left(1 \pm \frac{2}{\varepsilon d}\right)^3 (1-\varepsilon)(\varepsilon b_1 + a_1) \\
&= \left(1 \pm \frac{2}{\varepsilon d}\right)^3 a'_3 \in \left(1 \pm \frac{2}{\varepsilon d}\right)^6 a'_3.
\end{aligned}$$

We now turn to prove the induction step for i, assuming that it holds for $i-1$. Let us start with b_i. We have

$$\begin{aligned}
b_i &\in \left(1 \pm \frac{2}{\varepsilon d}\right)(\varepsilon b_{i-1} + a_{i-1}) && \text{(From (4.1).)} \\
&\in \left(1 \pm \frac{2}{\varepsilon d}\right)\left(\varepsilon \left(1 \pm \frac{2}{\varepsilon d}\right)^{2(i-1)} b'_{i-1} + \left(1 \pm \frac{2}{\varepsilon d}\right)^{2(i-1)} a'_{i-1}\right) && \text{(By the induction hypothesis.)} \\
&\in \left(1 \pm \frac{2}{\varepsilon d}\right)^{2i} \left(\varepsilon b'_{i-1} + a'_{i-1}\right) \\
&= \left(1 \pm \frac{2}{\varepsilon d}\right)^{2i} b'_i. && \text{(By (4.3).)}
\end{aligned}$$

Moreover, for a_i we have

$$\begin{aligned}
a_i &\in \left(1 \pm \frac{2}{\varepsilon d}\right)(1-\varepsilon)b_{i-1} \pm \frac{a_{i-1}}{d} && \text{(From (4.1).)} \\
&\in \left(1 \pm \frac{2}{\varepsilon d}\right)^{2(i-1)+1} (1-\varepsilon)b'_{i-1} \pm \left(1 \pm \frac{2}{\varepsilon d}\right)^{2(i-1)} \frac{a'_{i-1}}{d} && \text{(By the induction hypothesis.)} \\
&\in \left(1 \pm \frac{2}{\varepsilon d}\right)^{2i} (1-\varepsilon)b'_{i-1} && \text{(Since } a'_{i-1} \leq b'_{i-1}/\varepsilon \text{ for } i \geq 4.\text{)} \\
&\in \left(1 \pm \frac{2}{\varepsilon d}\right)^{2i} a'_i. && \text{(By (4.3).)}
\end{aligned}$$

This completes the proof of Claim 4.2.10. □

Observe that $a_1 + b_1 = 1$ since $u_1 \notin S$ and so either $u_1 \in A$ or $u_1 \in B$. Combined with Claim 4.2.10, we get that

$$\begin{aligned}
b_{\ell'} &\in \left(1 \pm \frac{2}{\varepsilon d}\right)^{2\ell'} \frac{(1-\varepsilon)(a_1+b_1) + (\varepsilon-1)^{\ell'}(a_1 - (1-\varepsilon)b_1)}{(2-\varepsilon)(1-\varepsilon)} \\
&\in \left(1 \pm \frac{2}{\varepsilon d}\right)^{2\ell'} \frac{1-\varepsilon + (\varepsilon-1)^{10\log n/\varepsilon}(a_1 - (1-\varepsilon)b_1)}{(2-\varepsilon)(1-\varepsilon)} && \text{(Since } \ell' \geq 10\log n\varepsilon.\text{)}
\end{aligned}$$

$$\in \left(1 \pm \frac{2}{\varepsilon d}\right)^{2\ell'+1} \frac{1-\varepsilon}{(2-\varepsilon)(1-\varepsilon)}$$
$$= \left(1 \pm \frac{1}{d}\right)^{O(\ell')} \frac{1}{2-\varepsilon}.$$

Observe that the conditioning $P(u_1)$ determines the values of a_1 and b_1. However, for large enough ℓ', as we see above, the dependence of $b_{\ell'}$ on a_1 and b_1 vanishes. In other words, by changing the conditioning $P(u_1)$ the value of $b_{\ell'}$ for $\ell' \geq 10 \log n/\varepsilon$ only changes by a $(1 \pm 1/d)^{O(\ell')}$ factor. The same also holds for $a_{\ell'}$. As such, whether vertex $u_{\ell'}$ belongs to A or B is essentially independent of the events above the root vertex u_1. To see why this implies Claim 4.2.9, take for example the $\mathcal{D}_{\mathsf{YES}}$ distribution and note that

$$\begin{aligned}\Pr_{\mathcal{D}_{\mathsf{YES}}}[T(u_{\ell'}) \mid P(u_1)] &= \Pr_{\mathcal{D}_{\mathsf{YES}}}[u_{\ell'} \in A \mid P(u_1)] \cdot \Pr_{\mathcal{D}_{\mathsf{YES}}}[T(u'_\ell) \mid u_{\ell'} \in A] \\ &+ \Pr_{\mathcal{D}_{\mathsf{YES}}}[u_{\ell'} \in B \mid P(u_1)] \cdot \Pr_{\mathcal{D}_{\mathsf{YES}}}[T(u'_\ell) \mid u_{\ell'} \in B] \\ &= a_{\ell'} \Pr_{\mathcal{D}_{\mathsf{YES}}}[T(u'_\ell) \mid u_{\ell'} \in A] + b_{\ell'} \Pr_{\mathcal{D}_{\mathsf{YES}}}[T(u'_\ell) \mid u_{\ell'} \in B].\end{aligned}$$

Since $a_{\ell'}$ and $b_{\ell'}$ remain the same up to a $(1 \pm 1/d)^{O(\ell')}$ factor by changing the conditioning $P(u_1)$ to $P'(u_1)$, we arrive at the desired inequality of Claim 4.2.9 for the $\mathcal{D}_{\mathsf{YES}}$ distribution. The proof for $\mathcal{D}_{\mathsf{NO}}$ is exactly the same. □

4.2.8 Limitation of the Algorithm

Let us define E_A to be the set of all discovered edges of $G[A]$ by the algorithm. Also, let $V_A = \{v \mid (u, v) \in E_A\}$, where here (u, v) is directed and u is the parent of v in the forest of discovered edges.

Claim 4.2.11. *With high probability, any algorithm $\mathcal{A}$ that makes at most $o(n^{6/5}/\log^2 n)$ queries, discovers at most $o(d/\log^2 n)$ of edges in subgraph G' with one endpoint in S.*

Proof. The proof is similar to the proof of Claim 4.2.5. Since we assume $\mathcal{A}$ makes at most $o(n^{6/5}/\log^2 n)$ queries, there will be at most $o(n^{1/5}/\log^2 n)$ vertices for which the algorithm makes more than $\varepsilon N/2 = \Omega(n)$ adjacency list queries. For these vertices, we assume that $\mathcal{A}$ finds its edge in G' with one endpoint in S (note that the degree of S vertices in G' is one and each vertex of G' is connected to at most one vertex of S).

Now let V' be the set of vertices for which $\mathcal{A}$ makes at most $\varepsilon N/2$ queries. For each new query to a vertex $v \in V'$, since there are εN edges to $T_U \cup T_V$ in G and that we have already made at most $\varepsilon N/2$ queries to v, the new query goes to a vertex in S with probability at most $O(1/n)$. Hence, by applying the Chernoff bound, with probability at least $1 - 1/\operatorname{poly}(n)$, there are at most $o(d/\log^2 n)$ edges in subgraph G' with one endpoint in S. □

Lemma 4.2.12. *Suppose that the algorithm $\mathcal{A}$ has made t queries for some $t \leq O(n^{6/5}/\log^2)$. Then with probability of $1 - o(1)$, all the following hold:*

(i) $\mathcal{A}$ has found at most $o(d)$ vertices in total for all subtrees $T(v)$ for $v \in V_A$ up to a distance $20(\log^2 n)/\varepsilon$ from root v.

(ii) $\mathcal{A}$ has not found any edge in G' with one endpoint in S in subtree $T(v)$ up to distance $20(\log^2 n)/\varepsilon$ from root v, for all $v \in V_A$.

(iii) Conditioning on what $\mathcal{A}$ has queried so far, the probability of each edge in F_t belonging to $G[A]$ is $O(1/d)$.

Proof. We prove the lemma using induction. For $t = 0$, trivially the statement is true. Now assume that the lemma holds for $t-1$ and $\mathcal{A}$ makes a new query. If the new queried edge is an edge to $T_U \cup T_V$, we are done since none of the conditions in the lemma statement will change. So we assume that the newly queried edge is in G'. We prove each of the three claims separately.

Induction step for (ii): if the newly queried edge is between B and S, the probability that one of its $20\log^2 n/\varepsilon$ most recent predecessors is an edge in $G[A]$ is $O(\log^2 n)/d)$ using the induction hypothesis (iii). Since $\mathcal{A}$ can discover at most $o(d/\log^2 n)$ edges in G' with one endpoint in S by Claim 4.2.11, then with probability at least $1 - o(\frac{\log^2 n}{d} \cdot \frac{d}{\log^2 n}) = 1 - o(1)$, the statement of (ii) remains true throughout all the steps of the induction.

Induction step for (i): the probability that at least one of the $20\log^2 n/\varepsilon$ most recent predecessors of the newly queried edge is an edge in $G[A]$ is $O(\log^2 n/d)$ using the induction hypothesis (iii). Since $\mathcal{A}$ discovers at most $o(n^{2/5}/\log^2 n) = o(d^2/\log^2 n)$ edges of G' by Claim 4.2.5, then with probability at least $1 - o(\frac{\log^2 n}{d^2} \cdot \frac{d^2}{\log^2 n}) = 1 - o(1)$, the statement of (i) remains true throughout all the steps of the induction.

Induction step for (iii): let (u, v) be a discovered edge in the forest (u is parent of v). First, we condition on $u \in A$, otherwise, the probability of (u, v) being an edge in $G[A]$ is zero. Note that u has at least d neighbors in G' that are not the parent of u in the forest F_t. By Claim 4.2.11, $\mathcal{A}$ discovers at most $o(d)$ edges in subgraph G' with one endpoint in S. Hence, at most $o(d)$ of neighbors of u in G' have an edge with one endpoint in S in their subtree. Furthermore, by Claim 4.2.5, at least $\Theta(d)$ neighbors of u in G', have at most $o(d)$ vertices in their subtree. Also, note that by proof of Lemma 4.2.4, neighbors of u in G' that are not adjacent to u in the forest are singleton vertices in the forest. Let $v_1, v_2, \ldots, v_r$ be the union of (1): children of u in the forest such that each of them has no edge with one S endpoint in their subtree up to distance $20\log^2 n/\varepsilon$, and each of them has $o(d)$ vertices in their subtree up to distance $20\log^2 n/\varepsilon$, (2): the neighbors of u in G' that are singleton in the forest. Hence, we have $r = \Theta(d)$.

By (i) and (ii), if $v \in A$ then there are at most $o(d)$ vertices in subtree $T(v)$ up to distance $20\log^2 n/\varepsilon$ from root v, and there is no edge in subtree $T(v)$ with one endpoint in S up to distance $20\log^2 n/\varepsilon$ from the root. Thus, if $v \notin \{v_1, v_2, \ldots v_r\}$, then (u, v) is not an edge in $G[A]$. Now assume that $v \in \{v_1, v_2, \ldots v_r\}$. Note that in $\mathcal{D}_{\mathsf{NO}}$, there is no neighbor of u with label A and in $\mathcal{D}_{\mathsf{YES}}$, there is exactly one neighbor in A. Therefore, for $\mathcal{D}_{\mathsf{NO}}$, the probability of $v \in A$ is zero. Now assume that the graph is drawn from $\mathcal{D}_{\mathsf{YES}}$. Hence, exactly one of $v_1, v_2, \ldots, v_r$ is in A (if parent of u is A then the probability of (u, v) being in $G[A]$ is zero). We prove that each of v_i can be in A with almost the same probability using a coupling argument.

Let c_i be the subset that vertex v_i belongs to ($c_i \in \{A, B\}$). We say $C = (c_1, c_2, \ldots, c_r)$ is a *profile* for labels of vertices $v_1, \ldots v_r$. Therefore, exactly one of c_i is equal to A and all others are equal to B which implies that there are r different possible profiles. Assume that C and C' are two different profiles. Let v_i be a vertex with label A in C and v_j be a vertex with label A in C'. By Lemma 4.2.7, the probability of sampling subtree below v_i with label B is the same up to a factor of $\left(1 \pm \frac{1}{d}\right)^{O(Q(T(v_i),\ell))}$. Similarly, the probability of sampling subtree below v_j with label A is the same up to a factor of $\left(1 \pm \frac{1}{d}\right)^{O(Q(T(v_j),\ell))}$. Since

$Q(T(v_j), \ell) = o(d)$, then the probability of having profile C and C' is the same up to a $(1 + o(1))$ factor. Therefore, the probability of having one specific v_i to be in A is $O(1/d)$ since $r = \Theta(d)$. □

4.2.9 Indistinguishability of the YES and NO distributions

We define a *bad event* to be the event of $\mathcal{A}$ discovering more than $o(d)$ vertices in total for all subtrees $T(v)$ for $v \in V_A$ up to a distance $20 \log^2 n/\varepsilon$ from root t, or $\mathcal{A}$ has found at least one edge in G' with one endpoint in S in subtree $T(v)$ up to distance $20 \log^2 n/\varepsilon$ from root v for at least one $v \in V_A$. By Lemma 4.2.12, the bad event happens with probability $o(1)$. For the next lemma, we condition on not having a bad event.

Lemma 4.2.13. *Let us condition on not having the bad event defined above. Let F be the final forest found by algorithm $\mathcal{A}$ on a graph drawn from $\mathcal{D}_{\mathsf{YES}}$ after at most $O(n^{6/5}/\log^2 n)$ queries. Then, the probability of querying the same forest in a graph that is drawn from $\mathcal{D}_{\mathsf{NO}}$ is at least almost as large, up to $1 + o(1)$ multiplicative factor.*

Proof. First, we define n hybrid distributions $\mathcal{D}_0, \mathcal{D}_1, \ldots, \mathcal{D}_{n-1}$ as follows. Distribution $\mathcal{D}_i$ can be obtained by sampling from $\mathcal{D}_{\mathsf{YES}}$ until the i-th level in any tree in the forest, and then sampling from $\mathcal{D}_{\mathsf{NO}}$ below the i-th level. Hence, $\mathcal{D}_0$ is exactly the same as $\mathcal{D}_{\mathsf{NO}}$ and $\mathcal{D}_{n-1}$ is the same as $\mathcal{D}_{\mathsf{YES}}$. Our goal is, starting from a forest sampled according $\mathcal{D}_{n-1} = \mathcal{D}_{\mathsf{YES}}$, to inductively show that we can switch from $\mathcal{D}_i$ to $\mathcal{D}_{i-1}$ with only negligble total decrease in the probability.

To formalize our coupling argument, recall the notion of *special edges* from the input distribution: In $\mathcal{D}_{\mathsf{YES}}$, edges in $G[A]$ are special; in $\mathcal{D}_{\mathsf{NO}}$, every vertex in A has one special edge to B all forming a matching. We also extend the definition of bad events to $\mathcal{D}_{\mathsf{NO}}$ and hybrid distribuitons to include bad events in subtrees originating from special edges.

Consider the forest F found by algorithm $\mathcal{A}$ on a graph drawn from $\mathcal{D}_{\mathsf{YES}}$, and let $F^{\leq i}$ denote the forest as well as any choice of its special edges in all levels $\leq i$ (arbitrarily, conditioning on not having a bad event). We compare the probability of seeing $F^{\leq i}$ when making the same queries when the oracle samples its answers from $\mathcal{D}_i$ vs $\mathcal{D}_{i-1}$. We can couple the sampling for the two distributions so that it is identical for everything in levels $< i$ (including the choice of special edges), and also for levels $\geq i$ in all the sub-trees that are not descendants of special level-i edges.

For each special edge $(u \to v)$ in the i-th level, the label of vertex v is A when sampling from $\mathcal{D}_i$ and B when sampling from $\mathcal{D}_{i-1}$. Below this vertex (aka levels $> i$), both distributions $\mathcal{D}_i$ and $\mathcal{D}_{i-1}$ sample according to $\mathcal{D}_{\mathsf{NO}}$. Thus, by Lemma 4.2.7, the probability of sampling the subtree below v is the same regardless of the label of v, up to a factor of $\left(1 \pm \frac{1}{d}\right)^{O(Q(T(v_i),\ell))}$.

Let $V_{A,i}$ denote the set of vertices pointed to by i-th level special edges. By the argument in the previous paragraphs, we have

$$\Pr[F^{\leq i} | F^{\leq i} \sim \mathcal{D}_{i-1}] \geq \Pr[F^{\leq i} | F^{\leq i} \sim \mathcal{D}_i] \left(1 - \frac{1}{d}\right)^{\sum_{v \in V_{A,i}} O(Q(T(v),\ell))}. \tag{4.4}$$

Let $\mathcal{G}^{\leq i}$ denote the event of having no bad events corresponding to level-($\leq i$) special edges. Summing over (4.4) for all valid choices of special edges, we have the following:

$$\Pr[F \wedge \mathcal{G}^{\leq i-1} | F \sim \mathcal{D}_{i-1}] \geq \Pr[F \wedge \mathcal{G}^{\leq i} | F \sim \mathcal{D}_{i-1}]$$

$$\geq \Pr[F \wedge \mathcal{G}^{\leq i} | F \sim \mathcal{D}_i] \cdot \left(1 - \frac{1}{d}\right)^{\sum_{v \in V_{A,i}} O(Q(T(v),\ell))}.$$

We can now bound the total distribution shift across all hybrid steps:

$$\begin{aligned}
\Pr[F|F \sim \mathcal{D}_0] &\geq \Pr[F \wedge \mathcal{G}^{\leq n-1} | F \sim \mathcal{D}_{n-1}] \cdot \prod_{i=0}^{n-1} \left(1 - \frac{1}{d}\right)^{\sum_{v \in V_{A,i}} O(Q(T(v),\ell))} \\
&= \Pr[F \wedge \mathcal{G}^{\leq n-1} | F \sim \mathcal{D}_{n-1}] \cdot \left(1 - \frac{1}{d}\right)^{\sum_{v \in V_A} O(Q(T(v),\ell))} \\
&\geq \Pr[F \wedge \mathcal{G}^{\leq n-1} | F \sim \mathcal{D}_{n-1}] \cdot \left(1 - \frac{1}{d}\right)^{o(d)} && \text{(By Lemma 4.2.12.)} \\
&\geq \Pr[F \wedge \mathcal{G}^{\leq n-1} | F \sim \mathcal{D}_{n-1}] \cdot (1 - o(1)).
\end{aligned}$$

This completes the proof of Lemma 4.2.13. □

Proof of Theorem 4.2.1. Note that the probability of having the bad event that we defined is $o(1)$. Conditioning on not having the bad event, by Lemma 4.2.13, the distribution of the outcome that the algorithm discovers is in $o(1)$ total variation distance for $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$. Therefore, the algorithm is not able to distinguish between the support of two distributions with constant probability. According to our construction, the size of the maximum matching of both $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$ is $\Theta(n)$, and in both cases, the input graph is bipartite. □

4.3 Lower Bound for Bounded Degree Graphs

In this section, we prove a lower bound for the bounded degree regime. In the sublinear time model, Yoshida, Yamamoto, and Ito [176] showed there exists an $\Delta^{O(1/\varepsilon^2)}/\varepsilon^2$ time algorithm providing a $(1, \varepsilon n)$-approximation of maximum matching size. Whether there exists a $\text{poly}(\Delta/\varepsilon)$ time algorithm has remained open for more than a decade. Theorem 4.3.1 negatively resolves this question by showing that $\Delta^{\Omega(1/\varepsilon)}$ time is necessary.

Theorem 4.3.1. *Let $\varepsilon \leq 0.01$. For any choice of $\log^4 n \leq \Delta \leq n^{\varepsilon}$, there is an n-vertex bipartite graph G of maximum degree Δ such that any randomized algorithm that with probability at least 0.51 provides a $(1, \varepsilon n)$-approximation for the size of the maximum matching in G must make $\Delta^{\Omega(1/\varepsilon)}$ adjacency list queries.*

Theorem 4.3.2. *Let $\varepsilon \leq 0.01$. For any choice of $\log^4 n \leq \Delta \leq n^{\varepsilon}$, there is an n-vertex bipartite graph G of maximum degree Δ such that any LCA that with probability at least 0.51 computes a $(1, \varepsilon n)$-approximate maximum matching of G must make at least $\Delta^{\Omega(1/\varepsilon)}$ queries to G.*

Compared to prior lower bounds, several substantially new ideas are needed in the proof of Theorem 4.3.1. The main novelty of our proof is a new notion of *delusive vertices*. These are a total of $\Theta(\varepsilon n)$ vertices in the graph decomposed into $O(1/\varepsilon)$ levels of $O(\varepsilon^2 n)$ vertices each that essentially do not participate in a $(1, \varepsilon n)$-approximate maximum matching, but distinguishing them from those vertices that do participate in

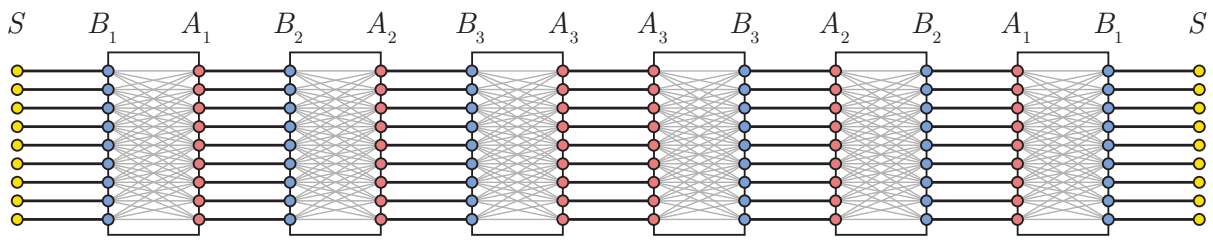


Figure 4.3: Core of the construction when $k = 3$.

the matching turns out to require a large number of queries. We present a detailed overview of these delusive vertices and our techniques in Section 4.3.1.

4.3.1 Technical Overview

In this section, we present a high-level and informal overview of our lower bound of Theorem 4.3.2, deferring the formal proofs to the forthcoming sections.

The Input Graph

We start by describing the input distribution. As the final construction might seem strange at the first glance, we present it step by step, gradually adding all the ingredients that are needed for the final proof. Note that the degree of construction outlined in the technical overview differs slightly from the actual construction, but this overview contains all the essential ideas.

Step 1 — The Core: The first step is simple and intuitive. The "core" of our input graph consists of a set S of vertices of degree 1. The core, in addition, has $k = \Theta(1/\varepsilon)$ vertex subsets A_i, B_i for $i \in [k]$. There are two types of edges in the core as illustrated in Figure 4.3 for $k = 3$. There are 'dense blocks' of d-regular graphs between A_i and B_i for any $i \in [k]$. Additionally, there are 'special edges' perfectly matching S to B_1, A_i to B_{i+1} for any $i \in [k-1]$, and A_k to A_k.

Note that the special edges combined form a maximum matching of the core. Importantly, any $(\frac{2k+1/2}{2k+1} \sim 1 - O(\varepsilon))$-approximate maximum matching of the core must include a constant fraction of the special edges going from A_k to A_k. Our goal is to hide these special edges and show that finding each one of them requires at least $d^{k-o(1)}$ queries to the graph. To do this, it is important *not* to give away the layer of a vertex. Towards this, our first idea is to assign a random ID to each of the vertices of the core and sort the adjacency lists randomly.

The nice thing about the core is that the local neighborhoods of all the vertices in higher levels are symmetric. In particular, it is not possible to distinguish an A_k vertex v from a B_k vertex without reaching an S vertex in its neighborhood, which are all at distance at least $2k$ from v. Note that while there are indeed $\Omega(d^k)$ vertices in the $2k$-hop of a vertex $v \in A_k$, an LCA is not obligated to explore the whole $2k$-hop of v. In fact, a random walk starting from any vertex v reaches an S vertex in just $O_\varepsilon(d)$ steps in expectation. Moreover, the distribution of the length of such a random walk until reaching S (which can be approximated sufficiently well with some $O_\varepsilon(\log n)$ repetitions) is enough to determine the layer of its starting vertex correctly with high probability. Therefore, we need more ideas to hide the layers of the core.

Step 2 — Delusive Vertices: Delusive vertices are a key component of our construction. Their main purpose is to guarantee what we showed the core cannot do on its own: hiding its layers. In our final construction, we will have a hierarchy of delusive vertices. But let us start with one level and see how it helps. We add a set D of $\Theta(\frac{\varepsilon}{1+\varepsilon}n)$ delusive vertices to the graph. We connect every vertex in $\{A_i, B_i\}_{i\in[k]}$ to εd delusive vertices in D.[1] This can be done in a way such that all the A_i, B_i, D vertices have the same degree $d' = d + \varepsilon d + 1$ overall, and each vertex in D has the same number of edges to all of the A_i, B_i layers.

It turns out that adding these delusive vertices is enough to kill the random-walk based algorithm outlined above. Indeed, because ε fraction of neighbors of each A_i, B_i vertex goes to D, the random walk is expected to hit D every $\Theta(1/\varepsilon)$ steps. As this is much smaller than the $\Omega(d)$ expected steps to hit an S vertex, the random walk, w.h.p., sees a D vertex before reaching S. On the other hand, the moment that we hit D, we completely lose information about where the random walk started. This is because conditioned on having reached a delusive vertex $u \in D$, all the layers have the same probability of being u's predecessor in the walk as u has the same degrees to all the layers.

While one layer of delusive vertices kills the random walk algorithm, it does not yet imply that $d^{k-o(1)}$ queries are needed for determining the label of A_k vertices. In fact, it is still possible to determine the label of any vertex in just $\widetilde{O}(d^2)$ time! To see this, observe first that it is possible to determine whether a vertex is a B_1 vertex in $O(d)$ time by simply scanning its neighbors and checking whether there is an S vertex among them. Now suppose that our task is to determine whether a vertex v belongs to D. Since only the vertices in D have ε fraction of their neighbors in B_1, we can random sample $\widetilde{O}(1)$ neighbors of v, check which ones belong to B_1, and report $v \in D$ iff this fraction is sufficiently close to ε. Now that we can check if a vertex belongs to D in $\widetilde{O}(d)$ time, we can modify the random walk algorithm, ensuring that we never step on a D vertex by running this test on each vertex that it visits. This only multiplies the running time of the random walk algorithm by a $\widetilde{O}(d)$ factor, thus it takes $\widetilde{O}(d^2)$ time to determine the core layers with one level of delusive vertices.

Step 3 — A Hierarchy of Delusive Vertices: In our final construction, instead of just a single layer of delusive vertices, we have a hierarchy of $k = \Theta(1/\varepsilon)$ levels of delusive vertices $D_1, \ldots, D_k$. We ensure that the total number of vertices in $D_1, \ldots, D_k$ is $O(\varepsilon^2 n)$ so that adding them to the graph does not drastically change the maximum matching of the core. As illustrated in Figure 4.4, for any i, vertices in D_i are made adjacent to A_j, B_j for all $j \geq i$ and to all D_j for $j > i$. Intuitively, while we can still check whether $v \in D_1$ in $\widetilde{O}(d)$ time by examining what fraction of its neighbors belongs to B_1, the same cannot be done for $D_2, D_3, \ldots$ as they do not have any direct neighbors in B_1. In particular, determining whether a vertex v belongs to D_i (or even A_i, B_i) will require $d^{i-o(1)}$ queries in the neighborhood of v which effectively hides the core layers.

Step 4 — Binomial Degrees: The 4^{th} and last step of our construction is more of a technical modification to the construction discussed above that is important for our proofs. In the graph illustrated above, each vertex has a fixed number of edges to every layer. Take a vertex $v \in B_1$ for example. It has one neighbor in S, d neighbors in A_1, and εd neighbors in D_1. In our final construction, we want every neighbor of $v \in B_1$ to belong to A_1, D_1, S independently from the rest of neighbors of v. To achieve this, we first draw the number of edges of v to each of A_1, D_1, S from a suitable binomial distribution with the right expected value and

[1] We note that after connecting the D vertices to all of A_i, B_i, the resulting graph will no longer be bipartite. Minor modifications will be needed to convert the graph into a bipartite one.

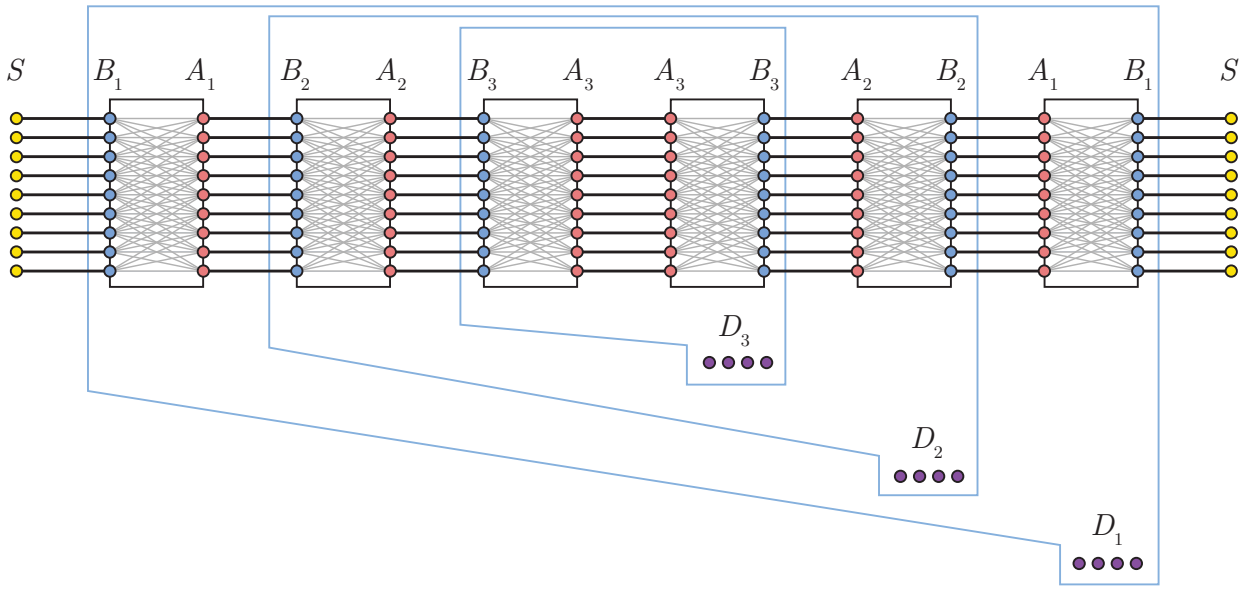


Figure 4.4: This figure shows how the k levels of delusive vertices are made adjacent to the core. For simplicity, this figure does not show the edges of the delusive vertices, but all the edges of each D_i vertex goes to the vertices in the smallest blue box enclosing it.

then try to satisfy these drawn degrees. A challenge that arises is that the drawn degree sequences of all vertices might not be realizable simultaneously. For instance, if the sum of degrees of B_1 to D_1 is not the same as the sum of degrees from D_1 to B_1, then clearly the graph is not realizable. Nonetheless, we show that by modifying the drawn degrees of a small number of "broken vertices", the resulting degree sequence will be realizable using a theorem of Gale-Ryser (see Proposition 2.2.7). We also show that the algorithm will, w.h.p., never see a broken vertex. Effectively, this implies that the layers of the neighbors of any vertex that the algorithm sees will be independent.

Formalizing the Lower Bound: The Label Guessing Game on Trees

Up to this point, we have presented a high-level overview of our input graph and have also explained why a certain random-walk based algorithm cannot find a $(1, \varepsilon n)$-approximate matching of it with less than $d^{\Omega(1/\varepsilon)}$ queries. In this section, we overview how we prove this lower bound against all algorithms.

The Label Guessing Game on Trees: We reduce our lower bound to a clean "label guessing game" on a Markovian tree (see Figure 4.5). In this problem, we have a tree T which initially only involves a single vertex v that is going to be the root of T throughout. At each step, the algorithm can adaptively pick a vertex $u \in T$ of its choice. Doing so will add a direct child below u. Each vertex added to T will have a *hidden* label. The goal is to guess the label of the root vertex v while querying a few vertices in its subtree. The hidden labels correspond to the vertex subsets of our input distribution. That is, each vertex has one label that is either S or A_i, B_i, D_i for some $i \in [k]$. The labels of the children of each vertex u are drawn independently from a distribution that depends only on the label of their parent u. These transition probabilities come from our input distribution. For example, each A_1 vertex in our input graph has d expected neighbors in B_1, εd expected neighbors in D_1, and $\widetilde{O}(1)$ expected neighbors in B_2. Thus, once we open a child w for a vertex

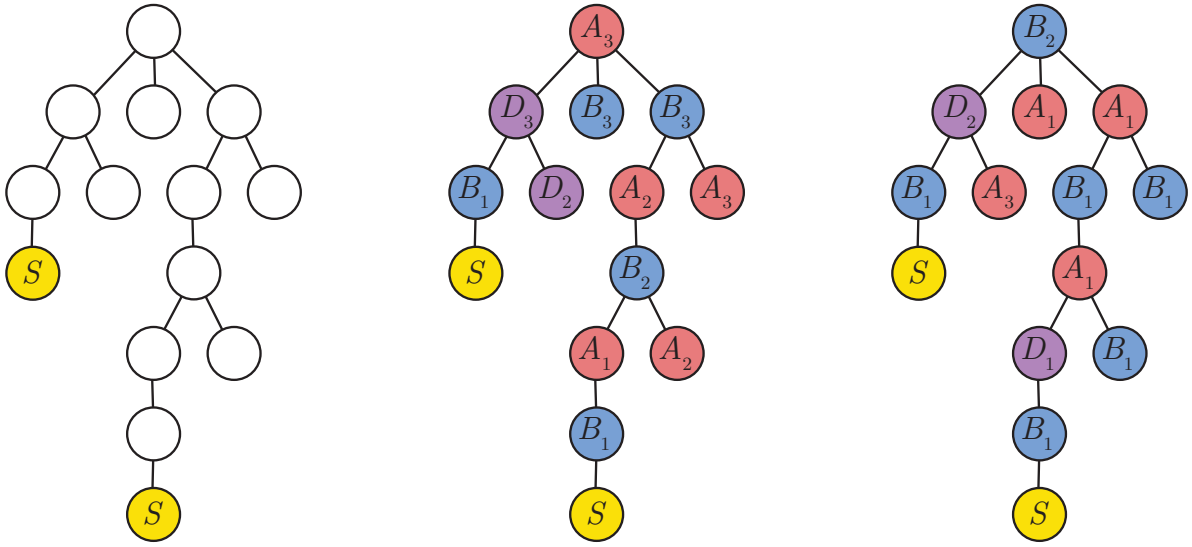


Figure 4.5: An example of the label guessing game. The tree on the left is what the algorithm sees. In particular, all the labels except for the S labels are hidden from the algorithm. On the right, we have two possible realizations of the labels leading to the same observed tree. The algorithm must pick its queries in such a way that it can guess the label of the root.

u whose hidden label is A_1, its child w takes label B_1 with probability $1 - \Theta(\varepsilon)$, label D_1 with probability $\Theta(\varepsilon)$, and takes label B_2 with probability $\widetilde{\Theta}(1/d)$ independently. The only information that the algorithm is given is whether the label of each vertex in the tree is S or not. Figure 4.5 shows an instance of the label guessing game and two of its possible realizations.

The Reduction to the Label Guessing Game: We show that any LCA for $(1, \varepsilon n)$-approximate matching for our input construction leads to an efficient label guessing algorithm in the tree model. To show this, we prove that any LCA that queries $d^{O(1/\varepsilon)}$ entries of the graph, with high probability, only sees a (rooted) forest. The proof relies heavily on the fact that the edges of the input graph G are sufficiently random (even conditioned on satisfying the degree constraints and conditioned on the previous $d^{O(1/\varepsilon)}$ queries) and thus expand well. Once we prove this, we are immediately done: conditioned on the high probability event that the LCA does not discover a cycle, the problem becomes exactly the same as the label guessing game.

Lower Bounds for the Label Guessing Game: Lower bounding the number of queries needed to solve the label guess game is the crux of our analysis. Our proof consists of two parts. In the first part of the proof, we show that any algorithm that solves the label guessing game must find a path from the root to an S vertex that does not go through a certain subset of delusive vertices that we call mixer vertices (Definition 4.3.15). To formalize this, via a careful coupling argument, we show that if every path from the root to an S vertex contains a mixer vertex, then the label of the root is equally likely to be, say, A_k or B_k. In the second part of the proof, we prove that to discover a path from root of level k to S that does not contain any mixer vertex, the subtree below the root must include at least $d^{k-o(1)}$ vertices. The proof of this is close (but more general) than the arguments we discussed above for why the random-walk based algorithm does not work.

4.3.2 Input Distribution and its Characteristics

In this section, we describe the input distribution of our construction. We have two types of input graphs where the first graph has an almost perfect matching and for the second graph, only $(1-\varepsilon)$ fraction of vertices are matched in the maximum matching. We prove that any *deterministic* LCA which with probability at least 0.51 computes a $(1, \varepsilon n)$-approximate maximum matching of graphs drawn from this distribution, must spend at least $\Delta^{\Omega(1/\varepsilon)}$ time. From Yao's minimax theorem [175], we thus get that any *randomized* LCA that computes a $(1, \varepsilon n)$-approximate matching for all inputs with success probability at least 0.51 must also spend at least $\Delta^{\Omega(1/\varepsilon)}$ time per query.

Let N be a parameter that controls the number of vertices in our input distribution. Moreover, in our construction, let $d \leq n^{\varepsilon/3}$ be a parameter that controls the degree of vertices. Graphs in our input distribution have $n = (1/2 + 1/\varepsilon + \varepsilon - \varepsilon^2/2)N$ vertices. We first describe the vertex set of the graphs in our distribution.

The vertex set: The vertex set consists of disjoint subsets $A_i^1, B_i^1, A_i^2, B_i^2$ for each $i \in [1/\varepsilon]$ as well as two subsets S^1 and S^2. Each of these subsets except $A_{1/\varepsilon}^1, A_{1/\varepsilon}^2, S^1$, and S^2, has exactly $N/4$ vertices. Each of $A_{1/\varepsilon}^1$ and $A_{1/\varepsilon}^2$ has $(1-\varepsilon^2)N/4$ vertices. Also, each of S^1 and S^2 has $N/4$ vertices. Moreover, the vertex set consists of subsets D_i of *delusive vertices* for $i \in [1/\varepsilon]$, where each of these $1/\varepsilon$ subsets has exactly $\varepsilon^2 N$ vertices. So we have

$$|A_i^1| = |A_i^2| = |B_i^1| = |B_i^2| = \frac{N}{4} \qquad \forall i \in [\frac{1}{\varepsilon} - 1],$$

$$|B_{1/\varepsilon}^1| = |B_{1/\varepsilon}^2| = \frac{N}{4}, \quad |A_{1/\varepsilon}^1| = |A_{1/\varepsilon}^2| = (1-\varepsilon^2)\frac{N}{4},$$

$$|S^1| = |S^2| = \frac{N}{4},$$

$$|D_i| = \varepsilon^2 N \quad \forall i \in [\frac{1}{\varepsilon}].$$

Hence, the total number of vertices in each graph of our input distribution is $n = (1/2 + 1/\varepsilon + \varepsilon - \varepsilon^2/2)N$.

The edge set: For the edge set, we have two different distributions; $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$. In $\mathcal{D}_{\mathsf{YES}}$, the graph has an almost perfect matching. On the flip side, a maximum matching of $\mathcal{D}_{\mathsf{NO}}$ leaves at least εn vertices unmatched. In our input distribution, we draw the graph from $\mathcal{D}_{\mathsf{YES}}$ with probability $1/2$ and from $\mathcal{D}_{\mathsf{NO}}$ with probability $1/2$.

Let $X, Y \in \bigcup_{i=1}^{1/\varepsilon}\{A_i^1, A_i^2, B_i^1, B_i^2, D_i\} \cup \{S^1, S^2\}$ be any two vertex subsets. We use $\deg_X^Y(v)$ to denote the number of vertices of subset Y that are adjacent to a single vertex $v \in X$. First, we show how the degree of vertices will be determined in $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$, then we describe how to construct a graph with the corresponding degree sequence. Each vertex except vertices of S has exactly $d' = d + \varepsilon^3 d + \log^4 N$ neighbors. Also, all vertices of S have $\log^4 N$ neighbors. For a vertex $u \in X$, the type of its neighbor $v \in Y$ is determined independently at random according to the following binomial distribution for both $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$ (it helps to recall Figure 4.4 of Section 4.3.1):

- Vertices of S^j have $\log^4 N$ neighbors and the neighbors only can be B_1^j for $j \in \{1,2\}$.
- If $X = B_1^j$ for $j \in \{1,2\}$:

$$\Pr[Y = S^j] = \frac{\log^4 N}{d'}, \quad \Pr[Y = A_1^j] = \frac{d}{d'},$$

$$\Pr[Y = D_1] = \frac{\varepsilon^3 d}{d'},$$

- If $X = B_i^j$ for $j \in \{1,2\}$ and $1 < i < 1/\varepsilon$:

$$\Pr[Y = A_{i-1}^j] = \frac{\log^4 N}{d'}, \quad \Pr[Y = A_i^j] = \frac{d}{d'},$$

$$\Pr[Y = D_i] = \frac{(1/\varepsilon - i + 1)\varepsilon^4 d}{d'},$$

$$\Pr[Y = D_k] = \frac{\varepsilon^4 d}{d'} \quad \text{for } k < i.$$

- If $X = A_i^j$ for $j \in \{1,2\}$ and $1 \le i < 1/\varepsilon$:

$$\Pr[Y = B_{i+1}^j] = \frac{\log^4 N}{d'}, \quad \Pr[Y = B_i^j] = \frac{d}{d'},$$

$$\Pr[Y = D_i] = \frac{(1/\varepsilon - i + 1)\varepsilon^4 d}{d'},$$

$$\Pr[Y = D_k] = \frac{\varepsilon^4 d}{d'} \quad \text{for } k < i.$$

- If $X = D_i$ for $i \in [1/\varepsilon - 1]$:

$$\Pr[Y = D_i] = \frac{(1 - 2\varepsilon + 2i\varepsilon^2 - 5\varepsilon^2/2 + 3\varepsilon^4)d + \log^4 N}{d'},$$

$$\Pr[Y = D_j] = \frac{\varepsilon^4 d}{d'} \quad \text{for } j \neq i,$$

$$\Pr[Y = A_{1/\varepsilon}^j] = \frac{(\varepsilon^2 - \varepsilon^4)d}{d'} \quad \text{for } j \neq i,$$

$$\Pr[Y = A_i^j] = \Pr[Y = B_i^j] = \frac{(1/\varepsilon - i + 1) \cdot \varepsilon^2 d/4}{d'}$$
$$\text{for } j \in \{1,2\},$$

$$\Pr[Y = A^j_k] = \Pr[Y = B^j_k] = \Pr[Y = B^j_{1/\varepsilon}] = \frac{\varepsilon^2 d/4}{d'}$$
$$\text{for } j \in \{1,2\} \text{ and } i < k < 1/\varepsilon.$$

- If $X = D_i$ for $i = 1/\varepsilon$:

$$\Pr[Y = D_i] = \frac{(1 - 5\varepsilon^2/2 + 3\varepsilon^4)d + \log^4 N}{d'},$$

$$\Pr[Y = D_j] = \frac{\varepsilon^4 d}{d'} \quad \text{for } j \neq i,$$

$$\Pr[Y = A^j_{1/\varepsilon}] = \frac{(\varepsilon^2 - \varepsilon^4)d}{d'} \quad \text{for } j \in \{1,2\},$$

$$\Pr[Y = B^j_{1/\varepsilon}] = \frac{\varepsilon^2 d/4}{d'} \quad \text{for } j \in \{1,2\}.$$

Distribution of neighbors of vertices in $A^j_{1/\varepsilon}$ and $B^j_{1/\varepsilon}$ for $j \in \{1,2\}$ is different in $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$. The following binomial distribution is the distribution of neighbors in $\mathcal{D}_{\mathsf{YES}}$:

- If $X = B^j_{1/\varepsilon}$ for $j \in \{1,2\}$:

$$\Pr[Y = A^j_{1/\varepsilon - 1}] = \frac{\log^4 N}{d'}, \quad \Pr[Y = A^j_{1/\varepsilon}] = \frac{(1-\varepsilon^2)d}{d'},$$

$$\Pr[Y = B^{3-j}_{1/\varepsilon}] = \frac{\varepsilon^2 d}{d'},$$

$$\Pr[Y = D_k] = \frac{\varepsilon^4 d}{d'} \quad \text{for } k \leq 1/\varepsilon.$$

- If $X = A^j_{1/\varepsilon}$ for $j \in \{1,2\}$:

$$\Pr[Y = A^{3-j}_{1/\varepsilon}] = \frac{\log^4 N}{d'}, \quad \Pr[Y = B^j_{1/\varepsilon}] = \frac{d}{d'},$$

$$\Pr[Y = D_k] = \frac{\varepsilon^4 d}{d'} \quad \text{for } k \leq 1/\varepsilon.$$

The following binomial distribution is the distribution of neighbors in $\mathcal{D}_{\mathsf{NO}}$:

- If $X = B^j_{1/\varepsilon}$ for $j \in \{1,2\}$:

$$\Pr[Y = A^j_{1/\varepsilon - 1}] = \frac{\log^4 N}{d'},$$

$$\Pr[Y = A^j_{1/\varepsilon}] = \frac{(1-\varepsilon^2)(d+\log^4 N)}{d'},$$

$$\Pr[Y = B^{3-j}_{1/\varepsilon}] = \frac{\varepsilon^2(d+\log^4 N) - \log^4 N}{d'},$$

$$\Pr[Y = D_k] = \frac{\varepsilon^4 d}{d'} \quad \text{for } k \le 1/\varepsilon.$$

- If $X = A^j_{1/\varepsilon}$ for $j \in \{1,2\}$:

$$\Pr[Y = B^j_{1/\varepsilon}] = \frac{d+\log^4 N}{d'},$$

$$\Pr[Y = D_k] = \frac{\varepsilon^4 d}{d'} \quad \text{for } k \le 1/\varepsilon.$$

Note that for two different subsets X and Y, we only described how to determine $\deg^Y_X(v)$. Unfortunately, it may not be possible to construct a graph with the resulting degree sequence. As described in Section 4.3.1, if the sum of degrees from A^1_1 to B^1_1 is different from the sum of degrees from B^1_1 to A^1_1, then no graph can satisfy this degree sequence. Nonetheless, we prove that by ignoring the degrees of at most $O(\sqrt{nd}\cdot\log n)$ vertices, which we call *broken vertices*, the degrees of the rest of the vertices can be satisfied with high probability. The following Lemma 4.3.4 is useful in showing how we add the edges according to the degree sequence.

Definition 4.3.3 (Broken Vertices)**.** *Take a vertex v in our input graph. We say v is a broken vertex if its degree in the final graph is different from the degree initially drawn from the binomial distribution.*

Lemma 4.3.4. *Let $a_1 \ge a_2 \ge \ldots \ge a_{k_1}$ and $b_1 \ge b_2 \ge \ldots \ge b_{k_2}$ be two sequences of non-negative integers where a_i is drawn from a Binomial distribution with η_1 trials and success probability ρ_1, and b_i is drawn from a Binomial distribution with η_2 trials and success probability ρ_2 for all i, and suppose $\sum_i^{k_1} a_i \ge \sum_i^{k_2} b_i$. Also, assume that $k_1\eta_1\rho_1 = k_2\eta_2\rho_2$, $\eta_1\rho_1 = \Omega(\log^4 n)$, $\eta_2\rho_2 = \Omega(\log^4 n)$, $\eta_1\rho_1 = O(d)$, $\eta_2\rho_2 = O(d)$, $k_1 = \Theta(n)$, and $k_2 = \Theta(n)$. Then, with high probability, there exists a sequence of non-negative integers $a'_1 \ge a'_2 \ge \ldots \ge a'_{k_1}$ such that all the following hold:*

- *$0 \le a_i - a'_i \le 10\sqrt{\eta_1\rho_1}\log n$ for all i,*
- *$(a'_1, a'_2, \ldots, a'_{k_1})$ and $(b_1, b_2, \ldots, b_{k_2})$ is a bigraphic pair of sequences (see Definition 2.2.6), and*
- *there are at most $O(\sqrt{k_1\eta_1\rho_1}\cdot\log n)$ elements in the sequence where $a_i \ne a'_i$.*

Proof. First, note that since both sequences are drawn from a binomial distribution, by applying a Chernoff bound, with a probability of at least $1-2n^{-5}$, we have $a_i \in (\eta_1\rho_1 \pm 5\sqrt{\eta_1\rho_1}\cdot\log n)$ (resp., $b_i \in (\eta_2\rho_2 \pm 5\sqrt{\eta_2\rho_2}\cdot\log n)$). Thus, using a union bound, with a high probability this event holds all for a_i's and b_i's.

Similarly, using the Chernoff bound, we get that with high probability,

$$\sum_i^{k_1} a_i \in \left(k_1\eta_1\rho_1 \pm O(\sqrt{k_1\eta_1\rho_1}\cdot\log n)\right),$$
$$\sum_i^{k_2} b_i \in \left(k_2\eta_2\rho_2 \pm O(\sqrt{k_2\eta_2\rho_2}\cdot\log n)\right).$$

Let $D = \sum_i^{k_1} a_i - \sum_i^{k_2} b_i$. Since $k_1\eta_1\rho_1 = k_2\eta_2\rho_2$, by the above bounds, $D \le O(\sqrt{k_1\eta_1\rho_1}\cdot\log n)$. We construct a degree sequence $a' = (a'_1, a'_2, \ldots, a'_{k_1})$ in D iterations. Initially, we set $a'_i = a_i$ for all i. At each iteration, we choose the maximum a'_i and reduce its value by one. In the end, we sort a' in decreasing order. Note that according to the construction, we have $\sum_i^{k_1} a_i - \sum_i^{k_1} a'_i = D \le O(\sqrt{k_1\eta_1\rho_1}\cdot\log n)$. Furthermore, the maximum of a' cannot be less than $\eta_1\rho_1 - 5\sqrt{\eta_1\rho_1}\cdot\log n$, as otherwise, $\sum_i^{k_1} a_i$ should be significantly less than $k_1\eta_1\rho_1 - O(\sqrt{k_1\eta_1\rho_1}\cdot\log n)$ which is a contradiction. Thus, $0 \le a_i - a'_i \le 10\sqrt{\eta_1\rho_1}\cdot\log n$ for all $1 \le i \le k_1$. Therefore, it remains to show that $(a'_1, a'_2, \ldots, a'_{k_1})$ and $(b_1, b_2, \ldots, b_{k_2})$ is a bigraphic pair of sequences.

For this aim, we use Gale–Ryser theorem in Proposition 2.2.7. We need to show that the conditions in this theorem hold for the pair of sequences. Formally, for each $1 \le r \le k_1$, we claim that $\sum_i^r a'_i \le \sum_i^{k_2} \min(b_i, r)$. If $r \ge \eta_2\rho_2 + 5\sqrt{\eta_2\rho_2}\cdot\log n$, then

$$\sum_i^{k_2} \min(b_i, r) = \sum_i^{k_2} b_i \ge \sum_i^{r} a'_i,$$

where the first equality follows by the high probability event of having $b_i \le \eta_2\rho_2 + 5\sqrt{\eta_2\rho_2}\cdot\log n$ for all $1 \le i \le k_2$. If $r < \eta_2\rho_2 + 5\sqrt{\eta_2\rho_2}\cdot\log n$, then

$$\begin{aligned}\sum_i^{r} a'_i &\le r\cdot(\eta_1\rho_1 + 5\sqrt{\eta_1\rho_1}\cdot\log n)\\ &\le O(d^2) \le n \le \sum_i^{k_2}\min(b_i, r),\end{aligned}$$

which completes the proof. □

Corollary 4.3.5. *Let $(a_1, a_2, \ldots, a_n)$ be the degree sequence that is produced by the construction. Then, there exists a graph with sequences $(b_1, b_2, \ldots, b_n)$ such that there exists at most $O(\sqrt{nd}\log n)$ broken vertices in the constructed graph.*

Proof. Proof follows by applying Lemma 4.3.4 for the degree sequence of induced subgraph for all pairs (X, Y) such that $X, Y \in (S^1 \cup S^2) \cup (\bigcup_{i=1}^{1/\varepsilon}\{A_i^1, A_i^2, B_i^1, B_i^2, D_i\})$. □

Remark 4. *Note that there are edges inside D_i for each $i \in [1/\varepsilon]$, hence, we cannot use Lemma 4.3.4 to put edges in $G[D_i]$ since the graph is not bipartite. However, we can assume that the number of vertices in each D_i is even, and there are two parts in each D_i where vertices of each part are only connected to the other part. With this small modification, we can use Lemma 4.3.4 for $G[D_i]$.*

Edges of the graph: We use Corollary 4.3.5 to construct a graph with the given degree sequence that we determined before. By Corollary 4.3.5, there are at most $O(\sqrt{nd}\log n)$ broken vertices. Distribution $\mathcal{D}_{\mathsf{YES}}$ (resp., $\mathcal{D}_{\mathsf{NO}}$) picks a graph uniformly from the set of all possible graphs that satisfy the modified degree sequence corresponding to $\mathcal{D}_{\mathsf{YES}}$ (resp., $\mathcal{D}_{\mathsf{NO}}$).

Now we observe some properties of the input distribution that are immediately implied by the construction and important for the proof.

Observation 4.3.6. *For any graph that is drawn from the input distribution, with high probability, there exists at most $O(\sqrt{nd}\log n)$ broken vertices.*

Proof. The proof follows by Corollary 4.3.5. □

Claim 4.3.7. *With high probability, all the following hold:*

1. *There exists a matching of size $(1-\varepsilon^3)N/4$ between vertices of S^j and B_1^j for all $j \in \{1,2\}$ for all $j \in \{1,2\}$.*
2. *There exists a matching of size $(1-\varepsilon^3)N/4$ between vertices of A_i^j and B_{i+1}^j for all $i \in [1/\varepsilon - 1]$ and $j \in \{1,2\}$.*
3. *If the input graph is drawn from $\mathcal{D}_{\mathsf{YES}}$, then there exists a matching of size $(1-2\varepsilon^2)N/4$ between $A_{1/\varepsilon}^1$ and $A_{1/\varepsilon}^2$.*

Proof. Let $v \in A_i^j \cup B_{i+1}^j$. Degree of vertex v in $G[A_i^j, B_{i+1}^j]$ is concentrated around $\log^4 N$ with $10\log^3 N$ error since the expected degree is $\log^4 N$, we can show that using a standard Chernoff bound, the error is at most $10\log^3 N$ with high probability. Furthermore, by Lemma 4.3.4, the degree of a vertex can decrease by $10\log^3 N$ additive value when we put edges in the graph using Lemma 4.3.4. Thus, the degree cannot be smaller than $\log^4 N - 20\log^3 N$ and larger than $\log^4 N + 10\log^3 N$. We construct a fractional matching such that for each edge e in $G[A_i^j, B_{i+1}^j]$, we set $f_e = 1/(\log^4 N + 10\log^3 N)$. Since the degree is at most $\log^4 N + 10\log^3 N$, this fractional matching is feasible. Let $E(v)$ be the set of edges incident to v in $G[A_i^j, B_{i+1}^j]$, then due to the integrality gap of the fractional matching polytope in bipartite graphs, we have

$$\begin{aligned}\mu(G[A_i^j, B_{i+1}^j]) \geq \sum_{v \in A_i^j} \sum_{e \in E(v)} f_e &\geq \sum_{v \in A_i^j} \frac{\log^4 N - 20\log^3 N}{\log^4 N + 10\log^3 N} \\ &\geq \frac{N}{4} \cdot \left(1 - \frac{40}{\log N}\right) \\ &\geq (1-\varepsilon^3)\frac{N}{4},\end{aligned}$$

concluding the proof for statement (2).[2] A similar argument also works for statement (1) since the degrees and sizes of subgraphs are the same.

Proof of the third statement is similar to the second statement since the degree of vertices in $G[A_{1/\varepsilon}^1, A_{1/\varepsilon}^2]$ is concentrated around $\log^4 N$ with $10\log^3 N$ error. Let $E(v)$ be the set of edges incident to v in $G[A_{1/\varepsilon}^1, A_{1/\varepsilon}^2]$.

[2] In the proof of this lemma, we need ε to be constant. However, we might use a slightly modified version of the result by [92] to show that there exists a perfect matching in $G[A_i^j, B_{i+1}^j]$ and $G[A_{1/\varepsilon}^1, A_{1/\varepsilon}^2]$. With this change, we do not need the assumption for ε to be constant.

If we construct the same fractional matching, then we get

$$\begin{aligned}\mu(G[A^1_{1/\varepsilon}, A^2_{1/\varepsilon}]) &\geq \sum_{v\in A^1_{1/\varepsilon}} \sum_{e\in E(v)} f_e \\ &\geq \sum_{v\in A^1_{1/\varepsilon}} \frac{\log^4 N - 20\log^3 N}{\log^4 N + 10\log^3 N} \\ &\geq \frac{(1-\varepsilon^2)N}{4}\cdot\left(1-\frac{40}{\log N}\right) \\ &\geq (1-2\varepsilon^2)\frac{N}{4},\end{aligned}$$

concluding the proof for statement (3). □

Lemma 4.3.8. *Let $G_{\mathsf{YES}} \sim \mathcal{D}_{\mathsf{YES}}$ and $G_{\mathsf{NO}} \sim \mathcal{D}_{\mathsf{NO}}$. Then, with high probability,*

- $\mu(G_{\mathsf{YES}}) \geq \left(2/\varepsilon + 1 - 4\varepsilon^2\right)\frac{N}{4}$,
- $\mu(G_{\mathsf{NO}}) \leq (2/\varepsilon + 4\varepsilon)\frac{N}{4}$.

Proof. Consider the graph G_{YES}. For each $i \in [1/\varepsilon - 1]$ and $j \in \{1,2\}$, by Claim 4.3.7, we have a matching between A^j_i and B^j_{i+1} that matches $(1-\varepsilon^3)N/4$ vertices of each part. Also, for each $j \in \{1,2\}$, we have a matching between S^j and B^j_1 that matches $(1-\varepsilon^3)N/4$ vertices of each part. Moreover, there exists a matching between $A^j_{1/\varepsilon}$ and $A^{3-j}_{1/\varepsilon}$ that matches $(1-2\varepsilon^2)N$ vertices of each part. Since the vertex sets are disjoint, by taking the edges of all these matchings, we have

$$\begin{aligned}\mu(G_{\mathsf{YES}}) &\geq 2\left(\frac{1}{\varepsilon}-1\right)(1-\varepsilon^3)\frac{N}{4} + \frac{N}{2} + (1-2\varepsilon^2)\frac{N}{4} \\ &\geq \left(\frac{2}{\varepsilon} + 1 - 3\varepsilon^2\right)\frac{N}{4}.\end{aligned}$$

Now consider G_{NO}. First, we show that $\mu(G_{\mathsf{NO}}[V \setminus \bigcup_{i=1}^{1/\varepsilon} D_i]) \leq N/(2\varepsilon)$. To see this, note that $G_{\mathsf{NO}}[V \setminus \bigcup_{i=1}^{1/\varepsilon} D_i]$ is a bipartite graph which implies that the size of the vertex cover of this graph is equal to the size of the maximum matching by König's Theorem (Proposition 2.2.1). Since there is no edge in the induced graph $G[\bigcup_{i=1,j\in\{1,2\}}^{1/\varepsilon} A^j_i \cup \{S^1, S^2\}]$, we take $\bigcup_{i=1,j\in\{1,2\}}^{i\leq 1/\varepsilon} B^j_i$ as the vertex cover of this graph. Furthermore, since $|\bigcup_{i=1}^{1/\varepsilon} D_i| = \varepsilon N$, the number of maximum matching edges that have at least one endpoint in $|\bigcup_{i=1}^{1/\varepsilon} D_i|$ is at most εN. Thus, we have

$$\begin{aligned}\mu(G_{\mathsf{NO}}) &\leq \mu\left(G_{\mathsf{NO}}\left[V \setminus \bigcup_{i=1}^{1/\varepsilon} D_i\right]\right) + \varepsilon N \\ &\leq \frac{N}{2\varepsilon} + \varepsilon N \\ &= \left(\frac{2}{\varepsilon} + 4\varepsilon\right)\frac{N}{4}.\end{aligned}$$

□

Corollary 4.3.9. *Let $\varepsilon < 0.07$. Any algorithm that estimates the size of maximum matching of a graph G that is drawn from input distribution within a factor of $(1, \varepsilon n/7)$ with probability at least 0.51, must be able to distinguish whether G belongs to $\mathcal{D}_{\mathsf{YES}}$ or $\mathcal{D}_{\mathsf{NO}}$.*

Proof. Note that we have

$$\mu(G_{\mathsf{YES}}) - \mu(G_{\mathsf{NO}}) \geq \left(1 - 4\varepsilon - 4\varepsilon^2\right) \frac{N}{4}.$$

Moreover, since $\varepsilon < 0.07$,

$$\left(1 - 4\varepsilon - 4\varepsilon^2\right) \frac{N}{4} > \frac{N}{6}.$$

Combining with the fact that $N > 6/7 \cdot (\varepsilon n)$, we obtain the claimed bound. □

4.3.3 A Reduction to a Label Guessing Game on Trees

In this section, we prove that any algorithm that makes $o(d^{1/\varepsilon})$ queries, cannot discover any cycle and only sees a rooted forest with high probability. This effectively reduces the problem to the label guessing game on trees that we outlined in Section 4.3.1. The following lemma formalizes the main result of this section.

Lemma 4.3.10. *Let $\mathcal{A}$ be any algorithm that makes at most $o(d^{1/\varepsilon})$ queries. Let F_0 be the empty graph before the algorithm makes any queries, and for $t > 0$, let F_t be the subgraph that $\mathcal{A}$ discovers after t queries. The following property holds throughout the execution of $\mathcal{A}$ with probability $1 - o(1)$: Suppose that the t-th query is made to the adjacency list of vertex u and edge (u, v) is returned. Then, vertex v is a singleton vertex in F_{t-1}.*

Remark 5. *Lemma 4.3.10 implies that the discovered forest can be thought of as a rooted forest. In other words, if edge (u, v) is discovered by the algorithm at step t and v is the singleton vertex, then v is the leaf of F_t.*

The main technical part to prove Lemma 4.3.10 is to show that at any time during the execution of the algorithm, for any pair of vertices (u, v) that $\mathcal{A}$ has not discovered an edge yet, the probability of having an edge (u, v) is at most $O(d/n)$. To see this, note that if u and v belong to two blocks in the construction that there is no edge between them, then the probability of having an edge between them is zero. Now if they belong to two blocks that we put edges between them, then since we put almost regular graphs with a degree of at most $O(d)$ between any two blocks, the probability of having that edge is $O(d/n)$. This is not a formal argument and in order to formalize this intuition, we use a coupling argument.

Lemma 4.3.11. *Let (u, v) be a pair of vertices that the algorithm has not discovered an edge between them. Then, the probability of having the edge (u, v) in G is $O(d/n)$.*

Before proving Lemma 4.3.11, first we show how we can complete the proof of Lemma 4.3.10 using Lemma 4.3.11.

Proof of Lemma 4.3.10. The proof consists of two parts. First, we show that during the execution of the algorithm at any time t, if u and v are two non-singleton vertices, then there is no edge between u and v. We use induction on t to prove this claim. For $t = 0$ this claim clearly holds. At time $t > 0$, suppose that $\mathcal{A}$ finds an edge (u, v) such that v is a singleton in F_{t-1} (similarly, u can be a singleton vertex). Now we need to show that v does not have any edge to non-singleton vertices in F_{t-1} except v. Note that the probability of having an edge between v and any of non-singleton vertices in F_{t-1} is $O(d/n)$. Since $\mathcal{A}$ make

at most $o(d^{1/\varepsilon})$ queries, there are at most $o(d^{1/\varepsilon})$ non-singleton vertices in F_{t-1}. Thus, by union bound, the probability of having an edge between v and non-singleton vertices of F_{t-1} is $o(d^{1/\varepsilon}) \cdot O(d/n) = o(d^{1/\varepsilon+1}/n)$. Moreover, the induction has $o(d^{1/\varepsilon})$ steps since the algorithm makes at most $o(d^{1/\varepsilon})$ queries. Therefore, the probability of failure over all steps is at most $o(d^{1/\varepsilon}) \cdot o(d^{1/\varepsilon+1}/n) = o(1)$ because $d = n^{\varepsilon/3}$.

Second, we show that if we query the adjacency list of a singleton vertex u and the algorithm discovers edge (u, v), then v is also a singleton vertex. Fix a singleton vertex u. By Lemma 4.3.11, the probability of having an edge between u and each of the non-singleton vertices in the forest is $O(d/n)$. Hence, the expected number of edges between u and non-singleton vertices is at most $o(d^{1/\varepsilon+1}/n)$ since there are at most $o(d^{1/\varepsilon})$ non-singleton vertices. Furthermore, u has $\Omega(d)$ neighbors according to the construction and the adjacency list of u is randomly permuted which implies that the probability of the first neighbor in the adjacency list to be non-singleton is $o(d^{1/\varepsilon}/n)$. Since the algorithm makes at most $o(d^{1/\varepsilon})$ queries, the probability of seeing an edge between a singleton vertex and non-singleton vertex when the algorithm queries the singleton vertex's adjacency list is at most $o(d^{1/\varepsilon}) \cdot o(d^{1/\varepsilon}/n) = o(1)$ by union bound, which completes the proof. □

Proof of Lemma 4.3.11

Suppose that $u \in X$ and $v \in Y$, where X and Y show the subset in the construction that u and v belong to. If there is no edge in the construction between two subsets X and Y, then the probability of having edge (u, v) is zero. Now we consider two possible scenarios for the types X and Y: 1) one of X or Y is of type S^1 or S^2, 2) none of X or Y is of type S^1 or S^2.

In the first case, without loss of generality assume that $X \in S^1$ and $Y \in B_1^1$. Let $v \in B_1^1$. According to the binomial distribution of neighbors of v, the expected number of S^1 neighbors of v is $\log^4 N$. Thus, using the Chernoff bound, the total number of edges between B_1^1 and S_1^1 is not larger than $\frac{N}{3}\log^4 N$ with high probability, which implies that there are at least $\frac{N}{6}\log^4 N$ vertices of S_1^1 that have a degree equal to zero. Now let $\mathcal{G}$ be the set of all graphs in the input distribution that have edge (u, v), and $\hat{\mathcal{G}}$ be the set of all graphs in the input distribution that does not have edge (u, v). For a graph in $\mathcal{G}$, we can remove the edge (u, v) and add edge (w, v) for a vertex $w \in S_1^1$ that has degree zero. Since there exists $O(n \log^4 n)$ such w, we can couple the initial graph to $\Omega(n \log^4 n)$ graphs in $\hat{\mathcal{G}}$. On the other hand, each graph of $\hat{\mathcal{G}}$ is coupled to at most $O(\log^4 n)$ graphs in $\mathcal{G}$ since the degree of v is at most $O(\log^4 n)$. Hence, we have $|\mathcal{G}|/|\hat{\mathcal{G}}| \leq O(1/n)$, which concludes the proof for the first case since the number of graphs in the input distribution that have the edge (u, v) is $O(1/n)$ fraction of graphs that does not have the edge (u, v).

For the second case, we use a more complicated coupling argument. Suppose that the expected degree of a vertex in X in the subgraph of $G[X, Y]$ is d_1 and the expected degree of a vertex in Y is d_2 in $G[X, Y]$. By Lemma 4.3.4 and using Chernoff bound, the degree of all vertices X is in the range $d_1 \pm d_1/2$ in subgraph $G[X, Y]$. Similarly, the degree of all vertices Y is in the range $d_2 \pm d_2/2$ in subgraph $G[X, Y]$. We define $\mathcal{G}$ and $\hat{\mathcal{G}}$ similar to the previous case. The key idea for this case is that if edge (u, v) exists in a graph, we can find many edges (x, y) such that $x \in X$, $y \in Y$, edge (x, y) is not discovered by the algorithm, and there exist exactly two edges (u, v) and (x, y) in $G[\{u, v, x, y\}]$. Then, by removing edges $\{(u, v), (x, y)\}$ and adding edges $\{(u, y), (x, v)\}$ we can obtain a graph that does not have edge (u, v), its degree sequence does not change, and satisfy all properties of input distribution (if the initial graph is in $\mathcal{D}_{\mathsf{YES}}$, the final graph is also in $\mathcal{D}_{\mathsf{YES}}$. The same statement hold for $\mathcal{D}_{\mathsf{NO}}$).

Suppose that H is a graph that has edge (u, v). Since u has $\Theta(d_1)$ neighbors in Y, there exist $\Theta(n - d_1)$

non-adjacent vertices of Y to u. Let C_Y denote the set of non-adjacent vertices of Y to u. Each vertex in C_Y has at least $\Theta(d_2)$ neighbors in X. Therefore, there $\Theta((n-d_1)d_2)$ candidate vertices for x. However, some of these edges from y are already discovered by the algorithm. Note that the number of discovered edges is $o(n)$ at any point during the course of the algorithm because of the choice of d in the construction. So by removing these $o(n)$ edges, there are still $\Theta((n-d_1)d_2)$ candidate for x. Furthermore, at most $\Theta(d_2^2)$ of the edges from a vertex of C_Y to candidates for x, have an incident edge such that one of their endpoints of the incident edge is v. Therefore, there are at least $\Theta((n-d_1)d_2 - d_2^2)$ induced subgraphs of four vertices with the required properties.

Note that according to the construction, either $d_1 = \Theta(d)$ and $d_2 = \Theta(d)$, or $d_1 = \Theta(\log^4 n)$ and $d_2 = \Theta(\log^4 n)$ which implies that $\Theta((n-d_1)d_2 - d_2^2) = \Theta(nd_2)$. We couple subgraph H to all $\Theta(nd_2)$ graphs that are obtained by removing edges $\{(u,v),(x,y)\}$ and adding edges $\{(u,y),(x,v)\}$. On the other hand, each graph of $\hat{\mathcal{G}}$ is coupled with $\Theta(d_1 d_2)$ graphs in $\mathcal{G}$ since degree of u is $\Theta(d_1)$ and degree of v is $\Theta(d_2)$. Therefore, we have $|\mathcal{G}|/|\hat{\mathcal{G}}| \leq O(d_1/n)$ which completes the proof.

The Label Guessing Game on Trees

Claim 4.3.12. *Any algorithm $\mathcal{A}$ that makes at most $Q = o(d^{1/\varepsilon})$ adjacency list queries, does not discover any broken vertex with high probability.*

Proof. By Observation 4.3.6, there are at most $O(\sqrt{nd}\log n)$ broken vertices. Therefore, if we choose a random vertex, the probability of being a broken vertex is at most $O(\sqrt{d}\log n/\sqrt{n})$. Also, by the same 2-switch technique as the proof of Lemma 4.3.11, we can show that when we query a neighbor of a vertex, the probability of being broken is almost the same as when we choose a vertex uniformly at random. Since the algorithm makes at most $o(d^{1/\varepsilon})$ queries, the total probability of finding a broken vertex is $O(d^{1/\varepsilon}\sqrt{d}\log n/\sqrt{n}) = o(1)$. □

Corollary 4.3.13. *Let us condition on the high probability event of Claim 4.3.12 that none of the broken vertices has been queried by the algorithm. Suppose that the algorithm makes a query to the adjacency list of vertex v that is in subset X and u is the answer to the query. Then, the subset Y that u belongs to is determined by the binomial distribution that is defined in the construction.*

Proof. Fix a vertex v. Note that the type of neighbor of v that is connected to v by a non-broken edge is determined by a binomial random variable that is defined in the construction. □

By conditioning on the high probability event of Lemma 4.3.10 that the queried edges make a rooted forest and the properties of the input distribution, by Corollary 4.3.13, we can assume that we are in a tree model where each vertex has a label according to the subset that it belongs to and the distribution coming from the following transition probabilities. This is exactly the label guessing game outlined in the technical overview of Section 4.3.1 (see Figure 4.5).

Labels of vertices: We use S as the label of vertices in subset S^1 and S^2, A_i for vertices in subset A_i^1 and A_i^2, B_i for vertices in subset B_i^1 and B_i^2, and D_i for vertices in subset D_i for $i \in [1/\varepsilon]$.

Transition probabilities: Suppose that we condition on the high probability event that the algorithm does not query any broken vertex. Let (u, v) be an edge in the forest that is queried by the algorithm and u is the parent of v. Then, if u has label X and v has label Y for $X, Y \in \bigcup_{i=1}^{1/\varepsilon}\{A_i, B_i, D_i\} \cup \{S\}$, then the transition probabilities from label X to label Y is according to the binomial distribution for neighbors of X in Section 4.3.2.

4.3.4 Indistinguishability of the Label of the Root

In this section, we show that if the result of the queried edges is a rooted tree of size $Q = o(d^{1/(2\varepsilon)})$, then the algorithm can distinguish the label of the root with probability at most $\widetilde{O}(Q^2/d^{1/\varepsilon})$ if the label of the root is in $\{A_{1/\varepsilon}, B_{1/\varepsilon}, D_{1/\varepsilon}\}$. Our proof consists of two parts. First, we show that when we have $o(d^{1/(2\varepsilon)})$ queries in the tree, with probability at least $1 - \widetilde{O}(Q^2/d^{1/\varepsilon})$, all paths that start from the root and reach an S vertex must contain a *mixer vertex* that we define later in the section. We define mixer vertices such that if a path contains such a vertex, then the algorithm does not learn anything about the label of the root from this path. For this, we prove a stronger claim that starting from the root of the tree, there is no path that contains more than $1/\varepsilon - 1$ special edges before crossing a mixer vertex.

Second, conditioning on the above event, we prove that the algorithm will see the same tree if the root is in $\{A_{1/\varepsilon}, B_{1/\varepsilon}, D_{1/\varepsilon}\}$, which implies that the algorithm cannot distinguish the label of the root with probability at least $1 - \widetilde{O}(Q^2/d^{1/\varepsilon})$.

Definition 4.3.14 (Special Edges)**.** *We call an edge (u, v) special, if one of the following holds:*

- *$u \in B_i$ and $v \in A_{i-1}$, or $u \in A_{i-1}$ and $v \in B_i$ for $1 < i \leq 1/\varepsilon$,*
- *$u \in S$ and $v \in B_1$, or $u \in B_1$ and $v \in S$,*
- *Let $p = (1 - 2\varepsilon + 2i\varepsilon^2 - 5\varepsilon^2/2 + 3\varepsilon^4)d + \log^4 N$. For each vertex in D_i, each of its neighbors to D_i has a probability of $\log^4 N/p$ to be special (in other words, we can assume that there is $\log^4 N$ regular graph of special edges in each D_i),*
- *(u, v) is among edges that only exists in exactly one of $\mathcal{D}_{\mathsf{YES}}$ or $\mathcal{D}_{\mathsf{NO}}$.*

Definition 4.3.15 (Mixer Vertices)**.** *Let T be a rooted tree and u be its root. Suppose that we are given that $u \in \{A_{1/\varepsilon}, B_{1/\varepsilon}, D_{1/\varepsilon}\}$. Let v be a vertex in T and suppose that there are k special edges on the path between u and v. If $k < 1/\varepsilon - 1$, we say v is a mixer vertex if and only if $v \in \bigcup_{i=1}^{1/\varepsilon-k-1} D_j$.*

The following observation is directly implied by the Definition 4.3.14, Definition 4.3.15, and the construction of the input distribution.

Observation 4.3.16. *Let T be a rooted tree that is queried by the algorithm and u be its root where $u \in \{A_{1/\varepsilon}, B_{1/\varepsilon}, D_{1/\varepsilon}\}$. If there exists a path from u to an S vertex that does not contain a mixer vertex, then it contains at least $1/\varepsilon - 1$ special edges.*

The intuition behind defining mixer vertex this way is that if the root of the tree is a level $1/\varepsilon$ vertex and on a path that the algorithm queries, if there are k special edges, then all vertices with a level of at least $1/\varepsilon - k$, have the same probability of having neighbors among vertices of $\bigcup_{j=1}^{1/\varepsilon-k-1} D_j$ which implies that if the path crosses one of those mixer vertices, then the algorithm cannot distinguish the label of the root using that path.

Lemma 4.3.17. *Let $\mathcal{A}$ be any algorithm that makes at most $o(d^{1/\varepsilon})$ queries and T be one of the rooted trees queried by the algorithm. Moreover, assume that the root of the tree is a vertex with level $1/\varepsilon$. Then, with probability at least $1 - \widetilde{O}(|V(T)|/d^{1/\varepsilon-1})$, all paths between the root and a vertex that does not contain a mixer vertex, have at most $1/\varepsilon - 2$ special edges on it.*

Proof. First, we prove that each path that $\mathcal{A}$ finds to a vertex v that contains $1/\varepsilon - 1$ special edges on it, has a probability of $\widetilde{O}(1/d^{1/\varepsilon-1})$ of not having any mixer vertex on it. For a mixer vertex v such that $v \in D_j$, we use j to show the *index* of the mixer vertex. Assume that we have an oracle that each time the algorithm finds a path with $1/\varepsilon - 1$ special edges, it either returns the path that does not contain a mixer vertex or returns the mixer vertex on the path that has the lowest index among all mixers on the path.

Consider a path from the root to an S vertex and a time t that the algorithm has not queried the whole path yet. Suppose that the algorithm has found at most $1/\varepsilon - 2$ special edges until time t. This implies that this path does not reach level 1 or an S vertex yet according to the construction and Observation 4.3.16. By the transition probability of the tree model, the probability of querying a D_1 vertex from a vertex of level 2 or larger is constant, however, the probability of querying a special edge is $\widetilde{O}(1/d)$ which implies that with probability $\widetilde{O}(1/d)$ the path crosses the $(1/\varepsilon - 1)$-th special edge before crossing a mixer vertex of level 1. Thus, among all paths that cross at most $1/\varepsilon - 2$ special edges and are going to reach the next special edge, only $\widetilde{O}(1/d)$ fraction of them do not pass through a mixer vertex of level 1. Therefore, $\widetilde{O}(1/d)$ of all paths that have $1/\varepsilon - 1$ special edges, do not contain a mixer vertex of level 1.

Now consider all paths that do not contain a mixer vertex of level 1. With the same argument, for each of these paths, the probability of crossing $(1/\varepsilon - 2)$-th special edge before crossing a mixer vertex of level 2 is $\widetilde{O}(1/d)$. Therefore, since the oracle only reveals the mixer vertex with the lowest index, then the probability of having a path with $1/\varepsilon - 1$ special edges that do not contain a mixer vertex is $\widetilde{O}(1/d^{1/\varepsilon-1})$. Since there are at most $|V(T)|$ paths from the root, we obtain the claimed bound. □

Corollary 4.3.18. *Let $\mathcal{A}$ be any algorithm that makes at most $o(d^{1/\varepsilon-1})$ queries and T be one of the rooted trees queried by the algorithm. Moreover, assume that the root of the tree is a vertex with level $1/\varepsilon$. Then, with probability at least $1 - \widetilde{O}(|V(T)|/d^{1/\varepsilon-1})$, all paths between root and S vertices in the tree contain a mixer vertex.*

Proof. Note that if there exists a path between the root and an S vertex that does not contain a mixer vertex, it must contain at least $1/\varepsilon - 1$ special edges. To see this, the only way that a vertex from level i can reach level $i - 1$ is to either cross a mixer vertex or a special edge. Combining with Lemma 4.3.17 we get the claimed bound. □

Corollary 4.3.19. *Let $\mathcal{T}$ be a set of root trees such that the roots of all its trees belong to level $1/\varepsilon$. Also, let $r = \sum_{T\in\mathcal{T}} |V(T)|$, and assume that we have $r = o(d^{1/\varepsilon-1})$. Then, with probability at least $1 - \widetilde{O}(r/d^{1/\varepsilon-1})$, all paths between the roots of trees and a vertex that in the same tree that does not contain a mixer vertex, have at most $1/\varepsilon - 2$ special edges on it.*

Proof. Let $T_1, T_2, \ldots, T_k$ be all trees in F. By Lemma 4.3.17, for each tree T_i, the probability of having such a path is at most $\widetilde{O}(|V(T_i)|/d^{1/\varepsilon-1})$. Hence, using union bound, the probability of having no path with more $1/\varepsilon - 2$ special edges without any mixer vertex is at most $\widetilde{O}(r/d^{1/\varepsilon-1})$ which completes the proof. □

Lemma 4.3.20. *Let T be a tree that is queried by an algorithm $\mathcal{A}$ on a graph that is drawn from input distribution, where the root belongs to level $1/\varepsilon$. Also, suppose that on each path from the root of the tree to a vertex in the tree, if there are at least $1/\varepsilon - 1$ special edges, then there exists at least one mixer vertex on the path. Then, the probability of seeing the same tree is equal for all possible roots in $\{A_{1/\varepsilon}, B_{1/\varepsilon}, D_{1/\varepsilon}\}$ up to $(1 + o(d^{1/(2\varepsilon)+1}/n))^{|T|}$ multiplicative factor.*

Proof. The proof is involved and we begin by identifying some properties of input distribution that are useful in the proof. Let $G = (V, E)$ be the input graph that is drawn from the input distribution. Note that for all vertices in the graph except S vertices, when $\mathcal{A}$ queries a new edge, the probability of the edge being special is the same.

Observation 4.3.21. *Let u be an arbitrary vertex in a graph that is drawn from input distribution. Also, let (u, v) be a new queried edge by $\mathcal{A}$. Then, the probability of (u, v) being a special edge is $\log^4 N/d'$.*

Proof. The proof follows by the transition probability of the tree model and the way we defined special edges in Definition 4.3.14. □

Let $L_i = \{A_i, B_i, D_i\}$ for $i \in [1/\varepsilon]$. Also, let E_S be the set of all special edges defined in Definition 4.3.14. Let $G_i = G[\bigcup_{j=i}^{1/\varepsilon} L_j]$. Let u and v be two different vertices in G_i. One important property of our input distribution is that if we query a neighbor of u and v and the queried edge is not a special edge, then the probability that the queried neighbor is a vertex in G_i is equal for both u and v.

Claim 4.3.22. *Let $u, v \in V(G_i)$ for some $i \in [1/\varepsilon]$. Also, let (u, u') and (v, v') be two edges that are queried by $\mathcal{A}$ and both are not special edges. Then, $\Pr[u' \in V(G_i)] = \Pr[v' \in V(G_i)]$.*

Proof. By the construction of distribution, there is no edge between $\{u, v\}$ and $\bigcup_{j=1}^{i-1} A_j \cup B_j$. Furthermore, if $\mathcal{A}$ queries an edge of a vertex in $V(G_i)$, with probability $\varepsilon^4 d/d'$ the neighbor is in D_j for $j < i$. Thus, the probability of the neighbor being in $\bigcup_{j=1}^{i-1} D_j$ is $(i-1)\varepsilon^4 d/d'$. Therefore, we have $\Pr[u' \in V(G_i)] = \Pr[v' \in V(G_i)]$. □

Now we are ready to complete the proof. Let $\ell_1, \ell_2 \in \{A_{1/\varepsilon}, B_{1/\varepsilon}, D_{1/\varepsilon}\}$ be two different labels for the root of the tree T. The proof is based on a one-to-one coupling argument that for each tree that is queried by the algorithm if ℓ_1 is the label of the tree, $\mathcal{A}$ will see the same tree with an equal probability if it starts from label ℓ_2.

For a vertex u in tree T such that there exists no mixer vertex on its path to the root, we define the notion of *progress of a vertex*, i.e. p_u, which shows the number of special edges on the path of root to u. Because of the assumption in the lemma statement, we have $0 \leq p_u < 1/\varepsilon - 1$ for all $u \in T$.

Observation 4.3.23. *Let T be a tree that is queried by $\mathcal{A}$ and its root is in level $1/\varepsilon$. Also, let u be a vertex such that there is no mixer vertex on the path of u to the root. Then, we have $u \in V(G_{1/\varepsilon - p_u})$.*

Proof. Note that according to the construction of the input distribution, if there is no mixer vertex on the path, each special edge can be used for going at most one level down in the input graph. Therefore, if there are p_u special edges on the path, then $u \in V(G_{1/\varepsilon - p_u})$. □

Observation 4.3.24. *Let u be a vertex in T such that there is no mixer vertex on the path of u to the root. Suppose that $\mathcal{A}$ queries the adjacency list of u and let ℓ be the label of the neighbor. Then, for each $j \in [1/\varepsilon - p_u - 1]$ it holds that $\Pr[\ell = D_j] = \varepsilon^4 d/d'$.*

Proof. By Observation 4.3.23, we have $u \in V(G_{1/\varepsilon - p_u})$. According to the transition probabilities of the tree model, for a vertex in $G_{1/\varepsilon-p_u}$ the probability of seeing a neighbor with label D_j is $\varepsilon^4 d/d'$ for $j \in [1/\varepsilon - p_u - 1]$. □

Let $\mathcal{L}_1$ be a labeling for T that $\mathcal{A}$ sees when it starts from a root with label ℓ_1. Let $e = (u, v)$ be an edge in T such that u is the parent of v. There are three possible types for e if there is no mixer vertex on a path between the root and u: 1) the edge is a special edge, 2) v is a mixer vertex, 3) e is an edge in $G_{1/\varepsilon-p_u} \setminus E_S$. We give a labeling $\mathcal{L}_2$ for the same tree where the root has label ℓ_2 and all S vertices have label S and the probability that $\mathcal{A}$ sees this labeling is equal to the probability of seeing $\mathcal{L}_1$. We maintain the invariant that the progress of each vertex is the same in both labeling $\mathcal{L}_1$ and $\mathcal{L}_2$.

Now we start to process edges one by one according to the ordering that $\mathcal{A}$ makes queries. If the queried edge $e = (u, v)$ is of type (2), suppose that the label of v is D_j for some $j \in [1/\varepsilon - p_u - 1]$. We assign the same label D_j to v in $\mathcal{L}_2$. By Observation 4.3.24, because of the invariant that u has the same progress in both labelings, the probability of seeing label D_j is the same for both labelings. Moreover, for the subtree below v, we assume that all the labels are the same since the label of v is the same at this point in both $\mathcal{L}_1$ and $\mathcal{L}_2$. Also, the invariant still holds.

If the queried edge $e = (u, v)$ is of type (1), i.e. is a special edge, we assume that in labeling $\mathcal{L}_2$, the edge is also a special edge and determine the label accordingly. Note that v cannot have label S in $\mathcal{L}_1$ since in this case there is no mixer vertex on the path to v which is a contradiction Corollary 4.3.18. By Observation 4.3.21, the probability of querying a special edge is the same in both labelings. Furthermore, the invariant still holds since $p_v = p_u + 1$ in this case and the progress of vertex u is the same in both labelings.

Finally, if the queried edge $e = (u, v)$ is of type (3), we assume that v has a label that is drawn from crossing an edge of $G_{1/\varepsilon-p_u} \setminus E_S$. By Claim 4.3.22, the probability of crossing the edge of type (3) is the same for both labelings. Also, the invariant still holds since we did not add a special edge, which completes our coupling argument. It is also important to note that by the proof of Lemma 4.3.10, the probability that the vertex with a new label is among non-singleton vertices is at most $o(d^{1/(2\varepsilon)+1}/n)$. Since the total number of steps is at most $|T|$, the probability that the new labeling is also a forest is almost equal within $(1 + o(d^{1/(2\varepsilon)+1}/n))^{|T|}$ multiplicative factor. □

4.3.5 Indistinguishability of Crucial Edges

Since the algorithm can make $o(d^{1/(2\varepsilon)})$ queries, it might discover several edges that only appear in $\mathcal{D}_{\mathsf{YES}}$ because $\widetilde{\Theta}(1/d)$ fraction of total edges is specific to the $\mathcal{D}_{\mathsf{YES}}$. However, we show that no algorithm can distinguish those edges with high probability. More specifically, we prove that the probability of seeing the same forest in $\mathcal{D}_{\mathsf{NO}}$ is the same as $\mathcal{D}_{\mathsf{YES}}$ up to a $1 + o(1)$ multiplicative factor.

Let $E_Y = \{(u, v) | (u, v) \in E, u \in A^1_{1/\varepsilon}, v \in A^2_{1/\varepsilon}\}$, and E^Q_Y be the subset of E_Y that the algorithm discovers. Moreover, let $V^Q_Y = \{v \mid (u, v) \in E_Y\}$ where u is the parent of v in the queried rooted forest by algorithm $\mathcal{A}$. Also, assume that we remove all vertices of V^Q_Y that have at least one ancestor in the forest which is in V^Q_Y.

Claim 4.3.25. *Let F be a forest that is queried by an algorithm $\mathcal{A}$ using at most $Q = o(d^{1/(2\varepsilon)})$ queries. Then, with probability at least $1-\widetilde{O}(Q^2/d^{1/\varepsilon})$, all paths between the vertices of V_Y^Q and a vertex in its subtree that have more than $1/\varepsilon - 2$ special edges, contain a mixer vertex.*

Proof. Since the algorithm makes at most $o(d^{1/(2\varepsilon)})$ queries, then we have $|V_Y^Q| \leq Q \leq o(d^{1/(2\varepsilon)})$. Moreover, for each vertex in V_Y^Q, its subtree contains at most $o(d^{1/(2\varepsilon)})$ vertices. Hence, the total number of vertices in all subtrees with root in V_Y^Q is at most $Q^2 = o(d^{1/\varepsilon})$. Therefore, by Corollary 4.3.19, with probability at least $1 - \widetilde{O}(Q^2/d^{1/\varepsilon})$, there exists a mixer vertex on all paths between vertices of V_Y^Q and vertices of its subtree before crossing $1/\varepsilon - 1$ special edges. □

We define a *bad event* to be the event that $\mathcal{A}$ finds a path between a vertex in V_Y^Q and a vertex in its subtree that has more than $1/\varepsilon-2$ special edges without any mixer vertex. By Claim 4.3.25, since we assume that the algorithm makes $o(d^{1/(2\varepsilon)})$ queries, the bad event happens with probability $o(1)$.

Lemma 4.3.26. *Let us condition on having no bad event as we defined above. Let $\mathcal{A}$ be an algorithm that makes at most $Q = o(d^{1/(2\varepsilon)})$ queries and F be a rooted forest that is discovered by $\mathcal{A}$ on a graph that is drawn from $\mathcal{D}_{\mathsf{YES}}$. Then, the probability of querying the same forest in the graph that is drawn from $\mathcal{D}_{\mathsf{NO}}$ is equal to $\mathcal{D}_{\mathsf{YES}}$.*

Proof. We say an edge is *crucial* in $\mathcal{D}_{\mathsf{YES}}$, if the edge is in induced subgraph $G[A^1_{1/\varepsilon}, A^2_{1/\varepsilon}]$. In other words, in $\mathcal{D}_{\mathsf{YES}}$, edges of E_Y that we defined in this section are crucial edges. We extend the definition of crucial edges to have all edges that are specific to the $\mathcal{D}_{\mathsf{YES}}$. Also, for $\mathcal{D}_{\mathsf{NO}}$, we define crucial edges to be the set of edges that only exists in $\mathcal{D}_{\mathsf{NO}}$. Note that ignoring the crucial edges, if we query an edge, the probability of the neighbor is the same in both distributions.

Let the height of an edge in the forest be the distance of its closest endpoint to the root of the tree that belongs to. We prove this lemma using coupling between $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$. We iterate over the height of the tree in decreasing order and inductively we show that we can switch from $\mathcal{D}_{\mathsf{YES}}$ to $\mathcal{D}_{\mathsf{NO}}$. Consider height i in all trees. If the edge is not crucial, both distributions will sample similarly according to the construction. Since crucial edges are between vertices of level $1/\varepsilon$, since we condition on not having a bad event for vertices of V_Y^Q, the subtree below the crucial edges are the same regardless of their labels up to a factor of $(1+o(d^{1/(2\varepsilon)+1}/n))^{|T|}$ by Lemma 4.3.20 if $|T|$ shows the size of the subtree. Since the total number of vertices in all these subtrees are at most $o(d^{1/\varepsilon})$, and $d = n^{\varepsilon/3}$, the probability of discovering the same forest in both distributions is equal up to a $1 + o(1)$ multiplicative factor. □

Now we are ready to finish the proof of Theorem 4.3.1.

Proof of Theorem 4.3.1. By Claim 4.3.25, the probability of having a bad event is $o(1)$. If there is no bad event in the forest that the algorithm queries, the algorithm will discover the same forest with almost equal probability in both $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$, by Lemma 4.3.26 with $1+o(1)$ multiplicative factor. Therefore, combining with Corollary 4.3.9, any algorithm that computes a $(1, \varepsilon n/7)$-approximate maximum matching, must make at least $d^{1/(2\varepsilon)}$ queries. Also, by Remark 4, we can assume that each D_i consists of two parts where one of them is connected to label B vertices, the other one connected to label A, and there is no edge inside the induced subgraph of each of the two copies which implies that the graph is bipartite. Choosing $\varepsilon' = \varepsilon/7$ and combining it with the fact that $\Delta < 2d$ concludes the proof. □

Proof of Theorem 4.3.2. Suppose for the sake of contradiction that there exists an LCA that computes $(1, \varepsilon n)$-approximate maximum matching of G with running time of $\Delta^{o(1/\varepsilon)}$. We sample t random vertices in the graph, and run this LCA on the selected vertices. Let t' be the number of samples that the LCA returns a match for. We return $(t'/t) \cdot n/2$ as our estimate for the size of maximum matching. A simple Chernoff bound (see e.g. [34, 176]) shows that setting $t = \Theta(1/\varepsilon^2)$ suffices for an estimation that is accurate up to an additive error of εn. From this, we get that there must exist an algorithm that runs in $(1/\varepsilon^2) \cdot \Delta^{o(1/\varepsilon)}$ time and $(1, 2\varepsilon n)$ approximates the size of maximum matching. Since we ruled out the existence of such an algorithm in Theorem 4.3.1, there exists no such LCA. □

4.4 Approximating Maximum Matching Requires Almost Quadratic Time

In this section, we prove the following theorem.

Theorem 4.4.1 (Main Result). *For any $\delta > 0$ there is an $\varepsilon = \varepsilon(\delta) > 0$ such that any (randomized) algorithm that (with probability at least $2/3$) estimates the size of maximum matching of an n-vertex graph up to an additive error of εn has to make $\Omega(n^{2-\delta})$ adjacency list queries to the graph.*

Furthermore, this holds even if the graph is bipartite and is promised to either have a perfect matching or a matching that leaves $\Theta(\varepsilon n)$ vertices unmatched.

The prior lower bound analysis of Section 4.2 and Section 4.3 work in a certain *tree model* and rely crucially on the fact that the algorithm cannot discover any cycles. It turns out that this assumption completely breaks when the algorithm is allowed to make $\omega(n\sqrt{n})$ queries. This is the main conceptual and technical obstacle that our lower bound of Theorem 4.4.1 overcomes. In Section 4.4.1, we elaborate more on the cycle discovery barrier, its importance in the literature of sublinear time algorithms and lower bounds, and our techniques to bypass it for approximating maximum matchings.

4.4.1 Technical Overview

Background on Existing Lower Bounds in Previous Sections

High degree dummy vertices. The first basic idea for proving query complexity lower bounds in the adjacency list model, also common in earlier lower bounds [159, 43], is to add εn *dummy* vertices and make them adjacent to the rest of vertices. The dummy vertices do not contribute significantly to the maximum matching as there are few of them, but increase the number of edges to $\Omega(\varepsilon n^2)$, effectively congesting the adjacency lists with redundant calls. We henceforth refer to the non-dummy part of the graph as the *core*. That is, non-dummy vertices are *core vertices* and edges between core vertices are *core edges*.

Parnas and Ron [159] gave a linear lower bound of $\Omega(n)$ for any constant approximate algorithm by taking the core to be (essentially) either a random perfect matching or the empty graph. Intuitively, because of the dummy vertices, it takes the algorithm $\Omega(n)$ adjacency list queries to even hit one edge of the matching edges in the core. This argument breaks when the goal is to prove super-linear lower bounds. Note that if the algorithm is able to make nk queries, then it can random sample k vertices and query all of their

adjacency lists, therefore at least $\Omega(k)$ edges of the maximum matching of size $\Theta(n)$ will be revealed to the algorithm.

Camouflage the good matching. As discussed above it is impossible to hide the maximum matching edges in the sense that some of them will be revealed to the algorithm. The approach pioneered by the work of Behnezhad, Roghani, and Rubinstein [42] to overcome this challenge is to introduce a special construction which *camouflages* the edges of the maximum matching, in the sense that they are statistically indistinguishable to the algorithm from the rest of the edges in the core that do not participate in a maximum matching. This is the key feature of the new construction in [42] that obtains the first super-linear query lower bound of $\Omega(n^{1.2})$ for approximating maximum matching.

In a little more detail, it was shown in [42] that so long as the average degree in the core is not too large (say smaller than $n^{0.2}$) and the algorithm does not conduct too many queries (say smaller than $n^{1.2}$), then the discovered edges of the core will form a forest. This enables [42] to argue that the algorithm cannot distinguish the edges of the maximum matching from the rest of core edges by reducing the problem to a label guessing game on trees.

The Cycle Discovery Barrier

The assumption that the algorithm cannot discover any cycles in the core completely breaks when the algorithm is allowed to make $\omega(n^{1.5})$ queries, making it particularly challenging to prove such lower bounds. To provide some intuition about this, suppose that we take a vertex v and run a BFS from it (discarding dummy vertices) until discovering $\Theta(\sqrt{n})$ vertices of the core. Note that this takes only $O(n\sqrt{n})$ queries even if we query the whole adjacency list of each encountered core vertex. Informally speaking, if we run this BFS from two random starting vertices, then by the birthday paradox, we expect their discovered descendants to collide, therefore forming cycles.

At first glance, this may seem like a limitation of existing lower bounds proofs rather than a strength of these algorithms. However, we remark that there is indeed an algorithm running in $\widetilde{O}(n\sqrt{n})$ time that solves the construction of [42]. It is also worth noting that the cycle discovery threshold does indeed represent the correct bound for other problems in the sublinear time model. For instance, Goldreich and Ron [107] first gave a lower bound of $\Omega(\sqrt{n})$ for bipartite testing, using also the assumption that faster algorithms cannot discover cycles. Later, in a follow up work, they showed that there is indeed an algorithm running in $\widetilde{O}(\sqrt{n})$ time for this problem [106].

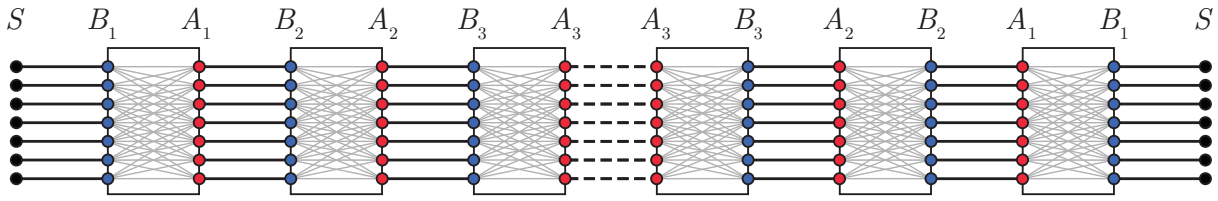
To recap, the approach in previous work [42] was to camouflage the edges of the good matching. The limitation of the previous approach is that once the algorithm gets $n^{1.5}$ queries, it can discover at least $\sqrt{n}$ core edges, at which point the algorithm discovers cycles. And cycles break the camouflage of the edges on the cycle.

Our Key Contribution: Bypassing the Cycle Discovery Barrier

Our key novel idea in this work to bypass the cycle discovery barrier is to camouflage the entire core instead of just the maximum matching. How do we camouflage the entire core? Roughly the same way that previous work camouflaged the good matching! This (in hindsight) inspires our construction: we have a recursive construction of L *levels*; the i-th level is similar to the entire construction of [41], with the main difference

being that we replace the hidden good matching with the $(i-1)$-level construction. To provide more details, let us first overview the base of the construction due to [41].

The base (due to [41]): Consider the graph illustrated below with $2r+1$ subsets of vertices $A_1, \ldots, A_r$, $B_1, \ldots, B_r$, and S, where r is a parameter of the construction (it is instructive to take $r = 1/\varepsilon$). For any $i \in [r]$, there is an $n^{2\varepsilon}$-regular bipartite graph between A_i and B_i that we call a block. There is a perfect matching from A_i to B_{i+1}, a perfect matching from S to B_1, and a perfect matching from A_r to A_r (which may or may not exist). We call the edges of these perfect matchings *special* edges.



It is not hard to see that any algorithm achieving better than a $(1 - \frac{1}{2r+1})$ approximation must verify whether the A_r-A_r matching exists. Therefore, it suffices to show that near quadratic queries are needed to determine this.[3] To do so, we would like to argue a vertex v does not know which of its edges are special, thus it has to do a BFS of depth $r = 1/\varepsilon$ to reach the S vertices, exploring $\Omega((n^{2\varepsilon})^r) = \Omega(n^2)$ edges. The problem, however, is exactly the cycle discovery problem. The birthday-paradox argument discussed earlier can be used to test in $O(n^{1.5-2\varepsilon})$ time whether two vertices u and v are in the same block. Therefore, a vertex can find its special edge in just $O(n^{1.5-2\varepsilon}) \cdot O(n^{2\varepsilon}) = O(n^{1.5})$ time by running this test on all of its $n^{2\varepsilon}$ neighbors. Continuing along these special edges, we reach S in just $O(rn^{1.5}) = O_\varepsilon(n^{1.5})$ time overall.

The recursion (new to this work): To resolve this problem, we give a recursive construction. In particular, we will define a sequence of input constructions denoted as $G^1, \ldots, G^L$, where G^1 is (essentially) the construction discussed above. The construction of G^ℓ is the same as our construction for G^1, except that we replace the special edges (i.e., the perfect matchings) with the graph of the previous level $G^{\ell-1}$. The figure below illustrates this.

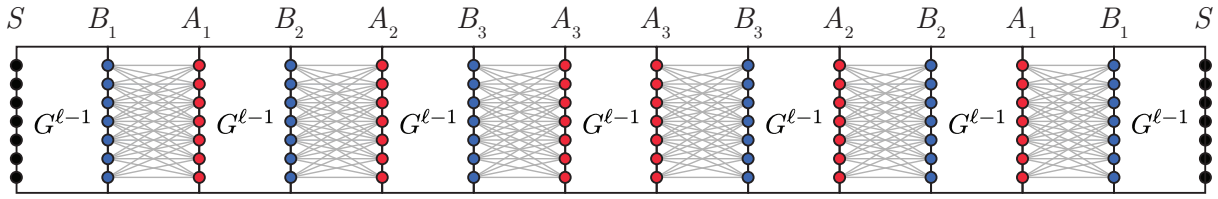


Figure 4.6: Construction of G^ℓ based on $G^{\ell-1}$.

Let us use $n^{2-\delta}$ to denote the number of queries that the algorithm makes and use n^σ to denote the degrees in the regular blocks of graph G^ℓ. The key to our analysis is to show that while we discover *some* cycles which "spoil" the camouflage of some edges, the number of cycles, and hence also the number of spoiled edges, decreases by an $n^{\delta-\sigma}$ factor at each level. With a sufficiently large number of levels, we can

[3] We remark that in our final construction, we ensure that the A_r vertices have the same degrees as vertices in other layers no matter whether the A_r-A_r matching exists. We hide these details here for our informal overview of our lower bound.

ensure that the total decrease in the number of spoiled edges is significant. This ensures that at the bottom level G^1, we cannot discover any cycles at all. Consequently, we can safely camouflage the edges of the good matching and prove our lower bound using the previously known techniques when there exists no cycle. Throughout the remainder of this technical overview, our main focus is to provide a high-level intuition for why this decrease in advantage occurs when we move one level down in the construction.

Decrease in the Advantage of the Algorithm in Discovering Camouflaged Edges

To understand this decrease in the algorithm's advantage in detecting camouflaged edges, examine the highest level in the recursive construction, denoted as G_L. For the remainder of this section, we say that an edge discovered by the algorithm is directed from u to v if it is obtained by querying the adjacency list of u.

We claim that each vertex has a probability of $O(1/n)$ to be the answer to each adjacency list query. First, note that the gadgets that we are using in our construction are random regular graphs. Consider a pair of vertices (u, v) in the core construction that is not among the edges discovered by the algorithm. Let x and y be the number of undiscovered core edges of u and v, respectively. Using a coupling argument, we can show that there exists an edge between u and v with a probability $O(\min(x, y)/n)$ (see Lemma 4.4.17). Combining the above argument and the fact that the adjacency list of vertices is ordered uniformly at random implies that when the algorithm queries the adjacency list of vertex u, given that this vertex has x undiscovered edges, the probability of the answer to this query being a specific vertex v is bounded by $O(1/n)$ (see Claim 4.4.20). Applying this observation, we obtain several properties of the subgraph queried by the algorithm. We mention a few of these properties that are useful in our proof:

(P1) **Each vertex in the core has $O(\log n)$ incoming edges:** Consider a vertex v in the core. If we query the adjacency list of vertex u in the core, the probability of obtaining a directed edge (u, v) is bounded by $O(1/n)$. Given that a fraction of $O(n^\sigma/n)$ edges of the whole input graph are in the core, the algorithm is going to discover at most $O(n^{1-\delta+\sigma}) \ll O(n)$ edges of the core (see Claim 4.4.16). Hence, the expected number of incoming edges for each vertex is less than one. Using a concentration inequality, we can show that with high probability, v has at most $O(\log n)$ incoming edges (see Claim 4.4.21).

(P2) **Most edges in the core do not close a cycle:** As discussed in (P1), the algorithm can discover at most $O(n^{1-\delta+\sigma})$ edges of the core. Therefore, at any time during the execution of the algorithm, there are at most $O(n^{1-\delta+\sigma})$ vertices with at least one edge in the core. This implies that the probability that the answer to each new query made by the algorithm is a vertex for which the algorithm has previously found an incident edge in the core is $O(n^{\sigma-\delta})$. This suggests that the majority of edges in the core do not close a cycle, with only a fraction of $O(n^{\sigma-\delta})$ closing cycles.

(P3) **Local directed neighborhood of most vertices in core is a small tree:** For a vertex v in core, let the *shallow subgraph* of v, denoted $T(v)$, be the set of vertices that are reachable from v using queried directed paths of length at most $O(\log n)$ with edges in the core. Now consider all the edges in the core. Using a stronger argument similar to (P1), we can show that each queried edge by the algorithm belongs to at most $\operatorname{poly}\log(n)$ different shallow subgraphs. Therefore, we have $\sum_v |T(v)| \leq \widetilde{O}(n^{1-\delta+\sigma})$. We say a shallow subgraph is *small* if it has less than $n^{\delta-2\sigma}$ vertices, and *large* otherwise. By our bound on $\sum_v |T(v)|$, we can have at most $\widetilde{O}(n^{1-2\delta+3\sigma})$ large shallow subgraphs. On the other hand, only

$O(n^{\sigma-\delta})$-fraction of the core edges close a cycle by (P2), aka there are at most $O(n^{1-2\delta+2\sigma})$ such edges. Consequently, there are at most $\widetilde{O}(n^{1-2\delta+2\sigma})$ shallow subgraphs that have a cycle. As a result, the local directed neighborhood of most of the vertices is a tree of size $O(n^{\delta-2\sigma})$.

For formal proof of (P2) and (P3), we encourage readers to see Lemma 4.4.28. Suppose that we define vertex labels similarly as discussed, i.e. vertices can have labels $A_1, \ldots, A_r$, $B_1, \ldots, B_r$. For a vertex with property (P3), referred to as an *unspoiled* vertex, we can demonstrate that the algorithm is incapable of distinguishing the vertex's label. The technical proof for this part is mostly borrowed from [41] and uses the fact that in each level of our construction, the graph is very similar to the construction of [41]. Finally, we can argue that for all edges with unspoiled endpoints, the algorithm has a negligible bias in the probability of the edge belonging to a lower level (gadget between A_r and A_r). More formally, considering the bound obtained in (P3), there are at most $n^{1-2\delta+4\sigma}$ spoiled vertices. Consequently, there are at most $n^{1-2\delta+5\sigma}$ edges for which the algorithm has a significant bias in the probability that they belong to a lower level (see Lemma 4.4.30). We encourage readers to refer to the warm-up presented in Section 4.4.4, as many of the ideas mentioned here are discussed in more detail there, and it contains many key ideas essential to our proof.

Assume that the degree of vertices for inner level G^{L-1} are asymptotically smaller, i.e. $n^{\sigma'}$ where $\sigma' < \sigma$. Exclude the edges—amounting to $n^{1-2\delta+5\sigma}$—that the algorithm distinctly identifies as belonging to a lower level due to a significant bias in probability. For all remaining edges, due to the minor bias, the probability of the edge belonging to level $L-1$ is at most $O(n^{\sigma'-\sigma})$. Intuitively, this implies that a majority of the edges belong to a higher level, and the algorithm is unable to form large connected components of the inner level using unbiased edges. To observe this contrast in the size of connected components, consider the following simple and intuitive example. At the highest level, the algorithm can concentrate all its queries to create a single large component of size $n^{1-\delta+\sigma}$. Now, let us suppose the algorithm is executing a BFS from an arbitrary vertex in the graph to create large components of inner edges using unbiased edges. In each step, the algorithm queries all neighbors during BFS. It is noteworthy that each edge belongs to the inner level with a probability of $O(n^{\sigma'-\sigma})$. Consequently, the size of the largest component with inner edges is $O(n^{(\sigma'/\sigma)(1-\delta+\sigma)}) \ll O(n^{1-\delta+\sigma})$. The decrease in the size of the connected components aids in demonstrating that, in the lower level, the count of vertices proximate to cycles is considerably smaller. By recursively applying this step, ultimately, we can reach the base level where we can prove, with high probability, the absence of cycles.

We point out that the informal outline above oversimplifies several important parts of our proof. Firstly, the construction discussed above as stated can be solved efficiently with a random-walk based argument. To resolve this, we add a number of *delusive* vertices (introduced before by [41]) to each level of the recursion where roughly speaking ε fraction of edges of each vertex go to these delusive vertices. Secondly, the degrees of the regular blocks and the number of blocks in each level of the recursion have to be balanced carefully. In particular, we need to ensure that the blocks in G^{ℓ} are sufficiently denser than those in $G^{\ell-1}$ to be able to argue that we see fewer cycles in $G^{\ell-1}$ than in G^{ℓ}. But having smaller degrees in $G^{\ell-1}$ requires increasing the number of blocks in $G^{\ell-1}$ to keep it essentially as "difficult" to solve as G^{ℓ}. Finally, the queries conducted at level G^{ℓ} reveal some information about the labels in the previous level $G^{\ell-1}$. This has to be quantified carefully in order to formalize the intuitive argument that the algorithm sees fewer cycles in $G^{\ell-1}$.

4.4.2 Table of Parameters

In this section, we present a table of variables (Table 4.1) employed in this proof. We assume that the algorithm makes $O(n^{2-\delta})$ queries. The table below provides definitions of these variables and their dependency on δ. While there is no imperative need to read this section, we have already introduced these variables in the relevant sections. We include this table to facilitate readers' comprehension of the interplay between these parameters in the context of the technical proofs.

4.4.3 Input Distribution and its Characteristics

In this section, we describe the construction of our input distribution. We will have two types of input distributions both on n vertices, which we denote by $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$. Any graph drawn from $\mathcal{D}_{\mathsf{YES}}$ will have a perfect matching which matches all n vertices. On the flip side, any maximum matching for a graph drawn from $\mathcal{D}_{\mathsf{NO}}$ will match at most $(1-\varepsilon)n$ vertices. Our final input distribution $\mathcal{D} := (\mathcal{D}_{\mathsf{YES}} + \mathcal{D}_{\mathsf{NO}})/2$ draws its graph either from $\mathcal{D}_{\mathsf{YES}}$ or $\mathcal{D}_{\mathsf{NO}}$, each with probability $1/2$. We show that any deterministic algorithm that can distinguish between a graph that is drawn from $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$, has to spend at least $\Omega(n^{2-\delta})$ time. We fix the dependency of δ on ε later in the proofs. Our main result will be the following:

Lemma 4.4.2. *Let G be drawn from $\mathcal{D}$. Any **deterministic** algorithm that provides an estimate $\widetilde{\mu}$ of the size of the maximum matching of G such that*

$$\mathbf{E}_G[\widetilde{\mu}] \geq \mu(G) - \varepsilon n,$$

will have to spend at least $\Omega(n^{2-\delta})$ time.

Plugging Lemma 4.4.2 into Yao's minimax theorem [175], we get our main result for randomized algorithms.

Proof of Theorem 4.4.1. Let $\mathcal{X}$ be the set of all possible inputs for the problem and $\mathcal{A}$ be the set of all possible deterministic algorithms. Also, let $c(a,x) \geq 0$ be the running time of the algorithm a on input x. By Lemma 4.4.2, we have $\min_{a\in\mathcal{A}} \mathbf{E}[c(a,\mathcal{D})] \geq \Omega(n^{2-\delta})$. Therefore, using Yao's minimax principle (Proposition 2.3.6), we have

$$\max_{x\in\mathcal{X}} \mathbf{E}[c(A,x)] \geq \min_{a\in\mathcal{A}} \mathbf{E}[c(a,\mathcal{D})] \geq \Omega(n^{2-\delta})$$

which implies that any randomized algorithm that estimates the size of the maximum matching with an additive error of εn must spend at least $\Omega(n^{2-\delta})$ time. □

For both $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$, our construction consists of L recursive *levels* of hierarchy. The level i graph is constructed by combining several graphs of level $i-1$ plus extra edges to increase the difficulty in distinguishing the edges of the level $i-1$ graphs. The high-level goal is to hide some of the edges of (one of) the level 1 graphs in the construction, which consists of a constant fraction of the maximum matching edges of the graph.

Parameter	Value	Definition
δ	-	Parameter that controls the running time of the algorithm. More specifically, the algorithm has $O(n^{2-\delta})$ running time.
L	$4/\delta$	Number of **levels** in the recursive hierarchy for the construction of input distribution.
r	$\left(\frac{10}{\delta}\right)^{L+1}$	Number of **layers** in the base construction (and in each level of the hierarchy).
$\mathcal{D}_{\mathsf{YES}}$	-	Distribution of graphs that have a perfect matching.
$\mathcal{D}_{\mathsf{NO}}$	-	Distribution of graphs that at most $(1-\varepsilon)$ fraction of their vertices can be matched in the maximum matching.
$\mathcal{D}^i_{\mathsf{YES}}$	-	Distribution of level i graphs in the construction hierarchy that have a perfect matching.
$\mathcal{D}^i_{\mathsf{NO}}$	-	Distribution of level i graphs that at most $(1-\varepsilon)$ fraction of their vertices can be matched in the maximum matching.
$\mathcal{D}$	$\frac{1}{2}\mathcal{D}_{\mathsf{YES}} + \frac{1}{2}\mathcal{D}_{\mathsf{NO}}$	Final input distribution.
σ_i	$\left(\frac{\delta}{10}\right)^{L+1-i}$	Parameter that controls the degree of vertices in graphs of level i.
d_i	$\Theta(n^{\sigma_i})$	Parameter that controls the degree of vertices in graphs of level i.
ζ	$1/r^2$	Fraction of vertices that are delusive in each level.
ξ	$1/r^2$	The gap between size of A_r and B_r in the base construction.
γ	$1/r^3$	Degree to delusive vertices is γd.
τ	$(20r^3)^{-L}$	Number of dummy vertices is τn.
N_i	$N_i = n_{i-1}/(2\zeta)$	Parameter that controls the number of vertices in graphs of level i.
n_i	$(8{+}16r{+}4\zeta r)N_i$	Total number of vertices in a graph of level i.
n	$(1+\tau)\cdot n_L$	Total number of vertices in a graph that is drawn from the final distribution.

Table 4.1: Variables used in the input distribution and proofs.

For each level i, there are two types of graphs which we call $\mathcal{D}^i_{\mathsf{YES}}$ and $\mathcal{D}^i_{\mathsf{NO}}$. Similar to the $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$, the two types of graphs for level i have different sizes of maximum matching. Also, each level of the

hierarchy consists of r *layers*. First, we show how we construct our level 1 graph (base level of the hierarchy). Next, we demonstrate how we can construct the core using a recursive process. Finally, we add some dummy vertices which are a small constant fraction of vertices in the graph and we connect them to all vertices in order to increase the cost of adjacency list queries. Our $\mathcal{D}_{\mathsf{YES}}$ will be $\mathcal{D}^L_{\mathsf{YES}}$ plus the dummy vertices and $\mathcal{D}_{\mathsf{NO}}$ will be $\mathcal{D}^L_{\mathsf{NO}}$ plus the dummy vertices. It is also noteworthy to mention that all graphs in our constructions are bipartite.

Base Level of the Hierarchy

Let N_1 and d_1 be two parameters that control the number of vertices and degree of vertices in the induced subgraph of the base level.

Vertex set: the vertex set of the base level consists of disjoint subsets of vertices A^j_i and B^j_i for $i \in [r]$ and $j \in \{1,2\}$. Also, for each $i \in [r]$, the base level consists of subsets of vertices D_i which we call *delusive* vertices. Finally, there are two subsets S^1 and S^2 in the construction. We have the following properties for the size of the subsets that we defined:

$$|S^j| = |A^j_i| = |B^j_i| = N_1 \qquad \forall i \in [r-1] \quad \& \quad j \in \{1,2\},$$

$$|B^1_r| = |B^2_r| = N_1, \qquad |A^1_r| = |A^2_r| = (1-\xi)N_1$$

$$|D_i| = \zeta N_1 \qquad \forall i \in [r]$$

Let n_1 be the total number of vertices in the base-level construction. Thus,

$$\begin{aligned} n_1 = |S^1| + |S^2| + \sum_{\substack{i\in[r]\\ j\in\{1,2\}}} |A^j_i| + |B^j_i| + \sum_{i\in[r]} |D_i| &= 2N_1 + 4rN_1 + \zeta r N_1 \\ &= (2 + 4r + 1/r)N_1 \qquad \text{(Since } \zeta = 1/r^2\text{)}. \end{aligned}$$

Furthermore, we assume that all subsets have even size.

Edge set: the edge set of $\mathcal{D}^1_{\mathsf{YES}}$ and $\mathcal{D}^1_{\mathsf{NO}}$ are slightly different such that $\mathcal{D}^1_{\mathsf{YES}}$ contains a perfect matching, however, a small fraction of vertices of graphs in $\mathcal{D}^1_{\mathsf{NO}}$ are unmatched in its maximum matching. The edge set consists of several biregular graphs between different subsets of vertices. Let X and Y be two different subsets of vertices. We use $\deg(X,Y)$ to show the degree of vertices of X in the induced regular graph between X and Y. In what follows, we determine the degree of vertices for different choices of X and Y. We have the following biregular graphs in both $\mathcal{D}^1_{\mathsf{YES}}$ and $\mathcal{D}^1_{\mathsf{NO}}$:

- Edges of vertices in S^j for $j \in \{1,2\}$:

$$\deg(S^j, B^j_1) = 1.$$

- Edges of vertices in B_1^j for $j \in \{1,2\}$:

$$\deg(B_1^j, S^j) = 1, \qquad \deg(B_1^j, A_1^j) = d_1, \qquad \deg(B_1^j, D_1) = r\gamma d_1.$$

- Edges of vertices in A_i^j for $i \in [r-1]$ and $j \in \{1,2\}$:

$$\deg(A_i^j, B_i^j) = d_1, \qquad \deg(A_i^j, B_{i+1}^j) = 1, \qquad \deg(A_i^j, D_i) = (r-i+1)\gamma d_1,$$
$$\deg(A_i^j, D_k) = \gamma d_1 \qquad \text{for } k < i.$$

- Edges of vertices in B_i^j for $1 < i \leq r$ and $j \in \{1,2\}$:

$$\deg(B_i^j, A_i^j) = d_1, \qquad \deg(B_i^j, A_{i-1}^j) = 1, \qquad \deg(B_i^j, D_i) = (r-i+1)\gamma d_1,$$
$$\deg(B_i^j, D_k) = \gamma d_1 \qquad \text{for } k < i.$$

- Edges of vertices in D_r:

$$\deg(D_r, D_r) = d_1 + 1 + \gamma d_1(1 - 4/\zeta + 2\xi/\zeta),$$
$$\deg(D_r, A_r^j) = (1-\xi)\gamma d_1/\zeta, \qquad \deg(D_r, B_r^j) = \gamma d_1/\zeta \qquad \text{for } j \in \{1,2\},$$
$$\deg(D_r, D_i) = \gamma d_1 \qquad \text{for } i \in [r-1].$$

- Edges of vertices in D_i for $i \in [r]$:

$$\deg(D_i, D_i) = d_1 + 1 + \gamma d_1 - 2\gamma d_1(4r - 8i - \xi + 2)/\zeta,$$
$$\deg(D_i, A_i^j) = (r-i+1)\gamma d_1/\zeta, \qquad \deg(D_i, B_i^j) = (r-i+1)\gamma d_1/\zeta \qquad \text{for } j \in \{1,2\},$$
$$\deg(D_i, D_k) = \gamma d_1 \qquad \text{for } k \neq i,$$
$$\deg(D_i, A_k^j) = \gamma d_1/\zeta, \qquad \deg(D_i, B_k^j) = \gamma d_1/\zeta \qquad \text{for } i < k < r \text{ and } j \in \{1,2\},$$
$$\deg(D_i, A_r^j) = (1-\xi)\gamma d_1/\zeta, \qquad \deg(D_i, B_r^j) = \gamma d_1/\zeta \qquad j \in \{1,2\}.$$

Neighbors of vertices A_r^j and B_r^j for $j \in \{1,2\}$ are slightly different in $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$. In $\mathcal{D}_{\mathsf{YES}}$, we add a random perfect matching between A_r^1 and A_r^2. Also, there exists a biregular graph between A_r^j and B_r^j such that the degree of vertices in A_r^j is d_1 and the degree of vertices in B_r^j is $(1-\xi)d_1$. Finally, we have a bipartite (ξd_1)-regular graph between vertices of B_r^1 and B_r^2. Hence, the degrees are as follows in $\mathcal{D}_{\mathsf{YES}}$:

- Edges of vertices in B_r^j:

$$\deg(B_r^j, A_r^j) = (1-\xi)d_1, \qquad \deg(B_r^j, A_{r-1}^j) = 1, \qquad \deg(B_r^j, B_r^{3-j}) = \xi d_1,$$
$$\deg(B_r^j, D_k) = \gamma d_1 \qquad k \in [r].$$

- Edges of vertices in A_r^j:

$$\deg(A_r^j, B_r^j) = d_1, \qquad \deg(A_r^j, A_r^{3-j}) = 1,$$

$$\deg(A_r^j, D_k) = \gamma d_1 \qquad k \in [r].$$

In $\mathcal{D}_{\mathsf{NO}}$, we remove a $(1-\xi)N_1$ edges of a perfect matching of subgraph between B_r^1 and B_r^2. Let $\overline{B}_r^1$ and $\overline{B}_r^2$ be the set of vertices that are endpoints of the perfect matching in B_r^1 and $\overline{B}_r^2$, respectively. Also, we do not have a perfect matching between vertices of A_r^1 and A_r^2. Instead, we add a perfect matching between vertices of A_r^j and $\overline{B}_r^j$ for $j \in \{1,2\}$.

Note that for each of the regular bipartite subgraphs that we used in our construction, we choose one uniformly at random graph among all possible biregular graphs with specific degrees. Also, we assume that upon querying a vertex by the algorithm, if it belongs to S^1 or S^2, we immediately reveal the label of the vertex. What is hidden from the algorithm is whether the vertex belongs to subset A, B, or D and the layer it belongs to. Now, we proceed to prove some characteristic properties of our base-level construction. The following observations are immediately implied by the construction.

Remark 6. *To maintain the graphs bipartite, it is necessary to have two subsets within each D_i because there are edges within each subset D_i. However, for the sake of simplicity in our construction, we omitted this aspect. It is possible to assume the presence of two subsets within each D_i and add edges between these subsets to preserve the bipartite property of the graphs.*

Observation 4.4.3. *Let $v \in S^1 \cup S^2$. Then, the degree of v in the base-level construction is 1.*

Observation 4.4.4. *Let $v \notin S^1 \cup S^2$. Then, the degree of v in the base level construction is $d_1 + r\gamma d_1 + 1 = \Theta(d_1)$.*

Based on the two aforementioned observations, the degrees of all vertices are identical at the base level, except for vertices in $S^1 \cup S^2$. Additionally, as γ is a small constant, we can assume that all vertices have approximately $O(d_1)$ neighbors at the base level. Next, we will demonstrate the contrast in the size of the maximum matching between a graph drawn from $\mathcal{D}_{\mathsf{YES}}$ and one drawn from $\mathcal{D}_{\mathsf{NO}}$.

Lemma 4.4.5. *Let $G^1_{\mathsf{YES}} \sim \mathcal{D}^1_{\mathsf{YES}}$ and $G^1_{\mathsf{NO}} \sim \mathcal{D}^1_{\mathsf{NO}}$. Then, we have*

$$\mu(G^1_{\mathsf{YES}}) = \frac{n_1}{2} \qquad \textit{and} \qquad \mu(G^1_{\mathsf{NO}}) \leq \frac{n_1}{2} - N_1.$$

Proof. First, we prove that G^1_{YES} contains a perfect matching. There exists a perfect matching between the following subsets of vertices:

- S^j and B_1^j for each $j \in \{1,2\}$,
- A_i^j and B_j^{i+1} for each $i \in [r-1]$ and $j \in \{1,2\}$,
- A_r^1 and A_r^2,
- induced subgraph of D_i for each $i \in [r]$ since the induced subgraph of vertices in D_i is a bipartite regular graph.

Therefore, we have $\mu(G^1_{\mathsf{YES}}) = n_1/2$. On the other hand, for G^1_{NO}, combining one part of the bipartite graph of vertices in D_i for all $i \in [r]$ and $\bigcup_{i=1,j\in\{1,2\}}^{r} B_i^j$ results in a vertex cover of the graph. Hence, using König's

Theorem, we get

$$\mu(G^1_{\mathsf{NO}}) \leq \sum_{i=1,j\in\{1,2\}}^{r} |B_i^j| + \sum_{i=1}^{r} |D_i|/2 = 2rN_1 + \frac{r\zeta N_1}{2} = (2r + \frac{1}{2r})N_1 = \frac{n_1}{2} - N_1. \quad \square$$

The Recursive Hierarchy

In this subsection, we show how we obtain our final construction from the base-level construction using a recursive procedure. We construct $\mathcal{D}^{\ell}_{\mathsf{YES}}$ and $\mathcal{D}^{\ell}_{\mathsf{NO}}$ from $\mathcal{D}^{\ell-1}_{\mathsf{YES}}$ and $\mathcal{D}^{\ell-1}_{\mathsf{NO}}$ for $1 < \ell \leq L$. Similar to the base level construction, each level has r layers of vertices. Similarly, we have subsets A_i^1, A_i^2, B_i^1, and B_i^2 for $i \in [r]$. Moreover, we have two subsets S^1, S^2. However, instead of having one subset D_i for $i \in [r]$, we have four subsets D_i^j for $1 \leq j \leq 4$. We let $D_i = \bigcup_{j=1}^{4} D_i^j$. We let A_i denote $A_i^1 \cup A_i^2$ (resp. B_i denote $B_i^1 \cup B_i^2$ and S denote $S^1 \cup S^2$). Henceforth, when we mention a vertex's membership in subset X at level ℓ of the hierarchy, we are referring to X one of the sets A_i, B_i, D_i, or S.

Let N_ℓ and d_ℓ be two parameters that control the number of vertices and degree of vertices in the graph of level ℓ. We have that $N_\ell = n_{\ell-1}/(2\zeta)$. We have the following properties for the sizes of the subsets that we defined:

$$|S^j| = |A_i^j| = |B_i^j| = 4N_\ell \qquad \forall i \in [r-1] \quad \& \quad j \in \{1,2\},$$

$$|B_r^1| = |B_r^2| = 4N_\ell, \qquad |A_r^1| = |A_r^2| = 4N_\ell$$

$$|D_i^j| = \zeta N_\ell \qquad \forall i \in [r] \quad \& \quad 1 \leq j \leq 4,$$

Let n_ℓ be the total number of vertices in level ℓ of construction. Thus,

$$\begin{aligned} n_\ell = |S^1| + |S^2| + \sum_{\substack{i\in[r]\\ j\in\{1,2\}}} |A_i^j| + |B_i^j| + \sum_{i\in[r]} |D_i| &= (8 + 16r + 4\zeta r)N_\ell \\ &= (4/\zeta + 8r/\zeta + 2r)n_{\ell-1} \qquad (\text{Since } N_\ell = n_{\ell-1}/(2\zeta)). \end{aligned}$$

We can also write the number of vertices in level ℓ in terms of N_1, which is the parameter that controls the number of vertices in the base level.

Observation 4.4.6. *It holds that* $n_\ell = (2 + 4r + \zeta r) \cdot (4/\zeta + 8r/\zeta + 2r)^{\ell-1} \cdot N_1$.

Observation 4.4.7. $n_L \leq (9r^3)^L \cdot N_1$.

Proof. By Observation 4.4.6, we have

$$\begin{aligned} n_L &= (2 + 4r + \zeta r) \cdot (4/\zeta + 8r/\zeta + 2r)^{L-1} \cdot N_1 \\ &\leq (4r^2 + 8r^3 + 2r)^L \cdot N_1 && (\text{Since } \zeta = 1/r^2) \\ &\leq (9r^3)^L \cdot N_1 && (\text{Since } r \geq 10). \quad \square \end{aligned}$$

Furthermore, we assume that all subsets have even size. The following edges are common in both $\mathcal{D}^{\ell}_{\mathsf{YES}}$ and $\mathcal{D}^{\ell}_{\mathsf{NO}}$:

- For $j \in \{1,2\}$, there are $4/\zeta$ bipartite graphs that are drawn from $\mathcal{D}^{\ell-1}_{\mathsf{YES}}$ with disjoint vertex sets between S^j and B^j_1.
- For $j \in \{1,2\}$ and $i \in [r-1]$, there are $4/\zeta$ bipartite graphs that are drawn from $\mathcal{D}^{\ell-1}_{\mathsf{YES}}$ with disjoint vertex sets between B^j_i and A^j_{i+1}.
- For $i \in [r]$ and $j \in \{1,3\}$, there exists a bipartite graph that is drawn from $\mathcal{D}^{\ell-1}_{\mathsf{YES}}$ between D^j_i and D^{j+1}_i.

Also, the edge set contains several biregular graphs similar to the construction of the base level. In what follows, we determine the degree of vertices for different choices of X and Y using the same notation of $\deg(X,Y)$.

- Edges of vertices in A^j_i for $i \in [r]$ and $j \in \{1,2\}$:

$$\deg(A^j_i, B^j_i) = d_\ell, \qquad \deg(A^j_i, D_i) = (r-i+1)\gamma d_\ell,$$
$$\deg(A^j_i, D_k) = \gamma d_\ell \qquad \text{for } k < i.$$

- Edges of vertices in B^j_i for $i \in [r]$ and $j \in \{1,2\}$:

$$\deg(B^j_i, A^j_i) = d_\ell, \qquad \deg(B^j_i, D_i) = (r-i+1)\gamma d_\ell,$$
$$\deg(B^j_i, D_k) = \gamma d_\ell \qquad \text{for } k < i.$$

- Edges of vertices in D_i for $i \in [r]$:

$$\deg(D_i, A^j_i) = (r-i+1)\gamma d_\ell/\zeta, \qquad \deg(D_i, B^j_i) = (r-i+1)\gamma d_\ell/\zeta \qquad \text{for } j \in \{1,2\},$$
$$\deg(D_i, A^j_k) = \gamma d_\ell/\zeta, \qquad \deg(D_i, B^j_k) = \gamma d_\ell/\zeta \qquad \text{for } k > i \text{ and } j \in \{1,2\}.$$

Further, for $i \in [r]$ and $j \in \{1,2\}$, there exists a biregular graph between D^j_i and D^{j+2}_i with degree $d_\ell + \gamma d_\ell - 4\gamma d_\ell(2r-2i+1)/\zeta$. Also, since we have four parts in each D_i, we can add edges between other vertices and corresponding subsets in D_i to keep the graph bipartite. For simplicity, we skip the detailed degrees of this part since it is only important to keep the graph bipartite and the reader can assume that we have a set D_i and ignore about how edges are inside the set.

The only difference between $\mathcal{D}^{\ell}_{\mathsf{YES}}$ and $\mathcal{D}^{\ell}_{\mathsf{NO}}$ is the subgraph between A^1_r and A^2_r. In $\mathcal{D}^{\ell}_{\mathsf{YES}}$, this subgraph is drawn from $\mathcal{D}^{\ell-1}_{\mathsf{YES}}$ and in $\mathcal{D}^{\ell}_{\mathsf{NO}}$, this subgraph is drawn from $\mathcal{D}^{\ell-1}_{\mathsf{NO}}$. The following observations are immediately implied by the construction.

Observation 4.4.8. *Let G be a graph that is drawn from $\mathcal{D}^{\ell}_{\mathsf{YES}}$ or $\mathcal{D}^{\ell}_{\mathsf{NO}}$. Suppose that we remove all subgraphs that are drawn from $\mathcal{D}^{\ell-1}_{\mathsf{YES}}$ and $\mathcal{D}^{\ell-1}_{\mathsf{NO}}$ during the recursive construction of G. Then, the degree of each vertex in $S^1 \cup S^2$ is 0. Moreover, the degree of vertices that are not in $S^1 \cup S^2$ is $d_\ell + \gamma d_\ell$.*

Observation 4.4.9. *Degree of vertices in a graph that is drawn from $\mathcal{D}^{\ell}_{\mathsf{YES}}$ or $\mathcal{D}^{\ell}_{\mathsf{NO}}$ is $O(d_\ell)$.*

Observation 4.4.10. *For every pair of vertices u and v, there is a unique level ℓ such that if there is an edge between them at all, it must belong to level ℓ.*

Lemma 4.4.11. *Let $G^{\ell}_{\mathsf{YES}} \sim \mathcal{D}^{\ell}_{\mathsf{YES}}$ and $G^{\ell}_{\mathsf{NO}} \sim \mathcal{D}^{\ell}_{\mathsf{NO}}$. Then, we have*

- $\mu(G^{\ell}_{\mathsf{YES}}) = n_{\ell}/2$,
- $\mu(G^{\ell}_{\mathsf{NO}}) \leq n_{\ell}/2 - N_1$.

Proof. We use induction to prove this lemma. For the base case where $\ell = 1$, the proof follows by Lemma 4.4.5. Similar to the proof of Lemma 4.4.5, we can show that G^{ℓ}_{YES} has a perfect matching since there exists a perfect matching between the following subsets of vertices:

- S^j and B^j_1 for each $j \in \{1, 2\}$,
- A^j_i and B^j_{i+1} for each $i \in [r-1]$ and $j \in \{1, 2\}$,
- A^1_r and A^2_r,
- D^1_i and D^2_i for all $i \in [r]$,
- D^3_i and D^4_i for all $i \in [r]$.

All the above subgraphs are vertex disjoint and have a perfect matching because their subgraph is drawn from $\mathcal{D}^{\ell-1}_{\mathsf{YES}}$. Therefore, we have $\mu(G^{\ell}_{\mathsf{YES}}) = n_{\ell}/2$.

For G^{ℓ}_{NO}, note that if we remove edges between A^1_r and A^2_r, then the size of maximum matching is at most

$$\sum_{i=1, j\in\{1,2\}}^{r} |B^j_i| + \sum_{i=1}^{r} |D_i|/2 = 8rN_{\ell} + 2r\zeta N_{\ell} = (8r + \frac{2}{r})N_{\ell}, \tag{4.5}$$

since combining one part of the bipartite graph of vertices in D_i for all $i \in [r]$ and $\bigcup^{r}_{i=1, j\in\{1,2\}} B^j_i$ results in a vertex cover of the graph. Also, by the induction hypothesis, we have

$$\mu(G^{\ell}_{\mathsf{NO}}[A^1_r, A^2_r]) \leq \frac{4}{\zeta} \cdot \mu(G^{\ell-1}_{\mathsf{NO}}) \leq \frac{4}{\zeta} \cdot \Big(\frac{n_{\ell-1}}{2} - N_1\Big) \leq 4N_{\ell} - N_1. \tag{4.6}$$

Summing up (4.5) and (4.6), we obtain

$$\mu(G^{\ell}_{\mathsf{NO}}) \leq (8r + \frac{2}{r} + 4)N_{\ell} - N_1 = \frac{n_{\ell}}{2} - N_1. \quad \square$$

Adding Dummy Vertices

Finally, in both $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$, we add τn_L dummy vertices to the whole graph and connect these vertices to all other vertices in the graph. Also, we assume that τn_L is an even number and we keep the graph bipartite after adding τn_L vertices, i.e. half of the dummy vertices are connected to one part of the graph, and the other half are connected to the other part. Further, we assume that there is a perfect matching between dummy vertices in order to have a perfect matching in $\mathcal{D}_{\mathsf{YES}}$. The intuition behind adding dummy vertices to the graphs in our input distribution is that they will increase the cost of adjacency list queries while the

size of the matching does not change that much since τ is a very small constant. Moreover, we assume that the algorithm knows which vertices are dummy. We use *core* to denote the induced subgraph of all vertices excluding dummy vertices.

Observation 4.4.12. $\tau n_L \leq N_1/2$.

Proof. By Observation 4.4.7, we have

$$\begin{aligned}\tau n_L &\leq \tau \cdot (9r^3)^L \cdot N_1 \\ &= (20r^3)^{-L} \cdot (9r^3)^L \cdot N_1 && \text{(Because of our choice of } \tau) \\ &\leq N_1/2 && \square\end{aligned}$$

Claim 4.4.13. *Let $G_{\mathsf{YES}} \sim \mathcal{D}_{\mathsf{YES}}$ and $G_{\mathsf{NO}} \sim \mathcal{D}_{\mathsf{NO}}$. Then, we have*

- $\mu(G_{\mathsf{YES}}) = n_L \cdot (1+\tau)/2$,
- $\mu(G_{\mathsf{NO}}) \leq n_L/2 - N_1/2$.

Proof. Combining Lemma 4.4.11 and the fact that there exists a perfect matching in the induced subgraph of dummy vertices implies that G_{YES} has a perfect matching. Thus, $\mu(G_{\mathsf{YES}}) = n_L \cdot (1+\tau)/2$.

If we remove dummy vertices, the size of the maximum matching in G_{NO} is at most $n_L/2 - N_1$ by Lemma 4.4.11. On the other hand, there are at most τn_L edges in the maximum matching of G_{NO} with at least one dummy endpoint. Hence,

$$\mu(G_{\mathsf{NO}}) \leq n_L/2 - N_1 + \tau n_L \leq n_L/2 - N_1/2,$$

where the last inequality follows by Observation 4.4.12. $\square$

Lemma 4.4.14. *Let $\varepsilon = (\delta/400)^{100/\delta^2}$. Any algorithm that estimates the size of the maximum matching of a graph that is drawn from the input distribution with εn additive error must be able to distinguish whether it belongs to $\mathcal{D}_{\mathsf{YES}}$ or $\mathcal{D}_{\mathsf{NO}}$.*

Proof. Let $G_{\mathsf{YES}} \sim \mathcal{D}_{\mathsf{YES}}$ and $G_{\mathsf{NO}} \sim \mathcal{D}_{\mathsf{NO}}$. By Claim 4.4.13, we have

$$\mu(G_{\mathsf{YES}}) - \mu(G_{\mathsf{NO}}) \geq n_L \cdot (1+\tau)/2 - n_L/2 + N_1/2 \geq N_1/2.$$

So it is enough to show that $\varepsilon n \leq N_1/2$. To see this

$$\begin{aligned}\varepsilon n = \varepsilon n_L(1+\tau) &\leq 2\varepsilon n_L \\ &\leq 2\varepsilon \cdot (9r)^{3L} \cdot N_1 && \text{(By Observation 4.4.7)} \\ &\leq 2\varepsilon \cdot (90/\delta)^{100/\delta^2} \cdot N_1 && \text{(Because of our choices of } r \text{ and } L) \\ &\leq N_1/2 && \text{(Since } \varepsilon = (\delta/400)^{100/\delta^2}). \quad \square\end{aligned}$$

Furthermore, we want to stress that the adjacency list of each vertex includes its neighbors in a random order. This ordering is chosen uniformly and independently for each vertex. In the rest of the proof, we

assume that $d_1 = n^{\sigma_1}, \ldots, d_L = n^{\sigma_\ell}$ where $\sigma_L \gg \sigma_{L-1} \gg \ldots \gg \sigma_1$. More specifically, we have

$$\sigma_i = \left(\frac{\delta}{10}\right)^{L+1-i},$$

for $i \in [L]$. Also, we let $\sigma_{L+1} = 1$ and $\sigma_0 = 0$.

4.4.4 Warm-Up: The algorithm cannot identify many edges that do not belong to the top level

It is important to keep in mind that the difference between a graph that is drawn from $\mathcal{D}_{\mathsf{YES}}$ and a graph that is drawn from $\mathcal{D}_{\mathsf{NO}}$ stems from the subgraph between A_r^1 and A_r^2 of the highest level. In $\mathcal{D}_{\mathsf{YES}}$, this subgraph is drawn from $\mathcal{D}_{\mathsf{YES}}^{L-1}$ and in $\mathcal{D}_{\mathsf{NO}}$, this subgraph is drawn from $\mathcal{D}_{\mathsf{NO}}^{L-1}$. Thus, any algorithm that distinguishes between $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$, should find the difference in this subgraph. In this subsection, we provide an upper bound on the number of edges that the algorithm can identify as belonging to this subgraph. In the following definition, we establish the notion of identifying or distinguishing an edge that belongs to the subgraph A_r^1 and A_r^2 in the following definition. When the algorithm queries a typical edge, because of our choices of d_L and d_{L-1}, we expect the probability that this edge belongs to subgraph A_r^1 and A_r^2 to be roughly equal to $d_{L-1}/d_L = n^{\sigma_{L-1}-\sigma_L}$. We say an edge can be identified when the algorithm has a bias on this probability condition on the subgraph that is queried by the algorithm.

Definition 4.4.15 (p_e^{inner} and distinguishability of an edge)**.** *Let e be an edge that is queried by the algorithm. Also, let p_e^{inner} be the probability that this edge belongs to the subgraph between A_r^1 and A_r^2 conditioned on all queries made by the algorithm so far and assuming either input distribution. We say the algorithm can* distinguish *or* identify *if e belongs to the subgraph between A_r^1 and A_r^2 if $p_e^{inner} > 10n^{\sigma_{L-1}-\sigma_L}$.*

Note that each vertex is adjacent to $O(d_L)$ edges in our core by our choice of $d_1, d_2, \ldots, d_L$. Further, each vertex is adjacent to $\Omega(n)$ dummy vertices that we added to the construction in order to increase the cost of adjacency list queries inside the core. Since the adjacency list of each vertex is ordered uniformly at random, each query to the adjacency list of a vertex results in an edge in the core with probability $O(d_L/n)$. Hence, we expect to have $O(d_L \cdot n^{1-\delta}) = O(n^{1-\delta+\sigma_L})$ queries inside the core since there are at most $O(n^{2-\delta})$ queries in total. We prove that the number of edges that the algorithm can identify as belonging to the subgraph between A_r^1 and A_r^2 is upper bounded by $O(n^{1-2\delta+4\sigma_L})$. Moreover, we show that for all other edges, the probability that the edge belongs to the subgraph between A_r^1 and A_r^2 is $O(n^{\sigma_{L-1}-\sigma_L})$.

In the next claim, we give an upper bound on the total number of edges without a dummy endpoint that the algorithm can query.

Claim 4.4.16. *Any algorithm that makes at most $Q = O(n^{2-\delta})$ queries, identifies at most $O(n^{1-\delta+\sigma_L})$ edges of the core with high probability.*

Proof. There are at most $O(n^{1-\delta})$ vertices such that the algorithm makes more than $\tau n_L/2$ adjacency list queries to them since the total number of queries is $O(n^{2-\delta})$. For each vertex that the algorithm makes more than $\tau n_L/2$ queries, we assume that the algorithm finds all its incident edges in the core which is at most $O(d_L) = O(n^{\sigma_L})$ and in total is at most $O(n^{1-\delta+\sigma_L})$.

Now consider a vertex v with at most $\tau n_L/2$ adjacency list queries. At the beginning of the algorithm, each query to v's adjacency list is in the core with probability at most $O(n^{\sigma_L}/n)$. While the algorithm has

made at most $\tau n_L/2$ queries, the queries made have only negligible effect on this probability, so it remains true that each query to v's adjacency list is in the core with probability at most $O(n^{\sigma_L}/n)$. Let X_i be the event that the ith query returns an edge in the core and let $X = \sum X_i$. Thus, $\mathbf{E}[X_i] \leq O(n^{\sigma_L - 1})$ and $\mathbf{E}[X] \leq O(Q \cdot n^{\sigma_L - 1}) = O(n^{1-\delta+\sigma_L})$. Further, random variables X_is are negatively correlated. Therefore, using Chernoff bound we have

$$\Pr\left[|X - \mathbf{E}[X]| \geq 6\sqrt{\mathbf{E}[X]\log n}\right] \leq 2\exp\left(-\frac{(6\sqrt{\mathbf{E}[X]\log n})^2}{3\,\mathbf{E}[X]}\right) \leq \frac{1}{n^{10}},$$

which implies that with a probability of at least $1 - n^{-10}$, the total number of edges in the core that is discovered by the algorithm is $O(n^{1-\delta+\sigma_L})$. □

Let us consider a scenario where, instead of the bipartite subgraphs found in our input distribution, we had Erdos-Renyi subgraphs with the same expected degree as the regular graphs. In this case, for a pair of vertices between which the algorithm has not yet discovered an edge, the probability of an edge's existence was upper-bounded by $O(d/n)$, where d represents the expected degree of vertices in that subgraph. We extend this observation and employ a coupling argument to establish a similar property, which is formally articulated in the subsequent lemma, for the graphs generated from our input distribution.

Lemma 4.4.17. *Let (u, v) be a pair of vertices in the core that is not among discovered edges by the algorithm. Consider a time during the execution of the algorithm that u and v have x and y undiscovered core edges, respectively; suppose further that $x \leq y$. Then, there exists edge (u, v) in the graph with probability at most $O(x/n)$.*

Proof. First, if $x = 0$ the edge exists with probability zero. Now, suppose that $x > 0$. According to the construction, if there exists an edge between u and v, it only exists in one level of the recursive construction by Observation 4.4.10. Let X_u and X_v be the subsets in the construction that u and v belong to in that level. If there are no edges between X_u and X_v, then the probability of having edge (u, v) is zero. Let d_u be the number of neighbors of u in X_v and d_v be the number of neighbors of v in X_u. Also, let $\mathcal{G}_{(u,v)}$ be the set of all graphs in our input distribution that have all discovered edges in the core and edge (u, v). On the other hand let $\overline{\mathcal{G}_{(u,v)}}$ be the set of all graphs in our input distribution that have all discovered edges in the core and do not have edge (u, v). We prove that $|\mathcal{G}_{(u,v)}|/|\overline{\mathcal{G}_{(u,v)}}| = O(x/n)$ which implies that the probability of existence of edge (u, v) is upper bounded by $O(x/n)$.

To show this claim holds, for each graph $G_{(u,v)} \in \mathcal{G}_{(u,v)}$ we find all pairs (u', v') such that $u' \in X_u$, $v' \in X_v$, the induced subgraph of $\{u, u', v, v'\}$ exactly has two edge (u, v) and (u', v'), and edge (u', v') has not been discovered by the algorithm. Then, by removing edges (u, v) and (u', v'), and replacing them with edges (u, v') and (u', v) we get a graph in our input distribution that is in $\overline{\mathcal{G}_{(u,v)}}$.

We now argue that there are many such (u', v') pairs. First, recall that $|X_u| = \Omega(n)$, $|X_v| = \Omega(n)$, $d_u \ll n$, and $d_v \ll n$. Thus most vertices in X_v are not adjacent to u; in particular, $|X_v \setminus \mathcal{N}(u)| = \Omega(n)$. Let P be the set of all edges (w, z) such that $w \in X_v \setminus \mathcal{N}(u)$, $z \in X_u$ ((w, z) may or may not have been discovered by the algorithm). Since each vertex in X_v has d_v neighbors in X_u and $|X_v \setminus \mathcal{N}(u)| = \Omega(n)$, we get $|P| = \Omega(nd_v)$. Now let P' be the subset of edges in P that have not been discovered by the algorithm. By Claim 4.4.16, the total number of discovered edges by the algorithm is $o(n)$ which implies that $|P'| = \Omega(nd_v)$. It is not hard to see each pair $(u', v') \in P'$ satisfies all the required conditions.

Hence, we can map each graph in $\mathcal{G}_{(u,v)}$ to at least $\Omega(nd_v)$ graphs in $\overline{\mathcal{G}_{(u,v)}}$. Conversely, in the case of each graph in $\overline{\mathcal{G}_{(u,v)}}$, it can be mapped to a maximum of xy graphs in $\mathcal{G}_{(u,v)}$, considering that the remaining undiscovered edges of u and v are x and y, respectively. Therefore, by double counting the edge of the mapping from both sides, it holds

$$\frac{|\mathcal{G}_{(u,v)}|}{|\overline{\mathcal{G}_{(u,v)}}|} \leq \frac{O(xy)}{\Omega(nd_v)} \leq \frac{O(xd_v)}{\Omega(nd_v)} \leq O\left(\frac{x}{n}\right),$$

which completes the proof. Furthermore, it is crucial to mention that in this mapping, every graph in the support of $\mathcal{D}_{\mathsf{YES}}$ is mapped with graphs solely in the support of $\mathcal{D}_{\mathsf{YES}}$, and likewise, every graph in the support of $\mathcal{D}_{\mathsf{NO}}$ is mapped with graphs solely from the support of $\mathcal{D}_{\mathsf{NO}}$ since both u and u' belong to the same subset in the construction, and similarly, v and v' belong to the same subset in the construction as well. □

Corollary 4.4.18. *At any point during the execution of the algorithm, for any pair of vertices (u, v) in the core that is not among the edges already discovered by the algorithm, (u, v) is an edge in the graph with probability at most $O(n^{\sigma_L - 1})$.*

Proof. At any point during the execution of the algorithm, there are $O(n^{\sigma_L})$ undiscovered edges in the core incident on u or v. Plugging this into Lemma 4.4.17 we obtain the claimed bound. □

Definition 4.4.19 (Direction of an Edge)**.** *Let (u, v) be an edge that is queried by the algorithm by making a query to the adjacency list of vertex u. When we refer to the direction of edge (u, v), we are indicating that it goes from u to v.*

In the next claim, we show that for any fixed pair (u, v), when the algorithm queries u's adjacency list the answer is v with probability at most $O(1/n)$, even when conditioning on the query returning a non-dummy vertex.

Claim 4.4.20. *Suppose that the algorithm queries the adjacency list of vertex u in the core. Let v be a vertex in the core that the algorithm has not discovered edge (u, v) yet. Then, the probability of getting v as the answer to the adjacency list query of vertex u is at most $O(1/n)$.*

Proof. Suppose that there are x remaining undiscovered edges of u at the time that the algorithm is making a query to the adjacency list of u. By Lemma 4.4.17, the probability of having an edge between u and v is $O(x/n)$. Now assume that there exists an edge (u, v). Since u has x undiscovered edges and the adjacency list of vertices is sorted in a random order, the probability of v being the first one is $1/x$ condition on the edge existence. Therefore, the probability of getting v as the answer to the adjacency list query of vertex u is at most $O(1/n)$. □

As an application of Claim 4.4.20, we can demonstrate that each vertex in the graph has an indegree of $O(\log n)$ because they all have a nearly uniform probability of being the answer to the adjacency list queries.

Claim 4.4.21. *With high probability, the indegree of every vertex is at most $5 \log n$.*

Proof. Let k be the number of edges that the algorithm finds in the core. By Claim 4.4.16, we have $k = O(n^{1-\delta+\sigma_L})$. Consider an arbitrary vertex v. For $i \in [k]$, let X_i be the event that ith queried edge

in the core be an incoming edge to v. By Claim 4.4.20, we have that $\Pr[X_i = 1] = O(1/n)$ for all i. Let $X = \sum_{i=1}^{k} X_i$ and $\lambda = (4 \log n)/\mathbf{E}[X]$. Also, $0 < \mathbf{E}[X] < 1$ and thus, $\lambda > e^2$ for large enough n. Note that X_i's are negatively associated random variables. Using Chernoff bound for negatively associated variables, we have

$$\begin{aligned}
\Pr\left[X \geq (1+\lambda)\,\mathbf{E}[X]\right] &\leq \left(\frac{e^{\lambda}}{(1+\lambda)^{1+\lambda}}\right)^{\mathbf{E}[X]} \\
&\leq \left(\frac{e^{\lambda}}{\lambda^{\lambda}}\right)^{\mathbf{E}[X]} && \text{(Since } \lambda > 1\text{)} \\
&= \left(\frac{e}{\lambda}\right)^{4\log n} && \text{(Since } \lambda = (4\log n)/\mathbf{E}[X]\text{)} \\
&\leq \frac{1}{n^4} && \text{(Since } \lambda > e^2\text{)}
\end{aligned}$$

Since $(1+\lambda)\,\mathbf{E}[X] = \mathbf{E}[X] + 4\log n < 5\log n$, the probability that v has more than $5\log n$ incoming edges is at most n^{-4}. Using a union bound over all vertices we get the claimed bound. □

Definition 4.4.22 (Shallow Subgraph)**.** *For a vertex v, we let v's* shallow subgraph *be the set of vertices that are reachable from v using queried subgraph directed paths of length at most $10\log n$. We use $T(v)$ to denote v's shallow subgraph.*

We can utilize Claim 4.4.20 to establish a more robust proposition than what Claim 4.4.21 offers. To clarify, we can demonstrate that the algorithm is unable to concentrate outgoing edges towards nearby vertices. Consequently, the majority of vertices that are close together in the queried subgraph will have only one incoming edge. As a result, each vertex will be part of $\widetilde{O}(1)$ shallow subgraphs.

Lemma 4.4.23. *With high probability, each vertex is in at most $\widetilde{O}(1)$ shallow subgraphs.*

Proof. Let v be an arbitrary vertex in the core. Suppose that we run a BFS from u in the queried subgraph with reverse edge directions and let V_i be the set of vertices that are in distance i from v for $i \in [10\log n]$. We show that with high probability, we have $|V_i| \leq i\log^2 n$. We do this using induction. For $i = 1$, the claim is held by Claim 4.4.21. Suppose that the claim holds for all i' such that $i' < i$. Let $u \in V_{i-1}$. By Claim 4.4.20, the probability that the algorithm makes a query that is an incoming edge to u is at most $O(1/n) \leq \log n/n$ for a large enough n. Also, we have that $|V_{i-1}| \leq (i-1)\cdot\log^2 n$. Hence, the probability that a queried edge goes to one of the vertices in V_{i-1} is at most $(i-1)\cdot\log^3 n/n$. Let k be the total number of edges the algorithm finds in the core. By Claim 4.4.16, we have $k \leq n^{1-\delta+\sigma_L}\cdot\log n$.

For $i \in [k]$, let X_i be the event that ith queried edge in the core be an incoming edge to V_{i-1}. Thus, we have that $\Pr[X_i = 1] \leq (i-1)\cdot\log^3 n/n$ for all i. Let $X = \sum_{i=1}^{k} X_i$ and $\lambda = (4\log n)/\mathbf{E}[X]$. Hence, $\mathbf{E}[X] \leq (i-1)\cdot\log^4 n \cdot n^{\sigma_L-\delta}$. Also, $0 < \mathbf{E}[X] < 1$ and thus, $\lambda > e^2$ for large enough n. Note that X_i's are negatively associated random variables. Using Chernoff bound for negatively associated variables, we have

$$\begin{aligned}
\Pr\left[X \geq (1+\lambda)\,\mathbf{E}[X]\right] &\leq \left(\frac{e^{\lambda}}{(1+\lambda)^{1+\lambda}}\right)^{\mathbf{E}[X]} \\
&\leq \left(\frac{e^{\lambda}}{\lambda^{\lambda}}\right)^{\mathbf{E}[X]} && \text{(Since } \lambda > 1\text{)} \\
&= \left(\frac{e}{\lambda}\right)^{4\log n} && \text{(Since } \lambda = (4\log n)/\mathbf{E}[X]\text{)}
\end{aligned}$$

$$\leq \frac{1}{n^4} \qquad \text{(Since } \lambda > e^2\text{)}$$

which implies that $|V_i| \leq i \log^2 n$ since $(1+\lambda)\,\mathbf{E}[X] = \mathbf{E}[X] + 4\log n < i \log^2 n$. Therefore,

$$\sum_{i=1}^{10\log n} |V_i| \leq \sum_{i=1}^{10\log n} i \cdot \log^2 n \leq \widetilde{O}(1). \quad \square$$

Corollary 4.4.24. *With high probability, each edge that the algorithm finds in the core is in at most $\widetilde{O}(1)$ shallow subgraphs.*

Proof. For edge (u, v), by Lemma 4.4.23, u is in at most $\widetilde{O}(1)$ shallow subgraphs. Therefore, edge (u, v) is in at most $\widetilde{O}(1)$ shallow subgraphs. $\square$

Spoiled vertices: In essence, spoiled vertices are those in close proximity to short cycles using directed edges or having large shallow subgraphs. Later, we can prove that, for vertices distanced from short cycles or lacking large shallow subgraphs, the algorithm cannot distinguish if their incoming edges originate from the inner hierarchy level.

Before we formally define spoiled vertices, we define a closely related notion of *spoiler vertices*. Intuitively, spoiler vertices are ones where the idealized forest structure of the queried core subgraph is violated (or "spoiled").

Definition 4.4.25 (Spoiler Vertex)**.** *We say a vertex u in the core is* spoiler *if at least one of the following conditions holds:*

(i) vertex u has more than one incoming edge,

(ii) there is an edge (u, v) that is discovered by the algorithm at a time when v already has non-zero degree.

Spoiled vertices are ones that have, or expect to have, spoiler vertices in their shallow subgraphs.

Definition 4.4.26 (Spoiled Vertex)**.** *A vertex v in core is* spoiled *if its shallow subgraph contains any of the following:*

- *a spoiler vertex; or*
- *at least $n^{\delta - 2\sigma_L}$ vertices.*

The following observation is directly implied by the way we defined spoiler and spoiled vertices.

Observation 4.4.27. *Let v be a vertex that is not spoiled. Then, the shallow subgraph of v is a rooted tree of size at most $n^{\delta-2\sigma_L}$. Moreover, for each edge (u, w) in the shallow subgraph of v, at the time that the algorithm made the query, w was a singleton vertex.*

Proof. At the time that the algorithm discovers an edge (u, w) in the shallow subgraph of v, vertex w should be singleton according to Definition 4.4.25 and Definition 4.4.26. Therefore, the shallow subgraph of v is a rooted tree. $\square$

In the next lemma, we show that even among vertices for which the algorithm finds a core edge, the vast majority remain unspoiled.

Lemma 4.4.28. *With high probability, there are at most $O(n^{1-2\delta+4\sigma_L})$ spoiled vertices.*

Proof. First, note that by Corollary 4.4.24, each edge in the queried subgraph of core only appears in $\widetilde{O}(1)$ shallow subgraphs. Hence, $\sum_v |T(v)| \leq O(n^{1-\delta+2\sigma_L})$ by Claim 4.4.16. Therefore, the total number of vertices whose shallow subgraph contains more than $n^{\delta-2\sigma_L}$ vertices is $O(n^{1-2\delta+4\sigma_L})$.

We show that with high probability, there exists at most $O(n^{1-2\delta+3\sigma_L})$ spoiler vertices in the graph. By Lemma 4.4.23, since each vertex is in at most $\widetilde{O}(1)$ shallow subgraphs, there are at most $O(n^{1-2\delta+4\sigma_L})$ spoiled vertices. Thus, it suffices to upper bound the number of spoiler vertices.

At the time that we add an edge (u, v), the probability that v has a non-zero degree in core is $O(n^{\sigma_L-\delta})$ since by Claim 4.4.16, there are at most $O(n^{1-\delta+\sigma_L})$ vertices with a non-zero degree in the core and by Claim 4.4.20, each of them has a probability of $O(1/n)$ to be the queried edge of u. For such an edge, condition (ii) of Definition 4.4.25 holds for vertex u and condition (i) holds for vertex v. We assume that during the process of adding edges, for such an edge we count two spoiler vertices (for both endpoints).

Let X_i be the indicator of having a new spoiler vertex after adding ith edge. By the discussion above, we have $\Pr[X_i = 1] \leq O(n^{\sigma_L-\delta})$. Let k be the number of edges found by the algorithm in the core and $X = \sum_{i=1}^{k} X_i$. Thus, $\mathbf{E}[X] \leq O(n^{1-2\delta+2\sigma_L})$ since $k = O(n^{1-\delta+\sigma_L})$. Since events are negatively correlated, we get

$$\Pr\left[|X - \mathbf{E}[X]| \geq 6\sqrt{\mathbf{E}[X]\log n}\right] \leq 2\exp\left(-\frac{(6\sqrt{\mathbf{E}[X]\log n})^2}{3\,\mathbf{E}[X]}\right) \leq \frac{1}{n^{10}},$$

which implies that there are at most $O(n^{1-2\delta+3\sigma_L})$ different i such that $X_i = 1$. For each edge, if the indicator is one, we count a constant number of spoiler vertices which concludes the proof. □

Lemma 4.4.29. *Let v be a vertex that is not spoiled and belongs to $\{A_r, B_r, D_r\}$. Let $\mathcal{L}(v)$ and $\mathcal{L}'(v)$ be an arbitrary label for v from $\{A_r, B_r, D_r\}$ and the entire queried subgraph of core from all available labels of level L excluding the shallow subgraph of v. Then, we have*

$$\Pr[T(v) \mid \mathcal{L}(v)] \leq \left(1 + O(n^{\sigma_L-\delta})\right)^{|T(v)|} \cdot \Pr[T(v) \mid \mathcal{L}'(v)].$$

As we have proved, the shallow subgraph of an unspoiled vertex forms a rooted tree. This property allows us to show that all paths starting from the root of this rooted tree and reaching an S vertex or a short cycle, eventually step on a delusive vertex, which, in turn, causes a loss of information about anything below that delusive vertex. Consequently, we can couple the labelings that the tree's root is a vertex of A_r or B_r conditioning on labels of everything outside the shallow subgraph of the root. Hence, the probability that the algorithm queries this exact shallow subgraph no matter what the label of the root is and anything outside of the shallow subgraph. We defer the formal proof of the above lemma to Section 4.4.7 as we extend it to all levels of the construction.

Lemma 4.4.30. *With high probability, there are at most $O(n^{1-2\delta+5\sigma_L})$ edges e such that $p_e^{inner} > 10n^{\sigma_{L-1}-\sigma_L}$.*

Proof. Let $\widetilde{E}$ be the set of edges (u, v) (directed from u to v) such that $u \in A_r$ that satisfy at least one the following conditions:

(i) v is a spoiled vertex; or

(ii) u has at least $n^{\sigma_L}/3$ spoiled neighbors in the queried subgraph of core.

First, we show $|\widetilde{E}| \leq O(n^{1-2\delta+5\sigma_L})$. By Lemma 4.4.28, the number of spoiled vertices is at most $O(n^{1-2\delta+4\sigma_L})$. Moreover, by Claim 4.4.21, each vertex has at most $\widetilde{O}(1)$ indegree which implies that there are at most $O(n^{1-2\delta+5\sigma_L})$ edges that satisfy condition (i). On the other hand, if vertex u satisfies the condition (ii), it must have at least $n^{\sigma_L}/4$ edges (u, w) (directed from u to w) such that w is spoiled since each vertex has at most $\widetilde{O}(1)$ indegree (Claim 4.4.21). Since the total number of spoiled vertices is $O(n^{1-2\delta+4\sigma_L})$, there are at most $O(n^{1-2\delta+5\sigma_L})$ such u that satisfy condition (ii).

Now, we prove that for all other edges that are not in $\widetilde{E}$, we have that $p_e^{inner} \leq 10n^{\sigma_{L-1}-\sigma_L}$. Since $|\widetilde{E}| \leq O(n^{1-2\delta+5\sigma_L})$, the aforementioned claim will complete the proof of lemma. For edge $e = (u, v)$ that is directed from u to v, if $u \notin A_r$, it is easy to see that $p_e^{inner} = 0$. So assume that $u \in A_r$. Let $v_0 = v, v_1, v_2, \ldots, v_k$ be the neighbors of u in the core in the original graph such that $v_i \in B_r \cup A_r$ and either v_i is a singleton vertex in the queried subgraph or v_i is a directed child of u that is not spoiled. Since u does not satisfy condition (ii), then, $k \geq n^{\sigma_L}/2$. Now we bound the probability that vertex v_0 belongs to A_r by using a coupling argument and Lemma 4.4.29.

Consider a labeling profile $\mathcal{P}$ of all vertices $U = \{v_0, v_1, \ldots, v_k\}$ such that $\mathcal{P}(v_0) = A_r$. By the construction of our input distribution, since $u \in A_r$, at most $O(d_{L-1}) = O(n^{\sigma_{L-1}})$ vertices of U are in A_r. We produce $\Omega(n^{\sigma_L})$ new profiles $\mathcal{P}'$ such that $\mathcal{P}'(v_0) \neq A_r$. For each vertex v_i in U such that $\mathcal{P}(v_i) = B_r$, we construct a new profile $\mathcal{P}'$ where $\mathcal{P}(v_j) = \mathcal{P}'(v_j)$ for $j \notin \{0, i\}$, $\mathcal{P}'(v_i) = A_r$, and $\mathcal{P}'(v_0) = B_r$. By Lemma 4.4.29, the probability of querying the same shallow subgraphs $T(v_0)$ and $T(v_i)$ in the new labeling profile will be the same up to a factor of

$$\big(1 + O(n^{\sigma_L - \delta})\big)^{|T(v_0)|},$$

and

$$\big(1 + O(n^{\sigma_L - \delta})\big)^{|T(v_i)|},$$

respectively. Since v_0 and v_i are not spoiled vertices, $|T(v_0)| \leq n^{\delta - 2\sigma_L}$ and $|T(v_i)| \leq n^{\delta - 2\sigma_L}$ by Definition 4.4.26, thus, the probability of having profile $\mathcal{P}$ and $\mathcal{P}'$ are the same up to a factor

$$\big(1 + O(n^{\sigma_L - \delta})\big)^{|T(v_0)|} \cdot \big(1 + O(n^{\sigma_L - \delta})\big)^{|T(v_i)|} \leq \big(1 + O(n^{\sigma_L - \delta})\big)^{2n^{\delta - 2\sigma_L}} \leq 1 + o(1).$$

We construct a bipartite graph $H = (P_1, P_2, E_P)$ of labeling profiles such that in P_1, we have all profiles $\mathcal{P}$ where $\mathcal{P}(v_0) = A_r$, and in the P_2, all profiles $\mathcal{P}'$ where $\mathcal{P}'(v_0) = B_r$. We add an edge between two profiles $\mathcal{P}$ and $\mathcal{P}'$ if we can convert $\mathcal{P}$ to $\mathcal{P}'$ according to the above process. Therefore, $\deg_H(\mathcal{P}) \geq k/2 \geq n^{\sigma_L}/4$ for $\mathcal{P} \in P_1$ since at least $k/2$ vertices of U belong to B_r. On the other hand, $\deg_H(\mathcal{P}') \leq 2n^{\sigma_{L-1}}$ for $\mathcal{P}' \in P_2$. To see this, there are at most $2d_{L-1} = 2n^{\sigma_{L-1}}$ vertices v_i in U such that $\mathcal{P}'(v_i) = A_r$ according to the construction of input distribution. Hence,

$$p_{(u,v)}^{inner} \leq (1 + o(1)) \cdot \frac{|P_1|}{|P_2|} \leq (1 + o(1)) \cdot \frac{2n^{\sigma_{L-1}}}{n^{\sigma_L}/4} \leq (1 + o(1)) \cdot 8n^{\sigma_{L-1}-\sigma_L} \leq 10n^{\sigma_{L-1}-\sigma_L},$$

which concludes the proof. □

4.4.5 The Algorithm Cannot Create Large Connected Components of Inner Edges

In this section, we show that as we move downward in the recursive construction, it is harder for the algorithm to create components of large size using edges of the inner level. According to Lemma 4.4.30, in the highest level of the construction, for at most $O(n^{1-2\delta+5\sigma_L})$ edges in the queried subgraph of the core, the algorithm has the advantage to distinguish that these edges belong to the inner level with probability more than $10n^{\sigma_{L-1}-\sigma_L}$. We assume that the algorithm knows if these edges belong to the inner level or not with probability 1. However, for all other edges that the algorithm queries, it is more likely that those edges belong to the higher level because of the choices of degrees as formalized in Lemma 4.4.30. More specifically, each other edge that the algorithm queries, has a probability of at most $O(n^{\sigma_{L-1}-\sigma_L})$ to belong to the inner level. Our goal is to prove a similar lemma to Lemma 4.4.30 for each level in the next two sections. Intuitively, the following lemma shows that as we go down in the recursive construction, the number of edges that the algorithm can distinguish if they belong to the inner level decreases. First, we extend Definition 4.4.15 for all levels in the hierarchy.

Definition 4.4.31 ($p_e^{\ell-inner}$ and Distinguishability of an Edge)**.** *Let e be an edge that is queried by the algorithm. Also, let $p_e^{\ell-inner}$ be the probability that this edge belongs to the subgraph between A_r^1 and A_r^2 in level ℓ. We say the algorithm can distinguish or identify if e belongs to the subgraph between A_r^1 and A_r^2 if $p_e^{\ell-inner} > 10n^{\sigma_{\ell-1}-\sigma_\ell}$.*

Before proving Lemma 4.4.33, we need to define a function and its characteristics which are crucial to formalize the loss in advantage of the algorithm to identify edges. For the rest of the proof, we define function g for $\ell \in [L]$ as follows

$$g(\ell) = (L-\ell+2)\cdot\delta - 5\left(\sum_{i=\ell}^{L}\sigma_i/\sigma_{i+1}\right) - 5\left(\sum_{i=\ell}^{L-1}\sigma_i\right)$$

where, $\sigma_{L+1} = 1$. Also, we let $\sigma_0 = 0$. We have the following observations about the function g that are immediately implied by our choices for δ and σ_i for $i \in [L+1]$.

Observation 4.4.32. *The following statements are true regarding function g:*

(i) $g(\ell-1) = g(\ell) + \delta - 5\sigma_{\ell-1}/\sigma_\ell - 5\sigma_{\ell-1}$ *for* $\ell \in (1, L]$,

(ii) $1 - g(\ell-1) - 3\sigma_{\ell-1} > 1 - g(\ell) - \delta + 5\sigma_{\ell-1}/\sigma_\ell + \sigma_{\ell-1}$ *for* $\ell \in (1, L]$,

(iii) $g(1) > 2$,

(iv) $1 - g(\ell) \neq 0$ *for all* $\ell \in [L]$.

Proof. (i): By the definition of function g, we have

$$\begin{aligned} g(\ell-1) &= (L-\ell+3)\cdot\delta - 5\left(\sum_{i=\ell-1}^{L}\sigma_i/\sigma_{i+1}\right) - 5\left(\sum_{i=\ell-1}^{L-1}\sigma_i\right) \\ &= \left[(L-\ell+2)\cdot\delta - 5\left(\sum_{i=\ell}^{L}\sigma_i/\sigma_{i+1}\right) - 5\left(\sum_{i=\ell}^{L-1}\sigma_i\right)\right] + \delta - 5\sigma_{\ell-1}/\sigma_\ell - 5\sigma_{\ell-1} \end{aligned}$$

$$= g(\ell) + \delta - 5\sigma_{\ell-1}/\sigma_\ell - 5\sigma_{\ell-1}.$$

(ii): By statement (i), we get

$$1 - g(\ell-1) - 3\sigma_{\ell-1} = 1 - g(\ell) - \delta + 5\sigma_{\ell-1}/\sigma_\ell + 2\sigma_{\ell-1} > 1 - g(\ell) + 5\sigma_{\ell-1}/\sigma_\ell + \sigma_{\ell-1}.$$

(iii): By the definition of function g, we have

$$\begin{aligned} g(1) &= (L+1)\cdot\delta - 5\left(\sum_{i=1}^{L}\sigma_i/\sigma_{i+1}\right) - 5\left(\sum_{i=1}^{L-1}\sigma_i\right) \\ &= (L+1)\cdot\delta - \left(\frac{\delta L}{2}\right) - 5\left(\sum_{i=1}^{L-1}(\frac{\delta}{10})^{L+1-i}\right) && \text{(By Table 4.1)} \\ &> (L+1)\cdot\delta - \left(\frac{\delta L}{2}\right) - \delta \\ &= 2 && \text{(Since } L = 4/\delta\text{).} \end{aligned}$$

(iv): If $g(\ell)$ is zero for a particular ℓ, we can perturb the parameters in Table 4.1 to meet all constraints and make g(l) non-zero.

□

Lemma 4.4.33. *With high probability, the following statements hold:*

(i) If $1 - g(\ell) < 0$, then with probability $1 - O(n^{1-g(\ell)})$, there exist no edge e such that $p_e^{\ell-inner} > 10n^{\sigma_{\ell-1}-\sigma_\ell}$. Also, with high probability, there are at most $\widetilde{O}(1)$ edges e such that $p_e^{\ell-inner} > 10n^{\sigma_{\ell-1}-\sigma_\ell}$.

(ii) If $1 - g(\ell) > 0$, with high probability, there are at most $O(n^{1-g(\ell)})$ edges e such that $p_e^{\ell-inner} > 10n^{\sigma_{\ell-1}-\sigma_\ell}$.

Note that if we replace $\ell = L$ in the above bound, we get the same bound as Lemma 4.4.30. We use E_ℓ^{inner} to show the set of edges that $p_e^{\ell-inner} > 10n^{\sigma_{\ell-1}-\sigma_\ell}$. If the algorithm can distinguish the difference between a graph from $\mathcal{D}_{\mathsf{YES}}$ and a graph from $\mathcal{D}_{\mathsf{NO}}$, it should be able to distinguish between the subgraphs between A_r^1 and A_r^2 of level ℓ as other parts of the two graphs are similar. In this proof, when we mention the inner level, we only mean the subgraph between A_r^1 and A_r^2 of that level. In this section, we denote the edges between A_r^1 and A_r^2 of level ℓ as *black edges* and we denote other edges as *green edges*. We prove that the algorithm cannot grow a large component of black edges. The following lemma is the main technical contribution of this section.

Lemma 4.4.34. *Let $C_1, C_2, \ldots, C_c$ be the underlying undirected connected components of black edges where there exists at least one edge of E_ℓ^{inner} in each of the components. Then, the following statements hold:*

(i) If $1 - g(\ell) < 0$, then $c = 0$ with probability $1 - O(n^{1-g(\ell)})$. Also, $\sum_{i=1}^{c}|C_i| \le \widetilde{O}(n^{5\sigma_{\ell-1}/\sigma_\ell})$ with high probability.

(ii) If $1 - g(\ell) > 0$, we have $\sum_{i=1}^{c}|C_i| \le O(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell})$ with high probability.

We use induction to show the correctness of Lemma 4.4.33 and Lemma 4.4.34. For the base case, we already proved that Lemma 4.4.33 holds when $\ell = L$ (Lemma 4.4.30). To prove Lemma 4.4.34 for a fix ℓ, we use the bound from Lemma 4.4.33 for ℓ. Then, we use the result to prove Lemma 4.4.33 for $\ell - 1$. In this section, we focus on proving Lemma 4.4.34 using Lemma 4.4.33. In the rest of this subsection, we focus on the step to prove Lemma 4.4.34.

With the same argument as Claim 4.4.16, we can give an upper bound for the number of black edges which is formalized in Claim 4.4.35.

Claim 4.4.35. *There are at most $O(n^{1-\delta+\sigma_{\ell-1}})$ black edges with high probability.*

Proof. The proof is similar to the proof of Claim 4.4.16. We repeat the argument for completeness. For all vertices to which the algorithm makes more than $\tau n_L/2$ adjacency list queries, we assume that it discovers all its black edges. Since the algorithm makes at most $O(n^{2-\delta})$ queries in total, the total number of vertices with more than $\tau n_L/2$ queries cannot be larger than $O(n^{1-\delta})$ and therefore, the total discovered black edges incident to these vertices is at most $O(d_{\ell-1} \cdot n^{1-\delta}) = O(n^{1-\delta+\sigma_{\ell-1}})$.

For all other vertices, each adjacency list query is a black edge with a probability of $O(d_{\ell-1}/n) = O(n^{\sigma_{\ell-1}}/n)$. Since there are $O(n^{2-\delta})$ queries in total, with high probability, the algorithm will find at most $O(n^{1-\delta+\sigma_{\ell-1}})$ black edges using a Chernoff bound. □

According to Lemma 4.4.33, for all edges excluding those in E_ℓ^{inner}, when the algorithm queries an edge, it has a higher probability of being a green edge. This intuitively implies that, for any given vertex u, the algorithm should not be capable of discovering numerous descendants that are exclusively reachable through directed black edges, provided we disregard edges in E_ℓ^{inner}.

Lemma 4.4.36. *Consider all queried black edges in the core except edges E_ℓ^{inner}. With high probability, each vertex has at most $n^{5\sigma_{\ell-1}/\sigma_\ell}$ descendants that are reachable by directed black edges. Moreover, for each vertex, the total number of black edges to all its descendants is at most $n^{5\sigma_{\ell-1}/\sigma_\ell}$.*

Proof. Fix a vertex u. First, we claim that the probability of having a directed path of length i that starts from u and ends in a vertex v is bounded by $n^{i(\sigma_{\ell-1}-\sigma_\ell)/2}$. We use induction to prove this claim. For the base case where $i = 1$, if there is no edge between u and v this probability is 0. If there exists an edge, by Lemma 4.4.33, this edge is black with probability of at most $10n^{\sigma_{l-1}-\sigma_l} < n^{(\sigma_{l-1}-\sigma_l)/2}$. Suppose that the claim holds for all $i' < i$. By Claim 4.4.21, vertex v has at most $5 \log n$ indegree in the whole queried subgraph (including all edges). Let $\{v_1, v_2, \ldots, v_k\}$ be the set of vertices that have directed edge to v. Thus, if there exists a directed black path of length i to v, there must exist a path of length $i-1$ to one of v_j and a black edge from v_j to v. Let B_w^i be the event that there exists a directed black path of length i to vertex w. Using a union bound,

$$
\begin{aligned}
\Pr[B_v^i] &\leq \sum_{j=1}^{k} \Pr[B_{v_j}^{i-1}] \cdot \Pr[(v_j, v) \text{ is black}] \\
&\leq \sum_{j=1}^{k} \Pr[B_{v_j}^{i-1}] \cdot 10n^{\sigma_{l-1}-\sigma_l} && \text{(By Lemma 4.4.33)} \\
&\leq k \cdot n^{(i-1)(\sigma_{l-1}-\sigma_l)/2} \cdot 10n^{\sigma_{l-1}-\sigma_l} && \text{(Induction hypothesis)} \\
&\leq 50 \cdot \log n \cdot n^{(i+1)(\sigma_{l-1}-\sigma_l)/2} && (k \leq 5 \log n \text{ by Claim 4.4.21})
\end{aligned}
$$

$$\leq n^{i(\sigma_{l-1}-\sigma_l)/2},$$

which completes the induction step.

Second, we show that there is no directed black path of length $5/\sigma_l$ with high probability in the graph. To see this, the probability of having a directed black path of length $5/\sigma_l$ between two vertices u and v is upper bounded by $n^{5(\sigma_{l-1}-\sigma_l)/(2\sigma_l)}$. Taking a union bound over all possible pairs, we obtain

$$\begin{aligned}\Pr\left[\exists \text{ directed black path of length } 5/\sigma_l\right] &\leq n^2 \cdot n^{5(\sigma_{l-1}-\sigma_l)/(2\sigma_l)} \\ &\leq n^2 \cdot n^{-9/4} && \left(\text{Since } \sigma_{l-1} < \frac{\sigma_l}{10}\right) \\ &\leq n^{-1/4}.\end{aligned}$$

Therefore, we can assume that with high probability there is no directed black path of length $5/\sigma_l$ in the queried subgraph.

Finally, suppose that we condition on not having a directed black path of length $5/\sigma_l$. Since each vertex has at most $n^{\sigma_{l-1}}$ black edges in total (even not queried by the algorithm), the total number of vertices and edges that are reachable up to distance $5/\sigma_l$ from a fixed vertex u is upper bounded by $n^{5\sigma_{\ell-1}/\sigma_\ell}$. □

Corollary 4.4.37. *Consider all queried black edges in the core except edges E_ℓ^{inner}. The longest directed path of black edges has length at most $5/\sigma_\ell$.*

Proof. The proof follows by the proof of Lemma 4.4.36. □

It is important to observe that the algorithm discovers black incident edges for only a small fraction of vertices when compared to the total number of vertices. Additionally, if we exclude E_ℓ^{inner}, the size of the black descendants of each vertex is constrained as indicated in Lemma 4.4.36. Consequently, we anticipate a limited number of intersections between the descendants of vertices. This insight is further formalized in the following claims and corollary.

Let $SCC_1, SCC_2, \ldots, SCC_s$ be the strongly connected components of directed black edges that are queried by the algorithm. For each component such that its indegree is zero (roots of the directed acyclic graph of strongly connected components), we choose a vertex to represent the component. Let $R = \{u_1, u_2, \ldots, u_{s'}\}$ be the set of the chosen vertices. Note that each vertex $v \notin R$, is in a black descendent of at least one of the vertices in R.

Claim 4.4.38. *Consider all queried black edges in the core except edges E_ℓ^{inner}. Let $v \in R$. Then, the probability that there exists a vertex $u \in R \setminus \{v\}$ such that u'descendants intersect v's descendants is at most $O(n^{5\sigma_{\ell-1}/\sigma_\ell - \delta + \sigma_{\ell-1}})$.*

Proof. By Lemma 4.4.36, vertex v has at most $n^{5\sigma_{l-1}/\sigma_l}$ descendants. Combining with Claim 4.4.20, the probability that each new query goes to a vertex that is descendant of v is at most $O(n^{5\sigma_{l-1}/\sigma_l}/n)$. Since the total number of black edges is upper bounded by $O(n^{1-\delta+\sigma_{l-1}})$, then the probability that there exists a vertex $u \in R \setminus \{v\}$ such that u'descendants intersect descendants v's descendants is at most $O(n^{5\sigma_{\ell-1}/\sigma_\ell - \delta + \sigma_{\ell-1}})$ using a union bound. □

Corollary 4.4.39. *Consider all queried black edges in the core except edges E_ℓ^{inner}. Let $k < 50\sigma_l/\sigma_{l-1}$ and $v_1, \ldots, v_k$ be k arbitrary vertices in R. Then, the probability that there exists a vertex $u \in R \setminus \{v_1, \ldots, v_k\}$ such that u'descendants intersect descendants of vertices in $\{v_1, \ldots, v_k\}$ is at most $O(n^{5\sigma_{\ell-1}/\sigma_\ell - \delta + \sigma_{\ell-1}})$.*

Proof. The proof follows the same as proof of Claim 4.4.38 and the fact that k is a constant. □

Hence, we anticipate these black connected components to have a very small size, given that the number of descendants for each vertex is quite limited, and the chances of their intersection are low.

Claim 4.4.40. *Consider all queried black edges except edges E_ℓ^{inner}. Let C be an arbitrary connected component of these edges. Then, with high probability $|C| \leq O(n^{5\sigma_{\ell-1}/\sigma_\ell})$. Moreover, each connected component has at most $O(|C|)$ black edges.*

Proof. We construct a new graph H with the same vertex set R. We add edge (u, v) to H if black descendants of u and v intersect. In what follows, we prove that the largest connected component of H is at most $\varrho = 10/\delta$ with high probability. Since the number of black descendants (and black edges to its descendants) for each vertex is at most $n^{5\sigma_{l-1}/\sigma_l}$ by Lemma 4.4.36, this claim is enough to finish the proof.

Suppose that we start the following process from vertex $v \in R$. In the beginning, we have a set C that only contains v. In each step, we reveal one of the edges from vertices in C to vertices in $R \setminus C$. Assume that this edge is (w, z) where $w \in C$ and $z \in R \setminus C$. We add z to C and continue the process. The process stops either when there is no edge from C to $R \setminus C$ or when $|C| > \varrho$. Let X_i be the event that there exists an edge between C and $R \setminus C$ in step i of the process. By Corollary 4.4.39, we have $\Pr[X_i = 1 \mid X_1, \ldots, X_{i-1}] \leq O(n^{5\sigma_{\ell-1}/\sigma_\ell - \delta + \sigma_{\ell-1}})$. Let Y_v be the event that the process stops when $|C| > \varrho$. Hence,

$$\begin{aligned}
\Pr[Y_v] &= \prod_{i=1}^{i \leq \varrho} \Pr[X_i \mid X_1, X_2, \ldots, X_{i-1}] \\
&\leq O\left((n^{5\sigma_{\ell-1}/\sigma_\ell - \delta + \sigma_{\ell-1}})^\varrho\right) \\
&= O\left(\frac{1}{n^5}\right) && \text{(Since } \varrho = 10/\delta\text{).}
\end{aligned}$$

Therefore, using a union bound over all possible $v \in R$, with a probability of $1 - O(n^{-4})$, there is no connected component of size larger than ϱ in H. □

Corollary 4.4.41. *Consider all queried black edges except edges E_ℓ^{inner}. Let C be an arbitrarily connected component of these edges that is created by the intersection of descendants of ϱ vertices in R. Then, with high probability $\varrho \leq 10/\delta$.*

Proof. The proof follows by the proof of Claim 4.4.40. □

We now possess all the necessary tools to establish Lemma 4.4.34. At a high level, we assume that the algorithm has control over where to put the edges of E_ℓ^{inner} to maximize $\sum_{i=1}^{c} |C_i|$. However, we show that even by giving this power to the algorithm, we can still prove the bound stated in the lemma.

Proof of Lemma 4.4.34. Suppose that an adversary chooses how edges of E_ℓ^{inner} are between the components. Let $\widehat{\mathcal{C}} = \{\widehat{C_1}, \widehat{C_2}, \ldots, \widehat{C_{c'}}\}$ be the connected components before adding edges E_ℓ^{inner} by the adversary. First, note that $|\widehat{C_i}| < O(n^{5\sigma_{\ell-1}/\sigma_\ell})$ for all $i \in [c']$ with high probability by Claim 4.4.40.

Let $C_1, C_2, \ldots, C_c$ be the connected components after adding edges E_ℓ^{inner} and removing the components that do not have any of the edges in E_ℓ^{inner}. Each edge of E_ℓ^{inner} can connect at most two components of $\widehat{\mathcal{C}}$. Therefore, the total number of components in $\widehat{\mathcal{C}}$ that have at least one edge of E_ℓ^{inner} is upper bounded by $O(|E_\ell^{inner}|)$. Now if $1 - g(\ell) < 0$, according to the statement (i) of Lemma 4.4.33, with probability $1 - O(n^{1-g(\ell)})$ we have $|E_\ell^{inner}| = 0$. Also, with a high probability $|E_\ell^{inner}| = \widetilde{O}(1)$. Combining with $|\widehat{C_i}| < O(n^{5\sigma_{\ell-1}/\sigma_\ell})$, we obtain the proof of statement (i).

If $1 - g(\ell) > 0$, according to the statement (ii) of Lemma 4.4.33, we have $|E_\ell^{inner}| \leq O(n^{1-g(\ell)})$ with high probability. Combining with $|\widehat{C_i}| < O(n^{5\sigma_{\ell-1}/\sigma_\ell})$, we obtain $\sum_{i=1}^{c} |C_i| \leq O(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell})$ which concludes the proof of (ii). □

4.4.6 Smaller Connected Components Results in Less Identified Inner Edges

In this section, we use Lemma 4.4.34 to show that as the size of connected components gets smaller, it is harder for the algorithm to identify black edges. We abuse the notation to generalize the definition of spoiled vertex and shallow subgraph similar to the warm-up section.

Definition 4.4.42 (ℓ-Shallow Subgraph)**.** *Suppose that we define green and black edges with respect to level ℓ and $\ell-1$ of the construction hierarchy. For a vertex v, we let the ℓ-shallow subgraph of v be a set of vertices that are reachable by v within a distance of $10 \log n$ using directed paths with only black edges from v in the queried subgraph. We use $T^\ell(v)$ to denote the ℓ-shallow subgraph of v.*

With the exact same proof as Corollary 4.4.24, we can extend its claim to ℓ-Shallow Subgraph.

Lemma 4.4.43. *With high probability, each vertex is in at most $\widetilde{O}(1)$ ℓ-shallow subgraphs.*

Corollary 4.4.44. *With high probability, each black edge that the algorithm finds is in at most $\widetilde{O}(1)$ ℓ-shallow subgraphs.*

Observation 4.4.45. *Let $C_1, C_2, \ldots, C_c$ be the underlying undirected connected components of black edges, and let $E(C_i)$ be the edges set of component C_i. Then, $|E(C_i)| \leq O(\log n) \cdot |C_i|$.*

Proof. The proof follows by the fact that each vertex has an incoming degree of at most $5 \log n$ in the whole queried subgraph of core by Claim 4.4.21. □

Observation 4.4.46. *Let $C_1, C_2, \ldots, C_c$ be the underlying undirected connected components of black edges where there exists at least one edge of E_ℓ^{inner} in each of the components. Let $E(C_i)$ denote the edge set of component C_i. Then, the following statements hold:*

(i) If $1-g(\ell) < 0$, then $c = 0$ with probability $1-O(n^{1-g(\ell)})$. Also, with a high probability, $\sum_{i=1}^{c} |E(C_i)| \leq \widetilde{O}(n^{5\sigma_{\ell-1}/\sigma_\ell})$

(ii) If $1 - g(\ell) > 0$, we have $\sum_{i=1}^{c} |E(C_i)| \leq \widetilde{O}(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell})$ with high probability.

Proof. Combining each statement of Lemma 4.4.34 and Observation 4.4.45 yields each statement. □

Definition 4.4.47 (ℓ-Spoiler Vertex)**.** *For $\ell \in (1, L]$, let $\hat{E}$ be the set of black edges that are in a connected component with at least one edge of E_ℓ^{inner}. Let u be a vertex that is in a black connected component that contains at least one edge of E_ℓ^{inner}. We say a vertex u in the core is* ℓ-spoiler *if at least one of the following conditions holds:*

(i) vertex u has more than one incoming edge,

(ii) there is an edge $(u, v) \in \hat{E}$ that is discovered by the algorithm at a time when v already has non-zero degree.

Definition 4.4.48 (ℓ-Spoiled Vertex)**.** *For $\ell \in (1, L]$, let v be a vertex that is in a black connected component that contains at least one edge of E_ℓ^{inner}. Then, vertex v is ℓ-spoiled if its ℓ-shallow subgraph contains any of the following:*

- *a ℓ-spoiler vertex; or*
- *at least $n^{\delta-2\sigma_L}$ vertices.*

Observation 4.4.49. *Let v be a vertex that is not ℓ-spoiled. Then, the ℓ-shallow subgraph of v is a rooted tree of size at most $n^{\delta-2\sigma_L}$. Moreover, for each edge (u, w) in the ℓ-shallow subgraph of v, at the time that the algorithm made the query, w was a singleton vertex.*

Proof. Because of Definition 4.4.47 and Definition 4.4.48, when the algorithm finds an edge (u, w) in the ℓ-shallow subgraph of v, the other endpoint must be singleton which implies that the ℓ-shallow subgraph of v is a rooted tree. □

In the next two claims, we provide a bound on probability and the number of vertices that do not satisfy the second condition in Definition 4.4.48.

Claim 4.4.50. *Suppose that $1-g(\ell-1)-3\sigma_{\ell-1} \geq 0$. With high probability, there are at most $O(n^{1-g(\ell-1)-2\sigma_{\ell-1}})$ vertices v where:*

- *there exist an edge of E_ℓ^{inner} in their black connected component; and*
- *$|T^\ell(v)| > n^{\delta-2\sigma_{\ell-1}}$.*

Proof. Let $C_1, C_2, \ldots, C_c$ be the underlying undirected connected components of black edges where there exists at least one edge of E_ℓ^{inner} in each of the components. Also, let $\widehat{V}$ be the set of vertices in these components. Let $E(C_i)$ be the set of edges of component i. By statement (ii) of Observation 4.4.46, we have $\sum_{i=1}^{c} |E(C_i)| \leq \widetilde{O}(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell})$. Applying Corollary 4.4.44, we obtain

$$\sum_{u\in\widehat{V}} |T^\ell(u)| \leq \widetilde{O}(1) \cdot \sum_{i=1}^{c} |E(C_i)| \leq \widetilde{O}(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell}).$$

Let ϱ denote the number of vertices v where $|T^\ell(v)| > n^{\delta-\sigma_{\ell-1}}$. Therefore,

$$\varrho \leq \frac{\sum_{u\in\widehat{V}} |T^\ell(u)|}{n^{\delta-2\sigma_{\ell-1}}} \leq \frac{\widetilde{O}(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell})}{n^{\delta-2\sigma_{\ell-1}}} \leq \widetilde{O}(n^{1-g(\ell)-\delta+5\sigma_{\ell-1}/\sigma_\ell+2\sigma_{\ell-1}}) \leq O(n^{1-g(\ell-1)-2\sigma_{\ell-1}})$$

where the last inequality is followed by statement (ii) of Observation 4.4.32. □

Claim 4.4.51. *Suppose that $1-g(\ell-1)-3\sigma_{\ell-1} < 0$. With high probability, there exists no vertex such that*

- *there exist an edge of E_ℓ^{inner} in their black connected component; and*
- *$|T^\ell(v)| > n^{\delta-2\sigma_{\ell-1}}$.*

Proof. Let $C_1, C_2, \ldots, C_c$ be the underlying undirected connected components of black edges where there exists at least one edge of E_ℓ^{inner} in each of the components. Also, let $\widehat{V}$ be the set of vertices in these components. First, if $1-g(\ell) < 0$, with high probability we have $\sum_{i=1}^{c} |E(C_i)| \leq \widetilde{O}(n^{5\sigma_{\ell-1}/\sigma_\ell})$ by statement (i) of Observation 4.4.46. Since $\delta - 2\sigma_{\ell-1} > 5\sigma_{\ell-1}/\sigma_\ell$, there is no component with an edge from E_ℓ^{inner} with size $n^{\delta-2\sigma_{\ell-1}}$ with high probability.

Next, if $1-g(\ell) > 0$, with high probability, we have $\sum_{i=1}^{c} |E(C_i)| \leq \widetilde{O}(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell})$ by statement (ii) of Observation 4.4.46. Applying Corollary 4.4.44, we obtain

$$\sum_{u\in\widehat{V}} |T^\ell(u)| \leq \widetilde{O}(1) \cdot \sum_{i=1}^{c} |E(C_i)| \leq \widetilde{O}(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell}).$$

Moreover,

$$\begin{aligned} 1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell &= 1-g(\ell-1)+\delta-5\sigma_{\ell-1} && \text{(By statement (i) of Observation 4.4.32)} \\ &= (1-g(\ell-1)-3\sigma_{\ell-1}) + (\delta-2\sigma_{\ell-1}) \\ &< \delta-\sigma_{\ell-1}, \end{aligned}$$

where the last inequality follows by the assumption that $1-g(\ell-1)-3\sigma_{\ell-1}/\sigma_\ell < 0$. Therefore, with a high probability, there is no component with an edge from E_ℓ^{inner} with size $n^{\delta-\sigma_{\ell-1}}$ with high probability. □

Just as in Lemma 4.4.28, we can give bounds on the probability and the count of ℓ-spoiled vertices. While the proof steps closely resemble those in the warm-up section, for the sake of thoroughness, we reiterate some of the key arguments.

Lemma 4.4.52. *Suppose that* $1-g(\ell-1)-3\sigma_{\ell-1} \geq 0$. *With high probability, there are at most* $O(n^{1-g(\ell-1)-2\sigma_{\ell-1}})$ *ℓ-spoiled vertices.*

Proof. The proof has the same steps as Lemma 4.4.28. First, by Claim 4.4.50, with high probability there are at most $O(n^{1-g(\ell-1)-2\sigma_{\ell-1}})$ vertices v in a component with at least one edge of E_ℓ^{inner} such that $|T^\ell(v)| > n^{\delta-\sigma_{\ell-1}}$. Let $C_1, C_2, \ldots, C_c$ be the underlying undirected connected components of black edges where there exists at least one edge of E_ℓ^{inner} in each of the components. Let $\widehat{E}$ be the set of black edges of these components. By statement (ii) of Lemma 4.4.34, $|\widehat{E}| \leq O(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell})$.

Next, suppose that we add edges $\widehat{E}$ that are queried by the algorithm in the same order as the algorithm queried them. We show that with high probability, there exists at most $O(n^{1-g(\ell-1)-4\sigma_{\ell-1}})$ ℓ-spoiler vertices in the graph. By Lemma 4.4.43, since each vertex is in at most $\widetilde{O}(1)$ ℓ-shallow subgraphs, then there are at most $O(n^{1-g(\ell-1)-3\sigma_{\ell-1}})$ ℓ-spoiled vertices. So in the rest, we focus on upper bounding the number of ℓ-spoiler vertices.

At the time that we add an edge (u, v), the probability that v has at least one black edge is $O(n^{\sigma_{\ell-1}-\delta})$ since by Claim 4.4.35, there are at most $O(n^{1-\delta+\sigma_{\ell-1}})$ vertices with a black edge and by Claim 4.4.20, each of them has a probability of $O(1/n)$ to be the queried edge of u. For such an edge, condition (ii) holds for vertex u and condition (i) holds for vertex v. We assume that during the process of adding edges, for such an edge we count two spoiler vertices (for both endpoints).

Let X_i be the indicator of having a new spoiler vertex after adding ith edge. By the discussion above,

we have $\Pr[X_i = 1] \leq O(n^{\sigma_{\ell-1}-\delta})$. Let $X = \sum_{i=1}^{|\widehat{E}|} X_i$. Thus,

$$\mathbf{E}[X] \leq O(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell-\delta+\sigma_{\ell-1}}),$$

since $|\widehat{E}| \leq O(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell})$. Since events are negatively correlated, we get

$$\Pr\left[|X - \mathbf{E}[X]| \geq 6\sqrt{\mathbf{E}[X]\log n}\right] \leq 2\exp\left(-\frac{(6\sqrt{\mathbf{E}[X]\log n})^2}{3\,\mathbf{E}[X]}\right) \leq \frac{1}{n^{10}},$$

which implies that there are at most $O(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell-\delta+\sigma_{\ell-1}})$ different i such that $X_i = 1$. For each edge, if the indicator is one, we count a constant number of spoiler vertices. Moreover, by statement (i) of Observation 4.4.32,

$$1 - g(\ell) + 5\sigma_{\ell-1}/\sigma_\ell - \delta + \sigma_{\ell-1} = 1 - g(\ell-1) - 4\sigma_{\ell-1},$$

which concludes the proof. □

Lemma 4.4.53. *Suppose that $1 - g(\ell-1) - 3\sigma_{\ell-1} < 0$. Then, with probability of $1 - O(n^{1-g(\ell-1)-3\sigma_{\ell-1}})$, there is no ℓ-spoiled vertex. Moreover, with a high probability, there are at most $\widetilde{O}(1)$ ℓ-spoiled vertices.*

Proof. Because of the assumption that $1-g(\ell-1)-3\sigma_{\ell-1} < 0$, with high probability there is no vertex v in a component with at least one edge of E_ℓ^{inner} such that $|T^\ell(v)| > n^{\delta-2\sigma_{\ell-1}}$ by Claim 4.4.51. Let $C_1, C_2, \ldots, C_c$ be the underlying undirected connected components of black edges where there exists at least one edge of E_ℓ^{inner} in each of the components. Let $\widehat{E}$ be the set of black edges of these components. We consider two possible scenarios:

(Case 1) $1-g(\ell) > 0$**:** in this case, we have $\sum_{i=1}^{c} |C_i| \leq \widetilde{O}(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell})$ with high probability according to statement (ii) of Observation 4.4.46. We prove that with probability $1 - O(n^{1-g(\ell-1)-3\sigma_{\ell-1}})$, there exists no ℓ-spoiler vertex. Suppose that we add edges of $\widehat{E}$ according to the ordering that the algorithm queried them. With the exact same argument as proof of Lemma 4.4.52, each edge that we add has a probability of $O(n^{\sigma_{\ell-1}-\delta})$ to create a constant number of ℓ-spoiler vertices. Using a union bound, the probability of having a ℓ-spoiler vertex is bounded by

$$|\widehat{E}| \cdot O(n^{\sigma_{\ell-1}-\delta}) \leq \widetilde{O}(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell+\sigma_{\ell-1}-\delta}) \leq \widetilde{O}(n^{1-g(\ell-1)-4\sigma_{\ell-1}}),$$

where the last inequality is followed by statement (i) of Observation 4.4.32. On the other hand, since the expected number of ℓ-spoiler vertices is less than 1, using a Chernoff bound we can show that with high probability there are at most $\widetilde{O}(1)$ ℓ-spoiler vertices.

(Case 2) $1 - g(\ell) < 0$**:** in this case, according to the statement (i) of Observation 4.4.46, there is no component with an edge of E_ℓ^{inner} with probability $1 - O(n^{1-g(\ell)})$ and therefore, there is no ℓ-spoiled vertex with probability of $O(n^{1-g(\ell)})$. Now suppose that we condition on having a component with an edge of E_ℓ^{inner}. By statement (i) of Observation 4.4.46, we have $|\widehat{E}| \leq \widetilde{O}(n^{5\sigma_{\ell-1}/\sigma_\ell})$ with high probability. Similar

to the previous case, the probability of having a ℓ-spoiler vertex is bounded by

$$|\widehat{E}| \cdot O(n^{\sigma_{\ell-1}-\delta}) \leq \widetilde{O}(n^{5\sigma_{\ell-1}/\sigma_\ell+\sigma_{\ell-1}-\delta}).$$

Since the probability of having a component with an edge of E_ℓ^{inner} is $O(n^{1-g(\ell)})$, the probability of having a ℓ-spoiler vertex is upper bounded by

$$O(n^{1-g(\ell)}) \cdot \widetilde{O}(n^{5\sigma_{\ell-1}/\sigma_\ell+\sigma_{\ell-1}-\delta}) \leq \widetilde{O}(n^{1-g(\ell-1)-4\sigma_{\ell-1}}).$$

Therefore, the probability of having a ℓ-spoiled vertex is at most $O(n^{1-g(\ell-1)-3\sigma_{\ell-1}})$. On the other hand, since the expected number of ℓ-spoiler vertices is less than 1, using a Chernoff bound we can show that with high probability there are at most $\widetilde{O}(1)$ ℓ-spoiler vertices. □

Claim 4.4.54. *Let $C'_1, C'_2, \ldots, C'_{c'}$ be the connected components of black edges that do not contain any edge of E_ℓ^{inner}. Then, with probability $1 - O(n^{-\delta+\sigma_{\ell-1}+10\sigma_{\ell-1}/\sigma_\ell})$ all components are trees.*

Proof. By Claim 4.4.40, with high probability $|C'_i| \leq O(n^{5\sigma_{\ell-1}/\sigma_\ell})$ for all $i \in [c']$. Also, $c' \leq O(n^{1-\delta+\sigma_{\ell-1}})$ since the total number of black edges is $O(n^{1-\delta+\sigma_{\ell-1}})$ by Claim 4.4.35. Hence, $\sum_{i=1}^{c'} |C'_i|^2 \leq O(n^{1-\delta+2\sigma_{\ell-1}+10\sigma_{\ell-1}/\sigma_\ell})$.

Suppose that we condition on high probability event that $\sum_{i=1}^{c'} |C'_i|^2 \leq O(n^{1-\delta+2\sigma_{\ell-1}+10\sigma_{\ell-1}/\sigma_\ell})$. We add the edges of these components one by one with respect to the ordering that the algorithm queried. When the algorithm queries the adjacency list of vertex v that is in a component of size x, the probability that the resulting queried edge goes to the same component is $O(x/n)$ by Claim 4.4.20. Therefore, if the edge is in the final connected component C'_i, this probability is upper bounded by $O(|C'_i|/n)$. Combining with the fact that each component has $|C'_i|$ edges, the probability of having a cycle is at most $\sum_{i=1}^{c'} O(|C'_i|^2/n) = O(n^{-\delta+\sigma_{\ell-1}+10\sigma_{\ell-1}/\sigma_\ell})$. □

For the rest, we condition on the event that each connected component of black edges that do not contain any edge of E_ℓ^{inner} is a tree. By Claim 4.4.54, the failure probability of this event is $O(n^{-\delta+\sigma_{\ell-1}+10\sigma_{\ell-1}/\sigma_\ell}) = o(1)$. Moreover, since the number of levels in our hierarchy construction is a constant, these events hold for all levels with probability $1 - o(1)$.

Lemma 4.4.55. *Let (u, v) be a directed black edge in the connected component C such that there is no edge of E_ℓ^{inner} in C. Also, suppose that $u \in A_r$ and v belongs to $\{A_r, B_r, D_r\}$ in level $\ell - 1$ of the hierarchy. Let $\overline{C}$ be the component that v belongs to after removing edge (u, v). Let $\mathcal{L}(v)$ and $\mathcal{L}'(v)$ be an arbitrary label for v from $\{A_r, B_r, D_r\}$ and the entire queried subgraph of the black edges excluding $\overline{C}$. Then, we have*

$$\Pr[\overline{C} \mid \mathcal{L}(v)] \leq \left(1 + O(n^{\sigma_{\ell-1}-\delta})\right)^{|\overline{C}|} \cdot \Pr[\overline{C} \mid \mathcal{L}'(v)].$$

We defer the proof of the above lemma to Section 4.4.8.

Lemma 4.4.56. *Let v be a vertex that is not ℓ-spoiled and it belongs to a connected component with at least one edge of E_ℓ^{inner}. Also, suppose that v belongs to $\{A_r, B_r, D_r\}$ in level $\ell - 1$ of the hierarchy. Let $\mathcal{L}(v)$ and $\mathcal{L}'(v)$ be an arbitrary label for v from $\{A_r, B_r, D_r\}$ and the entire queried subgraph of the black edges excluding the ℓ-shallow subgraph of v. Then, we have*

$$\Pr[T^\ell(v) \mid \mathcal{L}(v)] \leq \left(1 + O(n^{\sigma_{\ell-1}-\delta})\right)^{|T^\ell(v)|} \cdot \Pr[T^\ell(v) \mid \mathcal{L}'(v)].$$

We defer the proof of the above lemma to Section 4.4.7. Now we are ready to complete the proof of Lemma 4.4.33.

Proof of Lemma 4.4.33. As we discussed before, we need to prove Lemma 4.4.33 for $\ell-1$ using Lemma 4.4.34 for ℓ. Thus, $\ell > 1$. In this proof, when we use the label A_r or B_r, we mean the A_r and B_r in level $\ell-1$ of the hierarchy. The first part of the proof is similar to the proof of Lemma 4.4.30. Let $\widetilde{E}$ be the set of black edges (u, v) (directed from u to v) such that $u \in A_r$ that satisfy at least one the following conditions:

(i) v is a ℓ-spoiled vertex; or

(ii) u has at least $n^{\sigma_{\ell-1}}/3$ ℓ-spoiled neighbors in the queried subgraph.

We begin by proving that for all black edges $e \notin \widetilde{E}$, we have that $p_e^{(\ell-1)-inner} \leq 10n^{\sigma_{\ell-2}-\sigma_{\ell-1}}$. Then, we give an upper bound on $|\widetilde{E}|$ with a case distinction based on the value of $g(\ell-1)$.

Consider edge $e = (u, v)$ (directed from u to v) where $e \notin \widetilde{E}$. If $u \notin A_r$, then the bound $p_e^{(\ell-1)-inner} = 0 \leq 10n^{\sigma_{\ell-2}-\sigma_{\ell-1}}$ trivially holds. So let us assume that $u \in A_r$. Let $v_0 = v, v_1, v_2, \ldots, v_k$ be the neighbors of u that are adjacent to u with a black edge in the queried subgraph such that $v_i \in A_r \cup B_r$ and either v_i is a singleton vertex in the queried subgraph black edges or v_i is the directed child of u that is not spoiled. Note that $k \geq n^{\sigma_{\ell-1}}/2$ By condition (ii). We bound the probability that $v_0 \in A_r$ using a coupling argument.

Consider a labeling profile $\mathcal{P}$ of all vertices $U = \{v_0, v_1, \ldots, v_k\}$ such that $\mathcal{P}(v_0) = A_r$. By the construction of our input distribution, since $u \in A_r$, at most $O(d_{\ell-2}) = O(n^{\sigma_{\ell-2}})$ vertices of U are in A_r. We produce $\Omega(n^{\sigma_{\ell-1}})$ new profiles $\mathcal{P}'$ such that $\mathcal{P}'(v_0) \neq A_r$. For each vertex v_i in U such that $\mathcal{P}(v_i) = B_r$, we construct a new profile $\mathcal{P}'$ where $\mathcal{P}(v_j) = \mathcal{P}'(v_j)$ for $j \notin \{0, i\}$, $\mathcal{P}'(v_i) = A_r$, and $\mathcal{P}'(v_0) = B_r$. Since v_0 and v_i are not a ℓ-spoiled vertex, they are either in a component with no edge of E_ℓ^{inner} or their ℓ-shallow subgraph satisfies the conditions in Definition 4.4.48. In both cases, the probability of querying the same shallow subgraph or connected component in the new labeling profile will be the same up to a factor of

$$\left(1 + O(n^{\sigma_L - \delta})\right)^{n^{\delta - 2\sigma_{\ell-1}}},$$

by Lemma 4.4.56 and Lemma 4.4.55 since either $|T^\ell(v_0)| \leq n^{\delta-2\sigma_{\ell-1}}$ (resp. $|T^\ell(v_i)| \leq n^{\delta-2\sigma_{\ell-1}}$) or the component that v_0 or v_i belongs to has size of at most $O(n^{5\sigma_{\ell-1}/\sigma_\ell}) \leq O(n^{\delta-2\sigma_{\ell-1}})$. Therefore, the probability of having profile $\mathcal{P}$ and $\mathcal{P}'$ are the same up to a factor $1 + o(1)$. We construct a bipartite graph $H = (P_1, P_2, E_P)$ of labeling profiles such that in P_1, we have all profiles $\mathcal{P}$ where $\mathcal{P}(v_0) = A_r$, and in the P_2, all profiles $\mathcal{P}'$ where $\mathcal{P}'(v_0) = B_r$. We add an edge between two profiles $\mathcal{P}$ and $\mathcal{P}'$ if we can convert $\mathcal{P}$ to $\mathcal{P}'$ according to the above process. Therefore, $\deg_H(\mathcal{P}) \geq k/2 \geq n^{\sigma_{\ell-1}}/4$ for $\mathcal{P} \in P_1$ since at least $k/2$ vertices of U belong to B_r. On the other hand, $\deg_H(\mathcal{P}') \leq 2n^{\sigma_{\ell-2}}$ for $\mathcal{P}' \in P_2$. To see this, there are at most $2d_{\ell-2} = 2n^{\sigma_{\ell-2}}$ vertices v_i in U such that $\mathcal{P}'(v_i) = A_r$ according to the construction of input distribution. Hence,

$$\begin{aligned}
p_e^{\ell-inner} &\leq (1 + o(1)) \cdot \frac{|P_1|}{|P_2|} \\
&\leq (1 + o(1)) \cdot \frac{2n^{\sigma_{\ell-2}}}{n^{\sigma_{\ell-1}}/4} \\
&\leq (1 + o(1)) \cdot 8n^{\sigma_{\ell-2}-\sigma_{\ell-1}} \leq 10n^{\sigma_{\ell-2}-\sigma_{\ell-1}}.
\end{aligned}$$

Therefore, for all black edges $e \notin \widetilde{E}$, we have that $p_e^{\ell-inner} \leq 10n^{\sigma_{\ell-2}-\sigma_{\ell-1}}$. Now it remains to give an upper bound for $|\widetilde{E}|$. We prove this part using case distinction:

(Case 1) $1 - g(\ell-1) - 3\sigma_{\ell-1} \geq 0$**:** first note that in this case $1 - g(\ell-1) > 0$, so we are in statement (ii) of Lemma 4.4.33. By Lemma 4.4.52, with high probability there are at most $O(n^{1-g(\ell-1)-2\sigma_{\ell-1}})$ ℓ-spoiled vertices since $1 - g(\ell-1) - 3\sigma_{\ell-1} \geq 0$. Further, each vertex has at most $\widetilde{O}(1)$ indegree which implies that there are at most $O(n^{1-g(\ell-1)-\sigma_{\ell-1}})$ edges that satisfy condition (i). Now suppose that a vertex u satisfies the condition (ii). Then, u must have at least $n^{\sigma_{\ell-1}}$ edges (u, w) (directed from u to w) such that w is ℓ-spoiled since each vertex has at most $\widetilde{O}(1)$ indegree. Thus, the total number of vertices that satisfy condition (ii) is at most $\widetilde{O}(1) \cdot O(n^{1-g(\ell-1)-2\sigma_{\ell-1}}) \leq O(n^{1-g(\ell-1)-\sigma_{\ell-1}})$. Therefore, we have $|\widetilde{E}| \leq O(n^{1-g(\ell-1)-\sigma_{\ell-1}}) \leq O(n^{1-g(\ell-1)})$ with high probability.

(Case 2) $1 - g(\ell-1) - 3\sigma_{\ell-1} < 0$ **and** $1 - g(\ell-1) > 0$**:** in this case, since $1 - g(\ell-1) - 3\sigma_{\ell-1} < 0$, by Lemma 4.4.53, with high probability there are at most $\widetilde{O}(1)$ ℓ-spoiled vertices which implies that $|\widetilde{E}| \leq \widetilde{O}(1)$ with the same argument as case 1. Therefore, with high probability $|\widetilde{E}| \leq O(n^{1-g(\ell-1)})$.

(Case 3) $1 - g(\ell-1) - 3\sigma_{\ell-1} < 0$ **and** $1 - g(\ell-1) < 0$**:** in this case, since $1 - g(\ell-1) - 3\sigma_{\ell-1} < 0$, by Lemma 4.4.53, with probability of $1 - O(n^{1-g(\ell-1)-3\sigma_{\ell-1}}) \geq 1 - O(n^{1-g(\ell-1)})$, there is no ℓ-spoiled vertex which implies that $|\widetilde{E}| = 0$. Moreover, with high probability, there are at most $\widetilde{O}(1)$ ℓ-spoiled vertices which implies that $|\widetilde{E}| = \widetilde{O}(1)$. □

4.4.7 Proof of Lemma 6.15 and Lemma 6.42

In this section, we show our approach to proving Lemma 4.4.29 and Lemma 4.4.56. Our proof draws inspiration from the findings of [41]. The way in which we construct each level of our input distribution closely resembles the hard example presented in this proof. The key distinction lies in how we put edges between different subsets of vertices. In their construction, they make an assumption that the degrees follow a binomial distribution. This assumption is beneficial because with each query the algorithm makes to a vertex's adjacency list, the neighbor's label becomes independent of the labels of the previously discovered neighbors. However, in order to maintain the condition of binomial degrees, they require a minimum of $O(\sqrt{n})$ *bad vertices*, where the neighbor distribution deviates from the expectation. In their model, the total number of queries is significantly fewer than $O(\sqrt{n})$, allowing them to condition their process on not encountering any bad vertices. In contrast, in our setting, the algorithm can find one edge of at least $O(n^{1-\delta+\sigma_L})$ vertices which is much larger than $O(\sqrt{n})$. Therefore, we cannot expect not to see a bad vertex. So we slightly change their construction and use exact degrees between the subsets of vertices instead of binomial distribution. We will prove that the same result also holds in this construction. For now, suppose that we have a fixed level ℓ in our hierarchy. First, we introduced relevant notations and provided essential tools required to accomplish the final goal of this section. To provide a comprehensive overview, we reiterate certain definitions and claims as mentioned in [41]. For the rest of the section, assume that $d = d_\ell/d_{\ell-1} = \Theta(n^{\sigma_\ell-\sigma_{\ell-1}})$.

Definition 4.4.57 (Special Edge)**.** *[Similar to Definition 6.1 of [41]] We say an edge (u, v) is special if one of the following statements holds:*

- *$u \in S$ and $v \in B_1$, or $u \in B_1$ and $v \in S$,*

- *$u \in B_i$ and $v \in A_{i-1}$, or $u \in A_{i-1}$ and $v \in B_i$ for $i \in (1, r]$,*
- *edges that only exist in $\mathcal{D}^{\ell}_{\mathsf{YES}}$ or $\mathcal{D}^{\ell}_{\mathsf{NO}}$,*
- *edges between D_i^j and D_i^{j+1} for $j \in \{1, 3\}$ (for the base level we consider a perfect matching inside each D_i).*

Definition 4.4.58 (Mixer Vertex)**.** *[Similar to Definition 6.2 of [41]] Let T be a rooted tree and u be its root. Also, assume that $u \in \{A_r, B_r, D_r\}$. Let v be a vertex in T and suppose that there are k special edges on the path between u and v. If $k < r - 1$, we say v is a mixer vertex if and only if $v \in \bigcup_{i=1}^{r-k-1} D_i$.*

The following observation is a direct consequence of Definition 4.4.57, Definition 4.4.58, and the way the input distribution is constructed.

Observation 4.4.59. *Let T be a rooted tree where $u \in \{A_r, B_r, D_r\}$. Each path from u to an S vertex that does not contain a mixer vertex has at least $r - 1$ special edges.*

Lemma 4.4.60. *Let T be a rooted tree that is queried by the algorithm. Also, suppose that the root of the tree is in $\{A_r, B_r, D_r\}$. Then, with probability at least $1 - O(|V(T)|/d^{r-1})$, every path that starts from the root to an arbitrary vertex in the tree and does not contain a mixer vertex, must have at most $r - 2$ special edges.*

Proof. We prove that each path the algorithm finds to a vertex that contains at least $r - 1$ special edges does not have a mixer vertex with probability $O(1/d^{r-1})$. For a mixer vertex in D_i we use i to denote the index of the mixer vertex. Suppose that there exists an oracle that each time the algorithm finds a path with at least $r - 1$ special edges, it either returns that the path does not contain any mixer vertex or reveals the mixer vertex with the lowest index on the path.

Consider the first path that the algorithm finds with $r - 1$ special edges. Consider the first time that the algorithm finds $r - 2$ special edges on this path. Also, suppose that by this time, the path does not contain any D_1 vertex. Hence, by Observation 4.4.59, the path has not reached any vertex in layer 1 at this time. At this time, when the algorithm queries the next edge, the probability of seeing a special edge is $O(1/d)$ according to the construction. However, the probability of querying a vertex that is in D_1 is $\Theta(1)$. Therefore, the probability of the path going through the next special edge is $O(1/d)$ before stepping on a mixer vertex with index 1. The crucial difference between our construction and the construction in [41] appears here when for a fixed vertex v if the oracle reveals a lot of mixer vertices that are direct children of v. Then, the probability of seeing a mixer vertex of level 1 when the algorithm queries the adjacency list of v is not $\Omega(1)$ anymore. To deal with this issue, we give more power to the oracle. We assume that for vertex v, if the oracle revealed half of the mixer vertices of a fixed index i that are direct children of v, the oracle reveals a path from v downward to a vertex w that does not contain a mixer with index $[1, i]$ and consequently it reveals the mixer of the path from the root to w which must have an index larger than i. In the case that $i > r - 2$, the oracle returns a path that does not contain any mixer vertex from the root, and the process terminates.

With this modification, although the oracle gives more information, still we can get relatively the same result. Using the above argument, $O(1/d)$ fraction of paths do not cross a mixer vertex with index 1. Also, when the algorithm finds $\Omega(d)$ direct children of a vertex that are mixer vertices with index 1, the oracle

gives away a path without having an index 1 mixer. Hence, the ratio of paths that the algorithm finds that do not contain a mixer vertex of index 1 is $O(1/d)$ fraction of all paths. Also, it is important to observe that when a mixer vertex w is revealed by the oracle, all queries below that mixer vertex are pointless since the highest index mixer that will be revealed by the oracle for paths that cross w is going to be w.

Now consider all paths that do not contain a mixer vertex of index 1. With a similar argument, $O(1/d)$ fraction of these paths does not cross a mixer with index 2. To see this, the probability of crossing $(r-2)$-th special edge before going through a mixer with index 2 is $O(1/d)$. Therefore, $O(1/d^2)$ fraction of paths does not go through a mixer of index 2 or below. Similarly, the probability of having a path that does not cross any mixer vertex with an index of at most i is $O(1/d^i)$. Therefore, the probability of having a path that does not contain any mixer vertex is $O(1/d^{r-1})$. Since the total number of paths from the root is at most $O(|V(T)|)$, with a probability of $1-O(|V(T)|/d^{r-1})$ all paths that have more than $r-2$ special edges contain a mixer vertex. □

Note that the failure probability of the above event is very small. To see this, first, we have that $|V(T)| = O(n)$. Moreover, we have

$$\begin{aligned}
d^{r-1} \geq \Omega\left((n^{\sigma_\ell - \sigma_{\ell-1}})^{3r/4}\right) &\geq \Omega\left((n^{2\sigma_\ell/3})^{3r/4}\right) && \text{(Since } \sigma_\ell \geq (10/\delta)\cdot\sigma_{\ell-1}\text{)}\\
&= \Omega\left(n^{r\sigma_1/2}\right) && \text{(Since } \ell \geq 1 \text{ and } \sigma_i \geq \sigma_{i-1}\text{)}\\
&= \Omega\left(n^{5/\delta}\right) && \text{(Since } r\sigma_1 = 10/\delta\text{)}\\
&= \Omega(n^5) && \text{(Since } \delta \leq 1\text{).}
\end{aligned}$$

Therefore, the failure probability is $O(n^{-4})$, and using union bound, we can condition on the event that for all vertices that are not spoiled, the condition above holds. Now that we have this property, the exact same coupling as [41] works here since the number of neighbors of each subset of vertices is similar to the transition probabilities in their construction. We restate the lemma in terms of our parameter for both Lemma 4.4.29 and Lemma 4.4.56.

Lemma 4.4.61 (Similar to the Coupling Lemma in [41]. See Lemma 6.7 of the Arxiv version.)**.** *Let T be a rooted tree that is queried by the algorithm where the root of the tree is in $\{A_r, B_r, D_r\}$. Also, suppose we condition on the event in Lemma 4.4.60. Then, the probability of seeing the same tree is equal for all possible roots in $\{A_r, B_r, D_r\}$ up to $(1 + o(n^{2\delta - 3\sigma_L - 1}))^{|T|}$ multiplicative factor.*

Proof of Lemma 4.4.29. First, by Observation 4.4.27, since v is not a spoiled vertex, the shallow subgraph of v is a rooted tree. Note that we condition on the labels of all vertices except the vertices in the shallow subgraph of vertex v. However, in the coupling in Lemma 4.4.61, there is no conditioning on labels of vertices. Since the total number of the vertices that we are conditioning on their label is $O(n^{1-\delta+\sigma_L})$, the shift in probability of each step of the coupling in Lemma 4.4.61 is at most $O(n^{1-\delta+\sigma_L}/n) = O(n^{\sigma_L - \delta})$. On the other hand, the number of steps in coupling is $|T(v)|$, which implies that the total shift is upper bounded by

$$\begin{aligned}
\left(1 + o(n^{2\delta - 3\sigma_L - 1})\right)^{|T(v)|} \cdot \left(1 + O(n^{\sigma_L - \delta})\right)^{|T(v)|} &\leq \left((1 + o(1)) \cdot O(n^{\sigma_L - \delta})\right)^{|T(v)|}\\
&\leq \left(1 + O(n^{\sigma_L - \delta})\right)^{|T(v)|},
\end{aligned}$$

which concludes the proof. □

Lemma 4.4.62 (Similar to the Coupling Lemma in [41]. See Lemma 6.7 of the Arxiv version.)**.** *Let T be a rooted tree with edges of level smaller than ℓ that is queried by the algorithm where the root of the tree is in $\{A_r, B_r, D_r\}$. Also, suppose we condition on the event in Lemma 4.4.60. Then, the probability of seeing the same tree is equal for all possible roots in $\{A_r, B_r, D_r\}$ up to $(1 + o(n^{2\delta - 3\sigma_{\ell-1}-1}))^{|T|}$ multiplicative factor.*

Proof of Lemma 4.4.56. To begin, as per Observation 4.4.49, since v is not an ℓ-spoiled vertex, the ℓ-shallow subgraph of v forms a rooted tree. It is important to note that we condition our analysis on the labels of all vertices, excluding those in the ℓ-shallow subgraph of vertex v. However, in the coupling detailed in Lemma 4.4.62, there is no conditioning on vertex labels. Given that the total number of vertices for which we condition on their labels are at most $O(n^{1-\delta+\sigma_{\ell-1}})$, each step of the coupling in Lemma 4.4.62 has a probability shift of at most $O(n^{1-\delta+\sigma_{\ell-1}}/n) = O(n^{\sigma_{\ell-1}-\delta})$. On the other hand, the number of steps involved in the coupling process is $|T^\ell(v)|$, which implies that the total shift is upper bounded by

$$\begin{aligned}\left(1 + o(n^{2\delta-3\sigma_{\ell-1}-1})\right)^{|T^\ell(v)|} \cdot \left(1 + O(n^{\sigma_{\ell-1}-\delta})\right)^{|T^\ell(v)|} &\leq \left((1 + o(1)) \cdot O(n^{\sigma_{\ell-1}-\delta})\right)^{|T^\ell(v)|} \\ &\leq \left(1 + O(n^{\sigma_{\ell-1}-\delta})\right)^{|T^\ell(v)|},\end{aligned}$$

which finishes the proof. □

4.4.8 Proof of Lemma 6.41

In this section, we also employ analogous lemmas, such as Lemma 4.4.60 and Lemma 4.3.20, to show a coupling between the two distributions.

Lemma 4.4.63. *Let C be a connected component of black edges that is a tree such that there is no edge of E_ℓ^{inner} in C. With high probability, the longest path of the undirected edges of C is smaller than $r - 1$.*

Proof. Consider a path in component C with length $k > r-2$. Suppose that we put the edges of the path on a line from left to right. Each edge has a direction that is either directed toward the left or directed toward the right. We let $a_i \in \{'\leftarrow', '\rightarrow'\}$ denote the direction of the edge on this line. Note that, by Corollary 4.4.37, the length of the longest directed path of black edges cannot be larger than $5/\sigma_\ell$. Thus, there must exist at least $k/(5/\sigma_\ell)$ different i such that $a_i \neq a_{i+1}$ and $i < k$. For such i, we say that there is a collision at edge i. Furthermore, if there is a collision at edges i_1 and i_2 such that $i_2 > i_1$ and i_2 is the first collision after i_1, then $a_{i_1} \neq a_{i_2}$. Therefore, there must exist at least $\lfloor k/(10/\sigma_\ell)\rfloor > k/(20/\sigma_\ell)$ collisions such that $a_i = '\rightarrow'$ and $a_{i+1} = '\leftarrow'$. It is not hard to see that these types of collision are intersections between descendants of two vertices. Hence, if there exists a path of length k, then there must exist at least $k/(20/\sigma_\ell)$ intersections between descendants of vertices in component C.

On the other hand, we have

$$\begin{aligned}\frac{20k}{\sigma_\ell} &> \frac{10r}{\sigma_\ell} && \text{(Since } k > r/2\text{)} \\ &= \frac{10^{L+1}}{\delta^{L+1} \cdot \sigma_\ell} && \left(\text{Since } r = \left(\frac{10}{\delta}\right)^{L+1}\right) \\ &> 10/\delta && \text{(Since } \sigma_\ell \geq 1 \text{ and } L \geq 0\text{)},\end{aligned}$$

which implies that there must exist more than $10/\delta$ intersections between descendants of vertices in component C which is not possible because of Corollary 4.4.41. □

Corollary 4.4.64. *Let C be a connected component of black edges that is a tree such that there is no edge of E_ℓ^{inner} in C. Consider an arbitrary vertex v in this component where $v \in \{A_r, B_r, D_r\}$. Then, all paths that start from v to an arbitrary vertex in the component that does not contain a mixer vertex, have at most $r-2$ special edges on it.*

Proof. By Lemma 4.4.63 the longest path of the component C is smaller than $r-1$ and therefore, no path in the component contains $r-1$ special edges. □

Similar to the previous subsection, we can apply the same coupling as shown in Lemma 6.7 of [41], as the number of neighbors for each subset of vertices aligns with the transition probabilities in their construction. Let us restate the lemma in the context of our parameters.

Lemma 4.4.65 (Similar to the Coupling Lemma in [41]. See Lemma 6.7 of the Arxiv version.)**.** *Let C be a connected component of black edges corresponding to the edges of level smaller than ℓ that is a tree such that there is no edge of E_ℓ^{inner} in C. Also, suppose that we condition on the event of Corollary 4.4.64. Then, the probability of seeing the same component is equal for both distributions up to $(1+o(n^{2\delta-3\sigma_{\ell-1}-1}))^{|C|}$ multiplicative factor.*

Proof of Lemma 4.4.55. Similar to the argument of proof of Lemma 4.4.29 and Lemma 4.4.56, the total shift in the probability of the coupling is upper bounded by

$$\begin{aligned}\left(1+o(n^{2\delta-3\sigma_{\ell-1}-1})\right)^{|\overline{C}|}\cdot\left(1+O(n^{\sigma_{\ell-1}-\delta})\right)^{|\overline{C}|} &\leq \left((1+o(1))\cdot O(n^{\sigma_{\ell-1}-\delta})\right)^{|\overline{C}|}\\ &\leq \left(1+O(n^{\sigma_{\ell-1}-\delta})\right)^{|\overline{C}|},\end{aligned}$$

which yields the proof. □

4.4.9 Indistinguishability of Base Level Construction

In this section, first, we show that as a corollary of results in the previous section, we have $|E_1^{inner}| = 0$. This implies that the queried edges by the algorithm in the base level of our hierarchy create very small components, i.e. with size $O(n^{\sigma_1/\sigma_2})$. Moreover, we have the property that the union of these components is a forest and each connected component of the forest has a constant longest path. Then, we are able to use Lemma 4.4.55 to show that the algorithm cannot distinguish if the base level construction is drawn from $\mathcal{D}_{\mathsf{YES}}$ or $\mathcal{D}_{\mathsf{NO}}$ with probability $1-o(1)$.

Corollary 4.4.66. *With a probability of $1-O(1/n)$, it holds $|E_1^{inner}| = 0$.*

Proof. Note that by statement (iii) of Observation 4.4.32, we have $g(1) > 2$. Thus, by Lemma 4.4.33,

$$\Pr[E_1^{inner} = \emptyset] \geq 1 - O(n^{1-g(1)}) \geq 1 - O(1/n) = 1 - o(1). \quad \square$$

Claim 4.4.67. *With probability $1-o(1)$, all connected components of queried edges in the base level of the hierarchy are trees.*

Proof. First, by Corollary 4.4.66, we have $|E_1^{inner}| = 0$ with probability $1 - O(1/n)$. Let us condition on this event. Now, by Claim 4.4.54, with probability $1 - O(n^{-\delta+\sigma_1+10\sigma_1/\sigma_2}) = 1 - o(1)$, all connected components of queried edges in the base level of the hierarchy are trees which conclude the proof. □

The above claim enables us to use Lemma 4.4.55 since all connected components are small and it is hard for the algorithm to learn the label of vertices in layer r of the construction. This will help us to prove that the algorithm cannot distinguish if the base level of the construction is drawn from $\mathcal{D}_{\mathsf{YES}}$ or $\mathcal{D}_{\mathsf{NO}}$. We define *bad event* to be the event that Claim 4.4.67 does not hold. By Claim 4.4.67 the probability of the bad event is $o(1)$. Let us condition on not having a bad event. Now we prove that if there is no bad event in the queried subgraph of the base level of the hierarchy, then it is not possible for the algorithm to distinguish if the input graph is drawn from $\mathcal{D}_{\mathsf{YES}}$ or $\mathcal{D}_{\mathsf{NO}}$.

Claim 4.4.68. *Let V_B be the set of vertices that the algorithm finds at least one of their incident edges in the base level. Let $v \in V_B$ and $N_B(v)$ be all neighbors of v in the queried subgraph of the base level. Then, with high probability, for each v there are at most $\widetilde{O}(1)$ edges to vertices of $V_B \setminus N_B(v)$ in the underlying subgraph of base level.*

Proof. We have $|V_B| = O(n^{1-\delta+\sigma_1})$ by Claim 4.4.35. Let $u \in V_B \setminus N_B(v)$. By Corollary 4.4.18, the probability of having an edge between v and u is at most $O(n^{\sigma_L - 1})$. Define X_u be the event that there exists an edge between v and u. Thus, $\Pr[X_u = 1] \leq O(n^{\sigma_L-1})$. Let $X = \sum_{u \in V_B \setminus N_B(v)} X_u$. Hence, $\mathbf{E}[X] \leq O(n^{\delta+\sigma_1+\sigma_L})$ because $|V_B \setminus N_B(v)| \leq O(n^{1-\delta+\sigma_1})$. Let $\lambda = (8 \log n)/\mathbf{E}[X]$. Since events are negatively correlated, using the Chernoff bound we obtain

$$\begin{aligned}
\Pr\left[X \geq (1+\lambda)\,\mathbf{E}[X]\right] &\leq \left(\frac{e^\lambda}{(1+\lambda)^{1+\lambda}}\right)^{\mathbf{E}[X]} \\
&\leq \left(\frac{e^\lambda}{\lambda^\lambda}\right)^{\mathbf{E}[X]} && \text{(Since } \lambda > 1\text{)} \\
&= \left(\frac{e}{\lambda}\right)^{8\log n} && \text{(Since } \lambda = (8\log n)/\mathbf{E}[X]\text{)} \\
&\leq \frac{1}{n^8} && \text{(Since } \lambda > e^2\text{).}
\end{aligned}$$

Therefore, with probability $1 - n^{-8}$, there are at most $\widetilde{O}(1)$ edges to vertices of $V_B \setminus N_B(v)$ in the underlying subgraph of base level. Applying union bound for all vertices finishes the proof. □

Lemma 4.4.69. *Let us condition on not having the bad event defined above. Let $C_1, C_2, \ldots, C_c$ be the components of the forest that the algorithm found in the base level of the construction on a graph drawn from $\mathcal{D}_{\mathsf{YES}}$. Then, the probability of querying the same forest in a graph that is drawn from $\mathcal{D}_{\mathsf{NO}}$ is at least almost as large, up to $1 + o(1)$ multiplicative factor.*

Proof. By Lemma 4.4.63, the maximum longest path of all components is smaller than $r - 1$. Therefore, there is no path in any of the components that has $r-1$ special edges on it. We prove that the probability of seeing the same set of components is almost the same in both $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$ within $1 + o(1)$ multiplicative factor. Let $\mathcal{L}$ be the labeling in $\mathcal{D}_{\mathsf{YES}}$. We will produce a labeling $\mathcal{L}'$ in $\mathcal{D}_{\mathsf{NO}}$ and prove that the probability of seeing this labeling is almost the same as $\mathcal{L}$. With a similar approach, we can also couple each labeling in $\mathcal{D}_{\mathsf{NO}}$ to a labeling in $\mathcal{D}_{\mathsf{YES}}$. We start to iterate over the components one by one. Consider a component C.

At any point, we condition on labels that we already revealed in $\mathcal{L}'$. If there is no edge between two vertices from A_r in the component, we use the same labeling for $\mathcal{L}'$ since all other edges of $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$ are the same.

Now suppose that there is an edge (u, v) such that $u, v \in A_r$. Let C_u and C_v be two components that will be created if we remove edge (u, v). In $\mathcal{L}'$, we let $u \in A_r$ and $v \in B_r$. We couple labels of C_u and C_v according to Lemma 4.4.65. We use the same approach as proof of Lemma 4.4.55 and Lemma 4.4.56. In proof of Lemma 4.4.65, we assumed that because of the conditioning on revealed labels (in total $O(n^{1-\delta+\sigma_1})$ labels), there is an $O(n^{\sigma_1-\delta})$ shift in the probability of the coupling of Lemma 4.4.65 for each step of the coupling. However, this argument is loose, since each vertex in the component is connected to at most $\widetilde{O}(1)$ vertices with revealed labels by Claim 4.4.68. Conditioning on this fact, each step in the coupling is going to have at most $\widetilde{O}(1/n)$ shift in the probability, and in total we have $o(1)$ shift in the probability since the number of steps is equal to the total number of edges queried by the algorithm in the base level of construction which is $O(n^{1-\delta+\sigma_1})$. Therefore, we can couple the two distributions such that the probability of querying the same forest in both distributions is almost the same, up to $1 + o(1)$ multiplicative factor. □

Proof of Lemma 4.4.2. By Lemma 4.4.14, any algorithm that estimates the size of the maximum matching with εn additive error must be able to distinguish whether it belongs to $\mathcal{D}_{\mathsf{YES}}$ or $\mathcal{D}_{\mathsf{NO}}$. Furthermore, according to Lemma 4.4.69, the outcome distribution discovered by the algorithm is in a total variation distance of $o(1)$ for $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$. Hence, the algorithm cannot between the support of two distributions with constant probability taken over the randomization of the input distribution. Therefore, any deterministic algorithm that provides an estimate $\widetilde{\mu}$ of the size of the maximum matching of G such that $\mathbf{E}_G[\widetilde{\mu}] \geq \mu(G) - \varepsilon n$ must spend at least $\Omega(n^{2-\delta})$ time. □

4.5 Extension to Adjacency Matrix

In this section, we extend the lower bound from the previous section to the adjacency matrix model. More formally, we prove the following theorem.

Theorem 4.5.1. *For every $\delta > 0$ there exists $\varepsilon > 0$ (i.e., ε is only a function of δ) such that any algorithm (possibly randomized) that estimates (with probability at least 2/3) the size of maximum matching up to an additive error of εn must make at least $\Omega(n^{2-\delta})$ queries to the adjacency matrix of the graph.*

4.5.1 Technical Overview

In this section, we provide a high-level overview of the lower bound presented in Theorem 4.5.1. Before diving into the new techniques and ideas introduced in this result, we first briefly review the previous constructions for adjacency list lower bounds and their key concepts. Then, we identify the challenges that arise when working with the adjacency matrix, particularly when the algorithm can query pairs corresponding to "non-edges", and explain how we address these challenges.

Existing Constructions: Ideas and Barriers

We first focus on the result of Section 4.2 and its core construction[4]. The input distribution in Section 4.2 consists of two types of graphs: $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$. There is a significant gap between the sizes of the maximum matchings in graphs drawn from these distributions, with $\mathcal{D}_{\mathsf{YES}}$ having a larger matching. The key idea of the result is to demonstrate that the algorithm cannot distinguish *good matching* edges in $\mathcal{D}_{\mathsf{YES}}$. To achieve this, they prove that the queried subgraph within the core forms a tree, as the core is sufficiently sparse and the algorithm makes at most $O(n^{1.2})$ queries. Finally, they show that the queried trees from both distributions are nearly identical using a coupling argument, preventing the algorithm from differentiating between the two types, which implies that no algorithm can achieve 2/3-approximation in $o(n^{1.2})$ time. The construction in Section 4.3 follows a similar outline but introduces a modification in the core that results in a lower bound with a different trade-off between the approximation ratio and running time. However, this lower bound still heavily relies on the fact that the queried subgraph within the core remains acyclic due to the choice of the core's degree and the imposed bound on the algorithm's running time.

Non-edges reveal information: Let us zoom out and examine why these approaches are insufficient for proving a lower bound in the adjacency matrix model. In this model, the algorithm can focus all its queries within the core, as it has the power to choose which pairs of vertices to query. Furthermore, while the core is sufficiently sparse and most of the queries result in non-edges, these non-edges still provide information about the construction. For example, consider the case where we are given a graph that contains only a perfect matching (which is hidden from the algorithm). Before any queries are made, if the algorithm randomly queries a pair of vertices (u, v), the probability that the pair forms an edge is exactly $1/(n-1)$. However, if the result is a non-edge, the probabilities of other pairs shift. For instance, the probability that a pair (u, w) is an edge, conditioned on (u, v) being a non-edge, becomes $1/(n-2)$. While this may at first appear insignificant, it has to be noted that the algorithm is given nearly quadratically many queries, and so will learn about a huge number of non-edges. Therefore, we must carefully account for the non-edges when proving lower bounds, especially when coupling the queried subgraph in the two distributions. In particular, we need to include non-edges in the coupling argument. Notably, the state-of-the-art result for proving lower bounds in the adjacency list model (see Section 4.4) discovers at most $o(n)$ relevant edges and its arguments substantially relied on this limitation.

Introducing Pseudo Edges

As discussed earlier, non-edges reveal information about the construction. However, a key observation is that the amount of information revealed by non-edges depends on the density of the graph. If the construction is sparse, we expect most of the queries to be non-edges. Intuitively, this means the information revealed by these non-edges is very limited. To provide more intuition, consider the following simple example. Suppose we have three subsets of vertices, V_1, V_2, and V_3, where there exists a perfect matching between V_1 and V_2, a 2-regular graph between V_2 and V_3, and no edges between V_1 and V_3. Now, suppose the algorithm queries a pair (u, v) and it turns out to be a non-edge. In this case, it is slightly more likely that the pair belongs to $V_1 \times V_3$ compared to other possible pairs, but the difference in probability is bounded by $O(1/n)$.

[4]We disregard the dummy vertices in the construction, as their primary role is to congest the results of adjacency list queries when the core construction is sparse.

To formalize this observation, we introduce the concept of *pseudo edges*. The idea is to mark some of the non-edges as pseudo edges such that all other non-edges reveal no information about the construction. In the example above, suppose that for each pair in $V_i \times V_j$ for $i \neq j$, we randomly choose a $\log n$-regular graph and mark those pairs as *pseudo edges*. Note that we do not place actual edges between these pairs; they remain non-edges but are marked for the sake of analysis. Additionally, suppose the actual edges are chosen as a subgraph of the pseudo edges. Now, if a query returns a non-edge that is not marked as a pseudo edge, it does not reveal any information about which V_i the endpoints belong to. This is because the distribution of non-edges that are not pseudo edges is identical across any two pairs of subsets. Therefore, we can ignore such queries, as they do not provide useful information, and instead focus on the pseudo edges, which are significantly fewer in number.

The Challenge with Pseudo Edges

Before describing how we actually use pseudo edges, we start with perhaps the "obvious" way of using pseudo edges and argue why these methods do not quite work. These examples are meant to illustrate why our final construction has to be somewhat involved and rather counter-intuitive.

Attempt 1: regular pseudo edges and regular real edges. This idea is exactly the same as what we discussed when introducing pseudo edges. However, although non-edges that are not marked as pseudo edges are independent of vertex labels, we face another challenge that makes proving a lower bound difficult. In this approach, we must ensure that the regular graph of real edges is a subgraph of the regular graph of pseudo edges. Achieving this requires a global view of the pseudo edge graph to determine the real edges. However, this introduces correlations between the real edges, making it extremely challenging to show any lower bound.

Attempt 2: Erdős–Rényi pseudo edges and regular real edges. Instead of using a regular graph for pseudo edges, we now consider an Erdős–Rényi random graph, which is more natural when working with the adjacency matrix. Take the example with vertex sets V_1, V_2, and V_3. For each pair in $V_i \times V_j$ with $i \neq j$, we mark the pair as a pseudo edge with probability $\log n/n$, mimicking an Erdős–Rényi graph with an expected degree of $\log n$. Additionally, suppose that real edges are selected only from among the marked pseudo edges. As before, if a query results in a non-edge that is not marked as a pseudo edge, it provides no information about which V_i the endpoints belong to. This is because the Erdős–Rényi graph of non-pseudo edges is identical across all pairs of subsets.

However, while non-pseudo edges remain independent of vertex labels, a new challenge arises that complicates proving a lower bound. Let F be the set of queried pairs that were identified as non-edges and were not marked as pseudo edges. Let L be an indicator random variable representing whether a given vertex v has a specific label. We know that it holds $\Pr[L \mid F] = \Pr[L]$. This is because pseudo edges are independent of vertex labels and form an Erdős–Rényi graph between different labels, regardless if they are real edges. Now, suppose H is a subset of the actual edges that have been queried and found. Our entire argument relies on the assumption that non-pseudo edges provide no information about labels, meaning the algorithm should ignore them. This requires showing that $\Pr[L \mid F, H] = \Pr[L \mid H]$. However, note that H and F are not necessarily independent. Essentially, our goal is to select a regular subgraph from an Erdős–Rényi graph. This process involves first examining all the edges of the Erdős–Rényi graph and then choosing a

subset of them. Consequently, the decision of whether a given pair forms an edge is not independent of the decisions for other pairs.

Attempt 3: Erdős–Rényi pseudo edges and Erdős–Rényi real edges. With the intuition from the previous attempt in mind, we need to develop a construction that ensures the decision of whether a given pair forms a real edge is independent of the decisions for other pairs. To achieve this goal, we choose an Erdős–Rényi graph for both pseudo edges and real edges. More specifically, consider the example with vertex sets V_1, V_2, and V_3. For each pair of vertices, we mark the pair as a pseudo edge with probability $\log n/n$. Furthermore, each pair between V_1 and V_2 is marked as a real edge with probability $1/\log n$, each pair between V_2 and V_3 is marked as a real edge with probability $2/\log n$, and each pair between V_1 and V_3 is marked as a real edge with probability $2/\log n$. Finally, an edge is included in the final construction if and only if it is marked as both a pseudo edge and a real edge. As a result, the expected degree will match our intended values (1 for V_1, 3 for V_2, and 2 for V_3), consistent with the regular real-edge construction attempt. Furthermore, we achieve the desirable independence property we were seeking.

However, another challenge arises with this construction. Consider two vertices, v and u, with different labels but the same expected degree. Suppose that u and v each appear in a gadget containing n other vertices. In the first gadget, v forms real edges with other vertices with probability $2/n$, while u does so with probability $1/n$. In the second gadget, v's probability remains $2/n$, whereas u's increases to $3/n$. Thus, both v and u have an expected degree of 4. But the variance of the degree for v and u differs. More specifically, we have $\mathrm{Var}(\deg(v)) = 4 - 8/n$ and $\mathrm{Var}(\deg(u)) = 4 - 10/n$. The algorithm can use this variance to learn about the structure of the input and determine whether the instance is from $\mathcal{D}_{\mathsf{YES}}$ or $\mathcal{D}_{\mathsf{NO}}$.

Our Actual Construction via *Parallel* Pseudo Edges

To address the issue with variance, we use the following approach to bound the total variation distance between different degree distributions. Fix a pair of vertices u and v. Let ρ denote the probability that this pair can form a real edge according to the gadgets used in the construction. We add ρn parallel edges between (u, v) in the graph, which we refer to as *ground edges*. For each ground edge, we independently flip a coin with probability $1/n$ to determine whether it becomes a real edge. Note that for this fixed pair, the expected number of real edges between them is exactly ρ, aligning with our intended construction. If u and v have the same total expected degree, then the degree distribution is identical for all vertices. More formally, their degrees follow the same number of Bernoulli random variables (since their expected degrees are equal), each with probability $1/n$.

It is important to emphasize that our actual multigraph construction is significantly more intricate than the simplified version presented here, as it must incorporate pseudo edges and satisfy several additional properties. However, to convey the core idea while avoiding unnecessary technical details, we have chosen to present a simplified version. In the actual construction, pseudo edges are a subset of ground edges, and real edges are a subset of pseudo edges (See Figure 4.7).

Algorithm cannot distinguish between multigraph and simple graph. One caveat of using this multigraph approach is that there may be multiple real edges between a pair of vertices. Since the input to our problem is a simple graph, we need to show that, with high probability, the algorithm cannot identify pairs with multiple real edges. Suppose the algorithm runs in $O(n^{2-\delta})$ time. One key observation is that

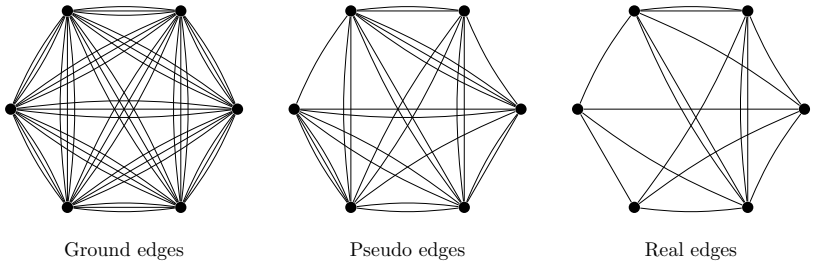


Figure 4.7: An illustration of ground, pseudo, and real edges. As shown in the figure, pseudo edges form a subset of ground edges, while real edges are a subset of pseudo edges.

our construction is sparse, with vertex degrees of roughly n^σ (where $2\sigma < \delta$), which implies $\rho = n^{\sigma-1}$. Since there are n^2 vertex pairs in the graph and at most ρn ground edges per pair, the total number of ground edges is at most ρn^3. Each of these ground edges is realized as a real edge with probability $1/n$, implying that the total number of real edges is at most $O(\rho n^2) = O(n^{1+\sigma})$. We condition on the high probability event that this bound holds. Let $x = O(n^{1+\sigma})$ denote the number of real edges in the graph. Using a birthday paradox argument, we can show that the number of vertex pairs with more than one real edge is approximately $x^2/n^2 = O(n^{2\sigma})$. Since the algorithm makes at most $O(n^{2-\delta})$ queries and $2\sigma < \delta$, it cannot, with probability $1 - o(1)$, query any pair that has multiple real edges. Therefore, with high probability, the algorithm cannot distinguish whether the input graph originates from this multigraph construction or from a simple graph.

Modifications to the previous techniques to account for pseudo edges. Finally, it is important to highlight additional technical challenges that arise due to the introduction of pseudo edges in the construction. Since non-edges marked as pseudo edges are not independent of the labeling, we need to adjust the coupling argument to properly account for them. Furthermore, since we aim to demonstrate that the algorithm's advantage in detecting cycles diminishes as it progresses deeper into the construction, we must show that deeper gadgets have lower density compared to higher-level gadgets even when we have pseudo edges. To achieve this, we assign different probabilities for marking pseudo edges to belong to different levels, using smaller probabilities for deeper levels. This adjustment helps reinforce the argument that the algorithm's advantage decreases as it explores deeper into the structure.

4.5.2 Reduction to Earth Mover's Distance

In this section, we reduce the estimation of maximum matching size to the estimation of Earth Mover's Distance (EMD, a.k.a. Optimal Transport, Wasserstein-1 Distance or Kantorovich–Rubinstein Distance), and obtain the same query lower bound for $\Omega(\varepsilon)$-additive approximation. This lower bound implies that the algorithm of Beretta and Rubinstein [45] is tight up to the $O_\varepsilon(1)$ factor in the power, i.e. for any $\varepsilon > 0$ they present a $n^{2-\Omega_\varepsilon(1)}$-time algorithm that approximates EMD up to an ε-additive error.

Definition 4.5.2 (EMD)**.** *Given two distributions p and q over a metric space $(\mathcal{M}, d)$, their EMD is defined as*

$$\mathrm{EMD}(p, q) = \min \left\{ \mathbf{E}_{x,y\sim\varphi}[d(x, y)] \mid \varphi \text{ is a coupling of } p \text{ and } q \right\}.$$

For this section, we normalize the distances and assume they lie in $[0,1]$ for all pairs. The lower bound is formalized as follows.

Theorem 4.5.3. *Let p and q be discrete distributions of support size n on metric $(\mathcal{M}, d)$, such that $d : \mathcal{M}^2 \to [0,1]$. For every $\delta > 0$ there exists $\varepsilon > 0$ (i.e., ε is only a function of δ), such that any algorithm with query access to d, that computes an ε-additive approximation of $\mathrm{EMD}(p,q)$, requires at least $\Omega(n^{2-\delta})$ queries.*

Remark 7. *The underlying metric in our lower bound is a $(1,2)$-metric.*

Note that the lower bound on the number of queries to d also provides a lower bound on the time complexity. The theorem follows directly from Theorem 4.5.1 and the reduction below.

Claim 4.5.4. *Assume there exists an algorithm $\mathcal{A}$ that, given two distributions of support size n on a normalized metric space, computes an $\varepsilon/2$-additive approximation to EMD using Q queries. Then, there exists an algorithm $\mathcal{A}'$ that, given a bipartite graph G, computes a εn-additive approximation to the maximum matching size using the same number of queries to the adjacency matrix.*

Proof. We define algorithm $\mathcal{A}'$. Let A and B be the two parts of G, and let $\mathcal{M} = A \cup B$. Assume, without loss of generality, that $|A| = |B| = n$. Let p and q be uniform distributions on A and B respectively, i.e. $p(u) = 1/n$ for $u \in A$ and $q(u) = 1/n$ for $u \in B$. Finally, let

$$d(u,v) = \begin{cases} 1/2, & \text{if } (u,v) \in E(G), \text{ and} \\ 1, & \text{otherwise.} \end{cases}$$

Note that other distances are not relevant to $\mathrm{EMD}(p,q)$, and any distance query can be answered using an adjacency matrix query between the same pair. Algorithm $\mathcal{A}'$ simply reports $2n - 2n\mathcal{A}(\mathcal{M}, d)$.

To prove $\mathcal{A}'$ is computing an εn-additive approximation of $\mu(G)$, it suffices to show

$$\mathrm{EMD}(p,q) = \frac{2n - \mu(G)}{2n}.$$

First, observe that a coupling φ of p and q, when scaled up by a factor of n, is a perfect fractional matching of A and B. That is, after weighting the edges according to d, the minimum-weight perfect fractional matching has weight $n \cdot \mathrm{EMD}(p,q)$. Since the graph is bipartite, this is the same as the minimum-weight perfect (integral) matching. It remains to show that the minimum-weight perfect matching has weight $n - \mu(G)/2$. Any perfect matching that includes x edges of G, must include $n - x$ edges outside G, and thus has weight

$$x \cdot \frac{1}{2} + (n - x) \cdot 1 = n - \frac{x}{2}.$$

Finally, note that the largest possible value for x is $\mu(G)$. Therefore, the minimum weight perfect matching has weight $n - \mu(G)/2$, and

$$\mathrm{EMD}(p,q) = \frac{n - \mu(G)/2}{n} = \frac{2n - \mu(G)}{2n}.$$

□

Parameter	Value	Definition
δ	-	Parameter that controls the running time of the algorithm. More specifically, the algorithm has $O(n^{2-\delta})$ running time.
ρ	$2n^{\delta/10-1}$	The probability that each pseudo edge exists in the Erdős–Rényi graph of pseudo edges.
ρ_i	$2n^{\sigma_i-1}$	The probability that each pseudo edge exists in the Erdős–Rényi graph of pseudo edges of level i.
L	$4/\delta$	Number of **levels** in the recursive hierarchy for the construction of input distribution.
r	$(10/\delta)^{L+1}$	Number of **layers** in each level of the hierarchy.
$\mathcal{D}^i_{\mathsf{YES}}$	-	Distribution of level i graphs in the construction hierarchy that have a perfect matching.
$\mathcal{D}^i_{\mathsf{NO}}$	-	Distribution of level i graphs that at most $(1-\varepsilon)$ fraction of their vertices can be matched in the maximum matching.
σ_i	$(\delta/10)^{L+1-i}$	Parameter that controls the degree of vertices in graphs of level i.
d_i	n^{σ_i}	Parameter that controls the degree of vertices in graphs of level i.
ζ	$1/r^2$	Fraction of vertices that are delusive in each level.
ξ	$1/r^2$	The gap between the size of A_r and B_r in the base construction.
γ	$1/r^4$	Degree to delusive vertices is γd.
N_i	$N_i = n_{i-1}/(2\zeta)$	Parameter that controls the number of vertices in graphs of level i.
n_i	$(8+16r+4\zeta r)N_i$	Total number of vertices in a graph of level i.
n	$(1+\tau)\cdot n_L$	Number of vertices in a graph drawn from the final distribution.

Table 4.2: Variables used throughout this section.

4.5.3 The Construction

We define our construction in several steps. First, we introduce a recursive procedure to generate the gadgets used in the construction. Next, we demonstrate how to construct a multigraph using these gadgets. We then establish key properties of the multigraph. Following this, we explain how to derive the final simple graph from the constructed multigraph.

Two Input Distributions

To prove the lower bound, we construct a distribution of graphs and show that any *deterministic* sublinear algorithm that computes an εn-additive approximation of the matching size on this distribution, requires $\Omega(n^{2-\delta})$ adjacency matrix queries. Then, it follows from Yao's min-max theorem that *any (possibly randomized)* algorithm that approximates the matching size up to an εn additive error requires the same number of queries.

The final distribution $\mathcal{D}$ is a mix of two input distributions $\mathcal{D} := (\mathcal{D}_{\mathsf{YES}} + \mathcal{D}_{\mathsf{NO}})/2$, where:

1. a graph drawn from $\mathcal{D}_{\mathsf{YES}}$ contains a perfect matching with high probability, and
2. a graph drawn from $\mathcal{D}_{\mathsf{NO}}$ leaves $\Omega(\varepsilon n)$ vertices unmatched.

Observe that any algorithm that computes an $O(\varepsilon n)$-additive approximation of the matching size with constant probability must be able to distinguish between $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$ with constant probability. Using this observation, we prove the following lemma.

Lemma 4.5.5. *For every $\delta > 0$ there exists $\varepsilon > 0$ (i.e., ε is only a function of δ), take a* deterministic *sublinear algorithm that given a graph G drawn from $\mathcal{D}$, produces an εn-additive approximation of the matching size with probability $\frac{2}{3}$. That is, the algorithm computes $\tilde{\mu}(G)$ such that*

$$\Pr[\mu(G) - \varepsilon n \leq \tilde{\mu}(G) \leq \mu(G)] \geq \frac{2}{3}$$

where the probability is over the graph G drawn from $\mathcal{D}$. Then, the algorithm requires $\Omega(n^{2-\delta})$ adjacency matrix queries.

Our main theorem then follows from a direct application of Yao's Lemma (Proposition 2.3.6).

Theorem 4.5.1. *For every $\delta > 0$ there exists $\varepsilon > 0$ (i.e., ε is only a function of δ) such that any algorithm (possibly randomized) that estimates (with probability at least 2/3) the size of maximum matching up to an additive error of εn must make at least $\Omega(n^{2-\delta})$ queries to the adjacency matrix of the graph.*

Proof. In the notation of Proposition 2.3.6, let the cost of an algorithm be the number of adjacency matrix queries. Lemma 4.5.5 states $F_{1,\frac{1}{3}} \geq \inf_{a \in B_{\mathcal{D},\lambda}} c(a, \mathcal{D}) = \Omega(n^{2-\delta})$. Then, from Proposition 2.3.6 we can conclude:

$$F_{2,\frac{1}{3}} \geq \Omega\left(F_{2,\frac{1}{6}}\right) \geq \Omega\left(F_{1,\frac{1}{3}}\right) = \Omega(n^{2-\delta}).$$

That is, for any randomized algorithm that computes an εn-additive approximation of the matching size with probability $\frac{2}{3}$, there exists an input graph G such that the average (hence also the worst case) number of queries is $\Omega(n^{2-\delta})$ □

Recursive Structure of Graphs Drawn from Input Distribution

Here, we describe the recursive structure of the graphs drawn from $\mathcal{D}$. For now, we present the structure as gadgets between vertices. Later, we show precisely how these gadgets are used to construct the multigraph, which is then turned into the final simple graph.

In the recursive structure, to obtain a level ℓ graph, we use graphs of level $\ell - 1$ combined with some other gadgets that make it hard for the algorithm to distinguish edges of graphs of level $\ell - 1$. The ultimate goal is

to hide some important edges of level 1 that make a difference between the $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$. We use $\mathcal{D}^{\ell}_{\mathsf{YES}}$ and $\mathcal{D}^{\ell}_{\mathsf{NO}}$ for the distributions of level i graphs for $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$, respectively. Also, we let $\mathcal{D}^{\ell} := (\mathcal{D}^{\ell}_{\mathsf{YES}} + \mathcal{D}^{\ell}_{\mathsf{NO}})/2$.

Gadgets for Level 1

In this section, we introduce gadgets and define the first level, i.e. the base case, of the recursive structure. Each gadget is defined between two disjoint vertex sets X and Y by a parameter $P(X, Y)$ that controls the density of the edges. Intuitively, the gadget mimics a bipartite Erdős–Rényi graph between X and Y where there exists an edge between each pair with probability $P(X, Y)$. Based on this intuition, we define $d_{\phi}(u)$, the contribution of a gadget ϕ to the expected degree of a vertex u:

$$d_{\phi}(u) = \begin{cases} P(X,Y)\,|Y|\,, & \text{if } u \in X, \\ P(X,Y)\,|X|\,, & \text{if } u \in Y, \text{ and} \\ 0, & \text{otherwise.} \end{cases}$$

Here, ϕ is a gadget between X and Y with density parameter $P(X, Y)$. For a set of gadgets Φ, the contribution is simply defined as $d_{\Phi}(u) = \sum_{\phi \in \Phi} d_{\phi}(u)$.

The actual construction, however, is more intricate as described later. There, a multigraph is constructed such that for a pair of vertices $u \in X$ and $v \in Y$ the expected number of edges between them is $P(X, Y)$. We use the following remark to prove some properties of the multigraph and postpone the proof to the end of this section to abstract away the details.

Remark 8. *Take two vertices u and v such that there is a gadget with density parameter p between them. Then in the multigraph, the expected number of edges between u and v is p (Corollary 4.5.17), and there is at least one edge between them with probability $p/2$ (Corollary 4.5.18). If there are no gadgets between u and v, then there are no edges between them.*

The base case of the construction consists of $6r + 2$ vertex sets which are divided into r layers. For a vertex u in a set X, we may refer to X as the label of u on this level. The sizes of these sets are fixed, but the vertices are otherwise divided between them at random. The vertex sets are defined as follows:

- A_i^j for $i \in [r]$ and $j \in \{1, 2\}$. For $i < r$ and $j \in \{1, 2\}$, each subset A_i^j contains N_1 vertices. Each of A_r^1 and A_r^2 contains $(1 - \xi)N_1$ vertices.
- B_i^j for $i \in [r]$ and $j \in \{1, 2\}$, each containing N_1 vertices.
- D_i^j *(the dummy vertices)* for $i \in [r]$ and $j \in \{1, 2\}$, each containing ζN_1 vertices.
- S^j *(the special vertices)* for $j \in \{1, 2\}$, each containing N_1 vertices.

Throughout the proof, we use S to denote $S^1 \cup S^2$. Likewise, we use A_i, B_i, and D_i to denote $A_i^1 \cup A_i^2$, $B_i^1 \cup B_i^2$, and $D_i^1 \cup D_i^2$, respectively. Also, we let n_1 be the total number of vertices in a level-1 graph.

Now, we present the gadgets that are the same for the $\mathcal{D}^1_{\mathsf{YES}}$ and $\mathcal{D}^1_{\mathsf{NO}}$ distributions. It consists of (1) sparse gadgets between S^j and B_1^j, (2) a dense gadget between A_i^j and B_i^j in each layer $i < r$, (3) a sparse

gadget between B_j^i and A_{i-1}^j that connects the layers $i \le r$ and $i-1$, and (4) the gadgets to the dummy vertices.

$$\begin{aligned}
&P(S^j, B_1^j) = \frac{\log^2 n}{N_1} && \forall \quad j \in \{1,2\},\\
&P(B_i^j, A_i^j) = \frac{d_1}{N_1} && \forall \quad j \in \{1,2\}, \quad 1 \le i < r,\\
&P(B_i^j, A_{i-1}^j) = \frac{\log^2 n}{N_1} && \forall \quad j \in \{1,2\}, \quad 1 < i \le r,
\end{aligned}$$

The gadgets to the dummy vertices are more involved. Each vertex set A_i^j (resp. B_i^j and D_i^j) on layer i has gadgets to the dummies D_k^{3-j} (resp. D_k^j and D_k^{j-3}) for all layers $k \le i$. There are some gadgets between the dummy vertices of the same layer to control the total degree of each dummy vertex.

$$\begin{aligned}
&P(A_i^j, D_k^{3-j}) = \frac{\gamma d_1}{\zeta N_1} && \forall \quad j{\in}\{1,2\}, \quad 1{<}i{\le}r, \quad k{<}i,\\
&P(A_i^j, D_i^{3-j}) = \frac{(r-i+1)\gamma d_1}{\zeta N_1} && \forall \quad j{\in}\{1,2\}, \quad 1{\le}i{\le}r,\\
&P(B_i^j, D_k^j) = \frac{\gamma d_1}{\zeta N_1} && \forall \quad j{\in}\{1,2\}, \quad 1{<}i{\le}r, \quad k{<}i,\\
&P(B_i^j, D_i^j) = \frac{(r-i+1)\gamma d_1}{\zeta N_1} && \forall \quad j{\in}\{1,2\}, \quad 1{\le}i{\le}r,\\
&P(D_i^j, D_k^{3-j}) = \frac{\gamma d_1}{\zeta N_1} && \forall \quad 1{\le}i{\le}r, \quad 1{\le}k{\le}r, \quad i{\ne}k, \quad j{\in}\{1,2\},\\
&P(D_i^j, D_i^{3-j}) = \frac{d_1 + \gamma d_1 + \log^2 n - (4r-4i+2-\xi)\gamma d_1/\zeta}{\zeta N_1} && \forall \quad 1{\le}i{\le}r, \quad j{\in}\{1,2\}.
\end{aligned}$$

Finally, we describe the gadgets that are different in $\mathcal{D}^1_{\mathsf{YES}}$ and $\mathcal{D}^1_{\mathsf{NO}}$. In the YES case, there is a sparse gadget between A_r^1 and A_r^2. In the NO case, there is no such gadget. There are gadgets between A_r^j and B_r^j, and between B_r^1 and B_r^2 the parameters of which are changed to control the degrees and make up for the discrepancy between the two cases. More precisely, we have the following gadgets **only** in $\mathcal{D}^1_{\mathsf{YES}}$:

$$\begin{aligned}
&P(A_r^j, B_r^j) = \frac{d_1}{N_1} && \forall \quad j \in \{1,2\},\\
&P(A_r^1, A_r^2) = \frac{\log^2 n}{(1-\xi)N_1},\\
&P(B_r^1, B_r^2) = \frac{\xi d_1}{N_1},
\end{aligned}$$

On the other hand, we have the following gadgets **only** in $\mathcal{D}^1_{\mathsf{NO}}$:

$$P(A^j_r, B^j_r) = \frac{d_1 + \log^2 n}{N_1} \qquad \forall \quad j \in \{1, 2\},$$
$$P(B^1_r, B^2_r) = \frac{\xi d_1 - (1-\xi)\log^2 n}{N_1},$$

Now, we establish some useful properties of the graphs $\mathcal{D}^1$. Firstly, we characterize the expected degree of the vertices: all special vertices have the same expected degree, and all the non-special vertices have the same expected degree. Recall that a gadget between vertex sets X and Y with parameter p contributes $p\,|Y|$ to the expected degree of each vertex in X.

Claim 4.5.6. *Let Φ^1_{YES} and Φ^1_{NO} be the gadgets introduced on level 1, and Φ^1 be either one of them. It holds that,*

(i) $d_{\Phi^1}(v) = \log^2 n$, *for* $v \in S$

(ii) $d_{\Phi^1}(v) = d_1 + r\gamma d_1 + \log^2 n$, *for* $v \notin S$.

Proof. A gadget between vertex sets X and Y with parameter p contributes $p\,|Y|$ to the expected degree of each vertex in X. For (i), note that the only edge gadget that we use for vertices of S in both $\mathcal{D}^1_{\mathsf{YES}}$ and $\mathcal{D}^1_{\mathsf{NO}}$ is the gadget between S and B_1. Thus, for $v \in S^j$, we have

$$d_{\Phi^1}(v) = P(S^j, B^j_1) \cdot |B^j_1| = \log^2 n.$$

To prove (ii), we consider different cases for vertex v:

- $v \in A_i$ **for** $1 \le i < r$**:** For $v \in A^j_i$, we have

$$\begin{aligned} d_{\Phi^1}(v) &= P(A^j_i, B^j_i) \cdot |B^j_i| + P(A^j_i, B^j_{i+1}) \cdot |B^j_{i+1}| + P(A^j_i, D^{3-j}_i) \cdot |D^{3-j}_i| \\ &\quad + \sum_{k=1}^{i-1} P(A^j_i, D^{3-j}_k) \cdot |D^{3-j}_k| \\ &= d_1 + \log^2 n + (r-i+1)\gamma d_1 + \sum_{k=1}^{i-1} \gamma d_1 \\ &= d_1 + r\gamma d_1 + \log^2 n. \end{aligned}$$

- $v \in B_i$ **for** $1 \le i < r$**:** We abuse notation in this proof and use A^j_0 to represent S^j. Then, for $v \in B^j_i$, we have

$$\begin{aligned} d_{\Phi^1}(v) &= P(A^j_i, B^j_i) \cdot |A^j_i| + P(B^j_i, A^j_{i-1}) \cdot |A^j_{i-1}| + P(B^j_i, D^j_i) \cdot |D^j_i| \\ &\quad + \sum_{k=1}^{i-1} P(A^j_i, D^j_k) \cdot |D^j_k| \\ &= d_1 + \log^2 n + (r-i+1)\gamma d_1 + \sum_{k=1}^{i-1} \gamma d_1 \end{aligned}$$

$$= d_1 + r\gamma d_1 + \log^2 n.$$

- $v \in A_r$ **in the YES case:** For $v \in A_r^j$, we have

$$\begin{aligned} d_{\Phi^1_{\mathsf{YES}}}(v) &= P(A_r^j, B_r^j) \cdot |B_r^j| + P(A_r^j, A_r^{3-j}) \cdot |A_r^{3-j}| + \sum_{k=1}^{r} P(A_r^j, D_k^{3-j}) \cdot |D_k^{3-j}| \\ &= d_1 + \log^2 n + \sum_{k=1}^{r} \gamma d_1 \\ &= d_1 + r\gamma d_1 + \log^2 n. \end{aligned}$$

- $v \in A_r$ **in the NO case:** For $v \in A_r^j$, we have

$$\begin{aligned} d_{\Phi^1_{\mathsf{NO}}}(v) &= P(A_r^j, B_r^j) \cdot |B_r^j| + \sum_{k=1}^{r} P(A_r^j, D_k^{3-j}) \cdot |D_k^{3-j}| \\ &= d_1 + \log^2 n + \sum_{k=1}^{r} \gamma d_1 \\ &= d_1 + r\gamma d_1 + \log^2 n. \end{aligned}$$

- $v \in B_r$ **in the YES case:** For $v \in B_r^j$, we have

$$\begin{aligned} d_{\Phi^1_{\mathsf{YES}}}(v) &= P(A_r^j, B_r^j) \cdot |A_r^j| + P(B_r^j, B_r^{3-j}) \cdot |B_r^{3-j}| + P(B_r^j, A_{r-1}^j) \cdot |A_{r-1}^j| \\ &\quad + \sum_{k=1}^{r} P(B_r^j, D_k^j) \cdot |D_k^j| \\ &= (1-\xi)d_1 + \xi d_1 + \log^2 n + \sum_{k=1}^{r} \gamma d_1 \\ &= d_1 + r\gamma d_1 + \log^2 n. \end{aligned}$$

- $v \in B_r$ **in the NO case:** For $v \in B_r^j$, we have

$$\begin{aligned} d_{\Phi^1_{\mathsf{NO}}}(v) &= P(A_r^j, B_r^j) \cdot |A_r^j| + P(B_r^j, B_r^{3-j}) \cdot |B_r^{3-j}| + P(B_r^j, A_{r-1}^j) \cdot |A_{r-1}^j| \\ &\quad + \sum_{k=1}^{r} P(B_r^j, D_k^j) \cdot |D_k^j| \\ &= (1-\xi)(d_1 + \log^2 n) + \xi d_1 - (1-\xi)\log^2 n + \log^2 n + \sum_{k=1}^{r} \gamma d_1 \\ &= d_1 + r\gamma d_1 + \log^2 n. \end{aligned}$$

- $v \in D_i$ **for** $1 \le i < r$**:** For $v \in D_i^j$, we have

$$d_{\Phi^1}(v) = P(B_i^j, D_i^j) \cdot |B_i^j| + P(A_i^{3-j}, D_i^j) \cdot |A_i^{3-j}| + P(D_i^j, D_i^{3-j}) \cdot |D_i^{3-j}|$$

$$
\begin{aligned}
&+ P(A_r^{3-j}, D_i^j) \cdot |A_r^{3-j}| + \sum_{k=i+1}^{r-1} P(A_k^{3-j}, D_i^j) \cdot |A_k^{3-j}| \\
&+ \sum_{k=i+1}^{r} P(B_k^j, D_i^j) \cdot |B_k^j| + \sum_{k \neq i} P(D_k^{3-j}, D_i^j) \cdot |D_k^{3-j}| \\
&= \frac{(r-i+1)\gamma d_1}{\zeta} + \frac{(r-i+1)\gamma d_1}{\zeta} + d_1 + \gamma d_1 + \log^2 n - \frac{(4r-4i+2-\xi)\gamma d_1}{\zeta} \\
&+ \frac{(1-\xi)\gamma d_1}{\zeta} + \frac{(r-1-i)\gamma d_1}{\zeta} \\
&+ \frac{(r-i)\gamma d_1}{\zeta} + (r-1)\gamma d_1 \\
&= d_1 + r\gamma d_1 + \log^2 n.
\end{aligned}
$$

- $v \in D_r$**:** For $v \in D_r^j$, we have

$$
\begin{aligned}
d_{\Phi^1}(v) &= P(B_r^j, D_r^j) \cdot |B_r^j| + P(A_r^{3-j}, D_r^j) \cdot |A_r^{3-j}| + P(D_r^j, D_r^{3-j}) \cdot |D_r^{3-j}| \\
&\quad + \sum_{k \neq r} P(D_k^{3-j}, D_r^j) \cdot |D_k^{3-j}| \\
&= \frac{\gamma d_1}{\zeta} + \frac{(1-\xi)\gamma d_1}{\zeta} + d_1 + \gamma d_1 + \log^2 n - \frac{(2-\xi)\gamma d_1}{\zeta} + (r-1)\gamma d_1 \\
&= d_1 + r\gamma d_1 + \log^2 n.
\end{aligned}
$$

Combining all the above cases, we can conclude the proof of (ii). □

Observation 4.5.7. *Any multigraph drawn from $\mathcal{D}^1$ is bipartite.*

Proof. Consider the following partitioning of the vertices to the following two disjoint parts:

(1) $\left(\bigcup_{i=1}^r A_i^1\right) \cup \left(\bigcup_{i=1}^r B_i^2\right) \cup \left(\bigcup_{i=1}^r D_i^1\right) \cup S^1$, and

(2) $\left(\bigcup_{i=1}^r A_i^2\right) \cup \left(\bigcup_{i=1}^r B_i^1\right) \cup \left(\bigcup_{i=1}^r D_i^2\right) \cup S^2$.

According to the gadgets, there are no edges inside each part, which concludes the proof. □

Claim 4.5.8. *In the multigraph construction, all the following hold with high probability:*

- *There exists a perfect matching between A_i^j and B_{i+1}^j for all $1 \leq i < r$ and $j \in \{1,2\}$ in each graph drawn from $\mathcal{D}^1$,*
- *There exists a perfect matching between B_1^j and S^j for all $j \in \{1,2\}$ in each graph drawn from $\mathcal{D}^1$,*
- *There exists a perfect matching between D_i^1 and D_i^2 for all $1 \leq i \leq r$ in each graph drawn from $\mathcal{D}^1$,*
- *There exists a perfect matching between A_r^1 and A_r^2 in each graph drawn from $\mathcal{D}^1_{\mathsf{YES}}$.*

Proof. Consider the induced subgraph between A_i^j and B_{i+1}^j. The induced subgraph is an Erdős–Rényi graph such that each edge exists with a probability of at least $\Omega\left((\log^2 n)/N_1\right)$ (by Remark 8). Also, we have $(\log^2 n)/N_1 > (\log^2 N_1)/N_1$. Moreover, we have $|A_i^j| = |B_i^j| = N_1$. Thus, by using Proposition 2.2.8, there exists a perfect matching in the induced subgraph between A_i^j and B_{i+1}^j with high probability. With the

exact similar approach, we can prove that there exists a perfect matching in each of the induced subgraphs in the claim statement with high probability. □

Lemma 4.5.9. *In the multigraph construction, the size of the maximum matching in $\mathcal{D}^1_{\mathsf{YES}}$ and $\mathcal{D}^1_{\mathsf{NO}}$ is characterized as follows:*

(i) If $G \sim \mathcal{D}^1_{\mathsf{YES}}$, then $\mu(G) = n_1/2$ with high probability.

(ii) If $G \sim \mathcal{D}^1_{\mathsf{NO}}$, then $\mu(G) \leq n_1/2 - N_1/2$.

Proof. Let us condition on the high probability event of Claim 4.5.8. Now suppose that G is drawn from $\mathcal{D}^i_{\mathsf{YES}}$. Therefore,

$$\begin{aligned}\mu(G) &\geq \mu(A^1_r, A^2_r) + \sum_{j\in\{1,2\}} \mu(S^j, B^j_1) + \sum_{j\in\{1,2\}, i<r} \mu(A^j_i, B^j_{i+1}) + \sum_{i\leq r} \mu(D^1_i, D^2_i) \\ &= (1-\xi)N_1 + rN_1(2+\zeta) = n_1/2.\end{aligned}$$

Also, the total number of vertices in the graph is n_1, thus the inequality is tight, which completes the proof of (i).

Now suppose that G is drawn from $\mathcal{D}^i_{\mathsf{NO}}$. Note that vertices of

$$\left(\bigcup_{i\leq r, j\in\{1,2\}} B^j_i\right) \cup \left(\bigcup_{i\leq r, j\in\{1,2\}} D^j_i\right),$$

form a vertex cover of G. Observe that the size of any vertex cover must be larger than the size of the maximum matching because for each edge in the maximum matching, at least one of the endpoints must be in the vertex cover. Therefore,

$$\mu(G) \leq \sum_{i\leq r, j\in\{1,2\}}^{r} |B^j_i| + \sum_{i\leq r, j\in\{1,2\}}^{r} |D^j_i| = 2rN_1 + 2r\zeta N_1 = \frac{n_1}{2} - (1-\xi-r\zeta)N_1 < \frac{n_1 - N_1}{2},$$

which completes the proof of (ii). □

Gadgets for Level ℓ

Now, we describe the higher levels of the recursive structure. For a level $1 < \ell \leq L$, constructing a graph of $\mathcal{D}^\ell$ involves drawing multiple instances from $\mathcal{D}^{\ell-1}$. Let $N_\ell = n_{\ell-1}/(2\zeta)$ be a parameter that controls the number of vertices in level ℓ of the construction, and d_ℓ be a parameter that controls the degree of vertices in level ℓ of the construction. Recall $n_{\ell-1}$ is the total number of vertices for a graph of level $\ell - 1$.

Each graph in $\mathcal{D}^\ell$ consists of $8r + 2$ vertex sets which are divided into r layers. For a vertex u in a set X, we may refer to X as the label of u on this level. The sizes of these sets are fixed, but the vertices are otherwise divided between them at random. We have the following disjoint subset of vertices:

- A^j_i for $i \in [r]$ and $j \in \{1, 2\}$, each containing N_ℓ vertices.
- B^j_i for $i \in [r]$ and $j \in \{1, 2\}$ each containing N_ℓ vertices.

- D_i^j *(the dummy vertices)* for $i \in [r]$ and $1 \le j \le 4$, each containing ζN_ℓ vertices.
- S^j *(the special vertices)* for $j \in \{1,2\}$ each containing N_ℓ vertices.

We define A_i equal to $A_i^1 \cup A_i^2$, and similarly, $B_i = B_i^1 \cup B_i^2$, $S = S^1 \cup S^2$, and $D_i = D_i^1 \cup D_i^2 \cup D_i^3 \cup D_i^4$.

The structure of the edges is somewhat similar to the base case. Aside from the gadgets to the dummies, the main difference is that the sparse gadgets are replaced with instances of $\mathcal{D}^{\ell-1}_{\mathsf{YES}}$ and $\mathcal{D}^{\ell-1}_{\mathsf{NO}}$. For two vertex sets X and Y of size $N_\ell = n_{\ell-1}/\zeta$, we use "drawing $1/\zeta$ disjoint graphs from $\mathcal{D}^{\ell-1}_{\mathsf{YES}}$ between X and Y" to refer to the following procedure: (1) divide the vertices of X into $1/\zeta$ sets $X_1, \ldots, X_{1/\zeta}$ of equal size at random, similarly for Y, (2) for each i, draw a graph from $\mathcal{D}^{\ell-1}_{\mathsf{YES}}$ and randomly map the vertices of one part to X_i and the vertices of the other part to Y_i, and (3) construct the corresponding gadgets between X_i and Y_i according to the graph drawn from $\mathcal{D}^{\ell-1}_{\mathsf{YES}}$

First, we describe the parts that are the same in $\mathcal{D}^{\ell}_{\mathsf{YES}}$ and $\mathcal{D}^{\ell}_{\mathsf{NO}}$. We recursively define the following parts:

- draw $1/\zeta$ disjoint graphs from $\mathcal{D}^{\ell-1}_{\mathsf{YES}}$ between A_i^j and B_{i+1}^j for each $1 \le i < r$ and $j \in \{1,2\}$.
- draw $1/\zeta$ disjoint graphs from $\mathcal{D}^{\ell-1}_{\mathsf{YES}}$ between B_1^j and S^j for each $j \in \{1,2\}$.
- draw one graph from $\mathcal{D}^{\ell-1}_{\mathsf{YES}}$ between D_i^j and D_i^{5-j} for each $1 \le i \le r$ and $j \in \{1,2\}$.

There are dense gadgets between A_i^j and B_i^j for every layer $i \in [r]$.

$$P(B_i^j, A_i^j) = \frac{d_\ell}{N_\ell} \qquad \forall \quad j \in \{1,2\}, \quad 1 \le i \le r$$

We also define the following gadgets to the dummy vertices. Each vertex set A_i^j or B_i^j has gadgets to the dummies of levels up to i, and there are some gadgets between the dummy vertices.

$$P(B_i^j, D_k^{j'}) = \frac{\gamma d_\ell}{2\zeta N_\ell} \qquad \forall \quad j{\in}\{1,2\}, \quad j'{\in}\{j,j{+}2\}, \quad 1{<}i{\le}r, \quad k{<}i,$$

$$P(B_i^j, D_i^{j'}) = \frac{(r-i+1)\gamma d_\ell}{2\zeta N_\ell} \qquad \forall \quad j{\in}\{1,2\}, \quad j'{\in}\{j,j{+}2\}, \quad 1{\le}i{\le}r,$$

$$P(A_i^j, D_k^{j'}) = \frac{\gamma d_\ell}{2\zeta N_\ell} \qquad \forall \quad j{\in}\{1,2\}, \quad j'{\in}\{3{-}j,5{-}j\}, \quad 1{<}i{\le}r, \quad k{<}i,$$

$$P(A_i^j, D_i^{j'}) = \frac{(r-i+1)\gamma d_\ell}{2\zeta N_\ell} \qquad \forall \quad j{\in}\{1,2\}, \quad j'{\in}\{3{-}j,5{-}j\}, \quad 1{\le}i{\le}r,$$

$$P(D_i^j, D_k^{j'}) = \frac{\gamma d_\ell}{2\zeta N_\ell} \qquad \forall \quad 1{\le}i{\le}r, \quad 1{\le}k{\le}r, \quad i{\neq}k, \quad j{\in}\{1,3\}, \quad j'{\in}\{2,4\}$$

$$P(D_i^j, D_i^{j+1}) = \frac{d_\ell + \gamma d_\ell - (2r-2i+1)\gamma d_\ell/\zeta}{\zeta N_\ell} \qquad \forall \quad 1{\le}i{\le}r, \quad j{\in}\{1,3\}$$

Finally, we present the gadgets that are different in $\mathcal{D}^{\ell}_{\mathsf{YES}}$ and $\mathcal{D}^{\ell}_{\mathsf{NO}}$. We have the following gadget **only** in $\mathcal{D}^{\ell}_{\mathsf{YES}}$:

- $1/\zeta$ disjoint graphs drawn from $\mathcal{D}^{\ell-1}_{\mathsf{YES}}$ between A_r^1 and A_r^2.

On the other hand, we have the following gadget **only** in $\mathcal{D}^{\ell}_{\mathsf{NO}}$:

- $1/\zeta$ disjoint graphs drawn from $\mathcal{D}^{\ell-1}_{\mathsf{NO}}$ between A^1_r and A^2_r.

Claim 4.5.10. *Let $\Phi^{\ell}_{\mathsf{YES}}$ and $\Phi^{\ell}_{\mathsf{NO}}$ be the gadgets introduced on level ℓ, and Φ^{ℓ} be either one of them.*

(i) $d_{\Phi^\ell}(v) = 0$, *for* $v \in S$,

(ii) $d_{\Phi^\ell}(v) = d_\ell + r\gamma d_\ell$, *for* $v \notin S$.

Proof. A gadget between vertex sets X and Y with parameter p contributes $p\,|Y|$ to the expected degree of each vertex in X. Observe that the non-recursive part is the same for the $\mathcal{D}^{\ell}_{\mathsf{YES}}$ and $\mathcal{D}^{\ell}_{\mathsf{NO}}$, and consists only of the gadgets involving the dummies and the gadgets between A^j_i and B^j_i. (i) holds because there are no non-recursive gadgets involving S. For the proof of (ii), we examine the expected degree of vertices in each vertex set separately.

- $v \in B^j_i$ for $i \in [r]$ and $j \in \{1,2\}$:

$$\begin{aligned} d_{\Phi^\ell}(v) &= P(B^j_i, A^j_i)\cdot\left|A^j_i\right| + P(B^j_i, D^j_i)\cdot\left|D^j_i\right| + P(B^j_i, D^{j+2}_i)\cdot\left|D^{j+2}_i\right| \\ &\quad + \sum_{k=1}^{i-1} P(B^j_i, D^j_k)\cdot\left|D^j_k\right| + P(B^j_i, D^{j+2}_k)\cdot\left|D^{j+2}_k\right| \\ &= d_\ell + \frac{(r-i+1)\gamma d_\ell}{2} + \frac{(r-i+1)\gamma d_\ell}{2} + \frac{(i-1)\gamma d_\ell}{2} + \frac{(i-1)\gamma d_\ell}{2} \\ &= d_\ell + r\gamma d_\ell \end{aligned}$$

- $v \in A^j_i$ for $i \in [r]$ and $j \in \{1,2\}$:

$$\begin{aligned} d_{\Phi^\ell}(v) &= P(A^j_i, B^j_i)\cdot\left|B^j_i\right| + P(A^j_i, D^{3-j}_i)\cdot\left|D^{3-j}_i\right| + P(A^j_i, D^{5-j}_i)\cdot\left|D^{5-j}_i\right| \\ &\quad + \sum_{k=1}^{i-1} P(A^j_i, D^{3-j}_k)\cdot\left|D^{3-j}_k\right| + P(A^j_i, D^{5-j}_k)\cdot\left|D^{5-j}_k\right| \\ &= d_\ell + \frac{(r-i+1)\gamma d_\ell}{2} + \frac{(r-i+1)\gamma d_\ell}{2} + \frac{(i-1)\gamma d_\ell}{2} + \frac{(i-1)\gamma d_\ell}{2} \\ &= d_\ell + r\gamma d_\ell \end{aligned}$$

- $v \in D^j_i$ for $i \in [r]$ and $j \in \{1,2,3,4\}$: We examine $j = 1$. The other cases follow similarly.

$$\begin{aligned} d_{\Phi^\ell}(v) &= P(D^1_i, D^2_i)\cdot\left|D^2_i\right| \\ &\quad + \sum_{k\neq i} P(D^1_i, D^2_k)\cdot\left|D^2_k\right| + P(D^1_i, D^4_k)\cdot\left|D^4_k\right| \\ &\quad + P(D^1_i, B^1_i)\cdot\left|B^1_i\right| + P(D^1_i, A^2_i)\cdot\left|A^2_i\right| \\ &\quad + \sum_{k=i+1}^{r} P(D^1_i, B^1_k)\cdot\left|B^1_k\right| + P(D^1_i, A^2_k)\cdot\left|A^2_k\right| \\ &= d_\ell + \gamma d_\ell - \frac{(2r-2i+1)\gamma d_\ell}{\zeta} \end{aligned}$$

$$
\begin{aligned}
&+ (r-1)\frac{\gamma d_\ell}{2} + (r-1)\frac{\gamma d_\ell}{2} \\
&+ \frac{(r-i+1)\gamma d_\ell}{2\zeta} + \frac{(r-i+1)\gamma d_\ell}{2\zeta} \\
&+ \frac{(r-i)\gamma d_\ell}{2\zeta} + \frac{(r-i)\gamma d_\ell}{2\zeta} \\
&= d_\ell + r\gamma d_\ell
\end{aligned}
$$

□

Observation 4.5.11. *Any multigraph drawn from $\mathcal{D}^\ell$ is bipartite.*

Proof. Observe that all the recursively produced parts are bipartite. Therefore, the following is a bipartition of the graph:

(1) $\left(\bigcup_{i=1}^r A_i^1\right) \cup \left(\bigcup_{i=1}^r B_i^2\right) \cup \left(\bigcup_{i=1}^r D_i^1\right) \cup \left(\bigcup_{i=1}^r D_i^3\right) \cup S^1$, and

(2) $\left(\bigcup_{i=1}^r A_i^2\right) \cup \left(\bigcup_{i=1}^r B_i^1\right) \cup \left(\bigcup_{i=1}^r D_i^2\right) \cup \left(\bigcup_{i=1}^r D_i^4\right) \cup S^2$. □

Lemma 4.5.12. *In the multigraph construction, the size of the maximum matching in $\mathcal{D}^\ell_{\mathsf{YES}}$ and $\mathcal{D}^\ell_{\mathsf{NO}}$ is characterized as follows:*

(i) For $G \sim \mathcal{D}^\ell_{\mathsf{YES}}$, it holds $\mu(G) = n_\ell/2$ with high probability.

(ii) If $G \sim \mathcal{D}^\ell_{\mathsf{NO}}$, then $\mu(G) \le n_\ell/2 - N_1/2$.

Proof. For (i), we prove $G \sim \mathcal{D}^\ell_{\mathsf{YES}}$ has a perfect matching with high probability by induction on ℓ. The base case, $\ell = 1$, has been proven in Lemma 4.5.9. By the induction hypothesis, there exists a perfect matching in each recursively drawn graph from $\mathcal{D}^{\ell-1}_{\mathsf{YES}}$ with high probability. Therefore, by the union bound, there exists a perfect matching between B_1^j and S^j for $j \in \{1,2\}$; between A_i^j and B_{i+1}^j for $1 \le i < r$ and $j \in \{1,2\}$; between A_r^1 and A_r^2; and between D_i^j and D_i^{5-j} for $j \in \{1,2\}$. Putting these matchings together results in a perfect matching for the whole graph. Observe that using the union bound in the induction is valid because over the course of the induction up to $\ell = L$, we recursively take the union bound over $\left(\frac{8r+2}{\zeta}\right)^L = O_\varepsilon(1)$ events.

For (ii), we prove by induction on ℓ that $G \sim \mathcal{D}^\ell_{\mathsf{NO}}$ has a vertex cover of size $n_\ell/2 - N_1/2$. The base case, $\ell = 1$, has been proven in Lemma 4.5.9. Assuming the induction hypothesis for $\ell - 1$. There exists a vertex cover X of size $\frac{1}{\zeta}(n_{\ell-1} - N_1/2)$ for the induced graph between A_r^1 and A_r^2. The following set is a vertex cover:

$$
C := X \cup \left(\bigcup_{i \le r, j \in \{1,2\}} B_i^j\right) \cup \left(\bigcup_{i \le r, j \in \{1,2,3,4\}} D_i^j\right),
$$

as any edge outside of $G[A_r^1; A_r^2]$ is adjacent to B or D. Now, we calculate the size of this vertex cover:

$$
\begin{aligned}
|C| &= |X| \cup \sum_{i \le r, j \in \{1,2\}} \left|B_i^j\right| + \sum_{i \le r, j \in \{1,2,3,4\}} \left|D_i^j\right| \\
&= \frac{1}{\zeta}(n_{\ell-1}/2 - N_1/2) + 2rN_\ell + 4r\zeta N_\ell \\
&\le (2r + 1/2 + 4r\zeta)N_\ell - N_1/2 \\
&\le n_\ell/2 - N_1/2
\end{aligned}
$$

where the last inequality follows from $4r\zeta \leq \frac{1}{2}$ and $n_\ell/2 = (2r + 1 + 2r\zeta)N_\ell$. This concludes the proof of (ii) and the claim. □

Lemma 4.5.13. *Let $\varepsilon = (\delta/10)^{300/\delta^2}$. Any algorithm that estimates the size of the maximum matching of the multigraph that is drawn from the input distribution with εn additive error must be able to distinguish whether it belongs to $\mathcal{D}_{\mathsf{YES}}$ or $\mathcal{D}_{\mathsf{NO}}$.*

Proof. By Lemma 4.5.12, the maximum matching size in $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$ differ by $N_1/2$. Therefore, any algorithm that estimates the size of the maximum matching within a $(N_1/4)$-additive error, must distinguish between the two distributions. Now, we express N_1 in terms of $n = n_L$.

For the first level, we have

$$n_1 = |A| + |B| + |D| + |S| = (2r - 2\xi)N_1 + 2rN_1 + 2r\zeta N_1 + 2N_1 = (4r - 2r\xi + 2r\zeta + 2)N_1.$$

For the next levels $\ell > 1$, it holds

$$N_\ell = n_{\ell-1}/\zeta \qquad \text{and} \qquad n_\ell = 2rN_\ell + 2rN_\ell + 4r\zeta N_\ell + 2N_\ell = (4r + 4r\zeta + 2)N_\ell$$

Therefore,

$$n_\ell = \frac{(4r + 4r\zeta + 2)}{\zeta} n_{\ell-1}.$$

Then, it can be shown by induction

$$n_L = \left(\frac{(4r + 4r\zeta + 2)}{\zeta}\right)^{L-1} n_1 = \left(\frac{(4r + 4r\zeta + 2)}{\zeta}\right)^{L-1} (4r - 2r\xi + 2r\zeta + 2)N_1.$$

Plugging in the parameters

$$L = 4/\delta, \qquad r = (10/\delta)^{L+1}, \qquad \zeta = 1/r^2, \qquad \text{and} \qquad \xi = 1/r^4,$$

we get

$$n_L \leq (10r^3)^L N_1 \leq (10/\delta)^{3L^2} N_1 \leq (10/\delta)^{300/\delta^2} N_1.$$

Putting everything together, if we let $\varepsilon = (\delta/10)^{300/\delta^2}$, then any algorithm that computes an εn-additive approximation, computes an $(N_1/4)$-additive approximation, and thus distinguishes between $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$. □

Constructing the Multigraph

In this subsection, we demonstrate how to construct the multigraph using the gadgets introduced earlier. We first prove the following claim, which is essential for the construction of the multigraph.

Claim 4.5.14. *For every pair of vertices u and v, considering all levels, there exists at most one gadget between vertex sets X and Y such that $u \in X$ and $v \in Y$.*

Proof. The claim is proved by induction on ℓ. For the base case $\ell = 1$, the claim holds as the vertices are divided into disjoint sets, each gadget is defined between two of these sets, and there is at most one gadget between any two sets. For $\ell > 1$, the claim holds because (1) the gadgets on level ℓ are defined between disjoint vertex sets, and (2) the recursively defined gadgets of level smaller than ℓ are either between A_i^j and B_{i+1}^j, B_1^j and S^j, D_i^j and D_i^{5-j}, or A_r^1 and A_r^2. Therefore, they do not share pairs with each other or with gadgets of level ℓ. □

Given this fact, for any two vertices u and v we can define a parameter $p^{(u,v)}$, which is equal to the density parameter associated with the gadget between u and v, or zero if no such gadget exists.

Warm-up: "level-free" construction: We begin with a simplified "level-free" construction and then generalize to include levels. En route to our multigraph construction, we construct three helper multigraphs, the *ground graph*, the *labelled-ground graph*, and the *pseudo graph*. It will be helpful for the analysis that the ground and pseudo graphs are independent of the vertex labels, so their structure does not reveal any information about the labels. This scheme also helps obtain the same distribution for the degrees of the vertices, so that the algorithm cannot easily distinguish between two vertices, e.g. based on the distribution of the degrees of their neighbors.

All multigraphs are defined over the same set of vertices, and their respective edge sets satisfy the following inclusions:

$$\text{Ground edges} \supset \text{Labelled-ground edges}, \text{pseudo edges} \supset \text{Real edges}.$$

In particular, we will eventually let the real multigraph simply be the intersection of the pseudo and the labelled-ground edge sets:

$$\text{Real edges} = \text{Labelled-ground edges} \cap \text{Pseudo edges}.$$

Ground edges: The ground graph is completely deterministic. For every pair of vertices u and v, we add exactly ρn parallel ground edges between u and v (independently of their labels).

Labelled-ground edges: The labelled-ground edges are deterministic conditioned on the (random) labels of the vertices. For any pair of vertices u and v, let $p^{(u,v)}$ be density parameter of the gadget between u and v (and zero if no such gadget exists). We let $p^{(u,v)}n$ of the ground edges between u, v be labelled-ground edges. Observe that if two vertices have the same expected degree d in the gadget construction, they have the same degree dn in the labelled-ground graph.

Pseudo edges: Every ground edge is a pseudo edge independently with probability $1/n$.

Full multigraph construction with levels: Our actual construction closely follows the level-free construction, with the modification that each labelled-ground and pseudo edge is assigned a level. Then, the level-ℓ real edges are defined as:

$$\text{Level-}\ell \text{ real edges} = \text{Level-}\ell \text{ labelled-ground edges} \cap \text{Level-}(\leq \ell) \text{ pseudo edges}.$$

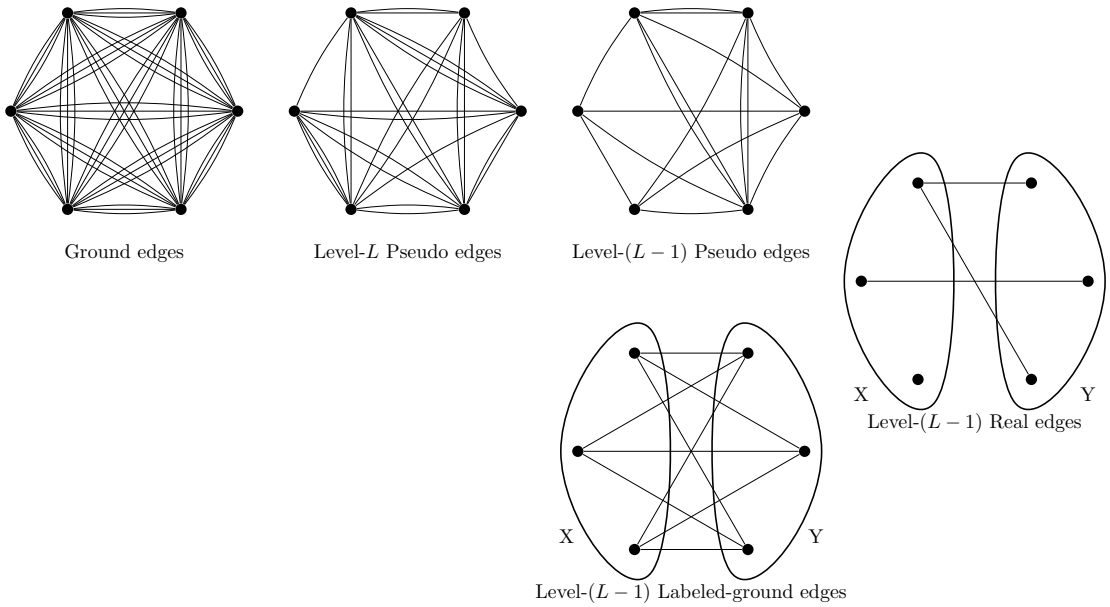


Figure 4.8: An illustration of the levelled construction for a gadget on level $L-1$. The ground edges and pseudo edges do not depend on the vertex labels. Level-L pseudo edges are a subset of the ground edges, and level-ℓ pseudo edges are a subset of the Level-$(\ell+1)$ pseudo edges. The level-ℓ Laballed-ground edges are determined based on the vertex labels and the level-ℓ gadgets between them, here the X-Y gadget on level $L-1$. The level-ℓ real edges are the intersection of level-ℓ pseudo edges and level-ℓ labelled-ground edges.

Note that we require level exactly ℓ for labelled-ground edges, but take any level $\leq \ell$ for pseudo edges.

The ground edges are not associated with any levels and are defined exactly as before, i.e. there is ρn of them between any pair. Then, for any two vertices u and v, we add $p^{(u,v)}n \cdot (\rho/\rho_\ell)$ of the ground edges between them to the level-ℓ labelled-ground edges, where ℓ is the level of the gadget defined between u and v (if any).

Finally, any ground edge is marked as a pseudo edge of level L with probability $1/n$. If it is marked as a pseudo edge of level L, then it is marked as a pseudo edge of level $L-1$ with probability ρ_{L-1}/ρ_L, and so on. More specifically, if the edge is marked as a pseudo edge of level ℓ, it will be marked as a pseudo edge of level $\ell-1$ with probability $\rho_{\ell-1}/\rho_\ell$. Observe that at this point the edge can become a real edge if it is also a level-ℓ labelled-ground edge.

We let $\deg^n_G(v)$, $\deg^p_G(v)$, and $\deg^r_G(v)$ be the number of non-pseudo edges, pseudo edge, real edges of vertex v in graph G, respectively.

Properties of the Constructed Multigraph

Claim 4.5.15. *It holds*

$$\text{Level-}\ell \text{ pseudo edges} \supset \text{Level-}(\ell-1) \text{ pseudo edges},$$

and for any ground edge, the probability of becoming a level-ℓ pseudo edge is $\rho_\ell/(n\rho)$.

Proof. The first part of the claim follows from the definition. The second part can be proven by induction on

ℓ. It holds for $\ell = L$ since $\rho_L = \rho$ and the probability of being a level-L pseudo edge is $\frac{1}{n} = \frac{\rho_L}{\rho n}$. For $\ell < L$, by the induction hypothesis, the edge becomes a level-$(\ell+1)$ pseudo edge with probability $\frac{\rho_{\ell+1}}{\rho n}$. Therefore, it becomes a level-ℓ pseudo edge with probability

$$\frac{\rho_{\ell+1}}{\rho n} \cdot \frac{\rho_\ell}{\rho_{\ell+1}} = \frac{\rho_\ell}{\rho n},$$

which concludes the proof. □

Claim 4.5.16. *For any two vertices u and v, the distribution of the number of edges between them is* $\text{Binom}\left(p^{(u,v)} n \cdot (\rho/\rho_\ell), \rho_\ell/n\rho\right)$*, where ℓ is the level of the gadget between u and v. If there is no such gadget the number of edges is simply zero.*

Proof. Recall, we added $p^{(u,v)} \cdot (\rho/\rho_\ell)$ level-ℓ labelled-ground edges between u and v. By Claim 4.5.15, each of them has a probability $\rho_\ell/(n\rho)$ of becoming a level-ℓ pseudo edge. Those edge then constitute the real edges between u and v. This concludes the proof. □

Corollary 4.5.17. *For two vertices u and v, the expected number of edges between them is $p^{(u,v)}$.*

Corollary 4.5.18. *For two vertices u and v, the probability that there exists an edge between them is at least $p^{(u,v)}/2$.*

Proof. The probability of there being no edges is:

$$(1 - \rho_\ell/n\rho)^{p^{(u,v)} n \cdot (\rho/\rho_\ell)} \leq e^{-p^{(u,v)}}.$$

Therefore, since $p^{(u,v)} \leq 1$, the probability that there is an edge is at least

$$1 - e^{-p^{(u,v)}} \geq p^{(u,v)} \left(1 - \frac{1}{e}\right) \geq p^{(u,v)}/2.$$ □

Observation 4.5.19. *Let e and e' be two different edges in the ground multigraph (possibly with the same endpoints). Then, the probability that e is not in the pseudo graph is equal to the probability that e' is not in the pseudo graph.*

Proof. The proof follows from the fact that each edge is a pseudo edge independently with probability $1/n$. □

Lemma 4.5.20. *Take a vertex u in the multigraph. Let Φ be a subset of level-ℓ gadgets, and let H be the subgraph including the ground edges associated with Φ. Then,* $\deg^r_H(u) = \text{Binom}(d_\Phi(u) \cdot n\rho/\rho_\ell, \rho_\ell/n\rho)$*. That is, the distribution of* $\deg^r_H(u)$*, the real degree of u from edges of H, is solely determined by $d_\Phi(u)$.*

Proof. We utilize Claim 4.5.16. Let $\Phi' \subseteq \Phi$ be gadgets that have u on one side. That is $\Phi' = \{\phi_1, \phi_2, \ldots, \phi_k\}$, where gadget ϕ_i with density parameter P_i is between vertex sets X_i and Y_i, and $u \in X_i$. Then, we have (below we use $\deg^r_H(u, Y)$ for a vertex set Y, to denote the number of real edges of H between u and Y):

$$\begin{aligned}\deg^r_H(u) &= \sum_i \deg^r_H(u, Y_i) \\ &= \sum_i \sum_{v \in Y_i} \deg^r_H(u, v)\end{aligned}$$

$$
\begin{aligned}
&= \sum_i \sum_{v \in Y_i} \text{Binom}(p^{(u,v)} \cdot n\rho/\rho_\ell, \rho_\ell/n\rho) && \text{(by Claim 4.5.16)} \\
&= \sum_i \text{Binom}(|Y_i|\, P_i \cdot n\rho/\rho_\ell, \rho_\ell/n\rho) \\
&= \sum_i \text{Binom}(d_{\phi_i}(u) \cdot n\rho/\rho_\ell, \rho_\ell/n\rho) && (d_{\phi_i}(u) = |Y_i|\, P_i) \\
&= \text{Binom}(\sum_i d_{\phi_i}(u) \cdot n\rho/\rho_\ell, \rho_\ell/n\rho) \\
&= \text{Binom}(d_\Phi(u) \cdot n\rho/\rho_\ell, \rho_\ell/n\rho) && \square
\end{aligned}
$$

Corollary 4.5.21. *Take a vertex u in the multigraph, and let H be the subgraph of ground edges associated with Φ^ℓ, the gadgets of level ℓ. Then, the distribution of $\deg^r_H(u)$ is solely determined by $d_{\Phi^\ell}(u)$.*

Proof. Follows from a direct application of *Lemma* 4.5.20 for $\Phi = \Phi^\ell$. □

Corollary 4.5.22. *Take a vertex u in the multigraph, and let H be the subgraph of ground edges associated with Φ^ℓ, the gadgets of level ℓ. Then, the distribution of the number of non-real pseudo edges of H adjacent to vertex u, i.e. $\deg^p_H(u) - \deg^r_H(u)$, is solely determined by $d_{\Phi^\ell}(u)$.*

Proof. The distribution of $\deg^r_H(u)$ is determined by $d_{\Phi^\ell}(u)$, and the distribution of $\deg^p_H(u)$ is determined by ℓ and is independent of the labels and gadgets. □

To wrap the section up, we prove a lemma that states the part of the graph queried by the algorithm is essentially a simple graph.

Lemma 4.5.23. *Let $\mathcal{A}$ be an algorithm that makes $O(n^{2-\delta})$ queries. Then, with high probability, it does not query any pair (u, v) such that there is more than one pseudo edge between them in G.*

Proof. Take any query (u, v) that the algorithm makes. Recall that the number of pseudo edges between them has distribution $\text{Binom}(\rho n, \frac{1}{n})$, and is independent of everything else in the graph (including the parts the algorithm has queried). Therefore, the probability that (u, v) has at most one pseudo edge is

$$
\begin{aligned}
\left(1 - \frac{1}{n}\right)^{\rho n} + \rho n \cdot \frac{1}{n}\left(1 - \frac{1}{n}\right)^{\rho n - 1} &\geq (1 + \rho)\left(1 - \frac{1}{n}\right)^{\rho n} \\
&\geq (1 + \rho)\left(1 - \rho + \frac{\rho^2}{2} - O(\rho/n)\right) \\
&= 1 - O(\rho^2).
\end{aligned}
$$

It follows that the probability of having more than one pseudo edge is at most $O(\rho^2)$. By taking the union bound over all the $O(n^{2-\delta})$ queries, the probability that the algorithm queries a pair with more than one pseudo edge is at most

$$O(\rho^2 \cdot n^{2-\delta}) = O(n^{\delta/5-2} \cdot n^{2-\delta}) = O(n^{-4\delta/5}) = o(1). \qquad \square$$

Multigraph Query Model and Reduction to Simple Graphs

Recall that the final algorithm aims to approximate the maximum matching in the real multigraph. A natural model for adjacency matrix queries for multigraphs is querying the number of edges between two

vertices u and v. We make the algorithm stronger as follows: Upon querying a pair (u, v), the algorithm finds out (1) how many pseudo edges there are between them, and (2) how many of them are real edges. In later sections, we prove the lower bound for this model. Note that solving the problem in this model is only easier than the natural multigraph model.

Remark 9. *In the proof of the lower bound, due to Lemma 4.5.23, we can assume that the algorithm upon querying a pair (u, v) either (1) sees no edges between them, (2) sees a pseudo edge that is not a real edge, or (3) sees a pseudo edge that is also a real edge.*

Furthermore, the problem is not easier for simple graphs. Consider the following reduction. Given an algorithm $\mathcal{A}$ for simple graphs, we obtain an algorithm for multigraphs: Run $\mathcal{A}$ on the multigraph; if a query between two vertices u and v reports more than one edge, simply report one edge to $\mathcal{A}$. This reduction essentially removes the multiplicity of the edges. Therefore, it does not change the size of the maximum matching, and the output of $\mathcal{A}$ is still an εn-additive approximation.

In the following sections, we prove that if the algorithm makes $\Omega(n^{2-\delta})$ queries, it cannot distinguish between input graphs drawn from $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$. Our proof is based on a coupling between $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$. The ground edges that the algorithm finds which are not pseudo or real edge do not affect the algorithm's ability to distinguish between the two cases Observation 4.5.19. This is because the probability that a ground edge is a non-pseudo edge is independent of the labels of the vertices by Observation 4.5.19. Consequently, such edges provide no useful information for distinguishing between the two distributions. As a result, we focus on coupling the pseudo and real edges between $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$. So for the rest of the proof, we assume that ground edges that are not marked as pseudo do not exist in the graph and when we refer to edges, we mean pseudo and real edges. Whenever we want to mention a ground edge, we explicitly mention that. Moreover, for the rest of the proof when we refer to level ℓ edge, we mean pseudo or real edges that belong to level ℓ or lower. Also, when we use A_i (resp. B_i, and D_i) without superscript, we mean A_i^1 and A_i^2 (the same for both B_i and D_i). Further, when in the proof we are considering level ℓ, by label of the vertex we mean the label of the vertex in the same level.

4.5.4 Losing Advantage in the Highest Level

The key distinction between graphs drawn from $\mathcal{D}^L_{\mathsf{YES}}$ and $\mathcal{D}^L_{\mathsf{NO}}$ lies in the subgraph between A_r^1 and A_r^2 at the highest level. Specifically, in $\mathcal{D}^L_{\mathsf{YES}}$, this subgraph is sampled from $\mathcal{D}^{L-1}_{\mathsf{YES}}$, while in ${\mathcal{D}_{\mathsf{NO}}}^L$, it is sampled from $\mathcal{D}^{L-1}_{\mathsf{NO}}$. Therefore, any algorithm aiming to differentiate between $\mathcal{D}^L_{\mathsf{YES}}$ and $\mathcal{D}^L_{\mathsf{NO}}$ must detect differences within this subgraph.

In this subsection, we derive an upper bound on the number of level $L-1$ edges the algorithm can recognize as belonging to this critical subgraph. To formalize this, we introduce the concept of the distinguishability of an edge within the subgraph between A_r^1 and A_r^2.

When the algorithm queries a typical edge, due to our choices of d_L and d_{L-1}, parameters that control the degrees of vertices at levels L and $L-1$, the probability that this edge belongs to the subgraph between A_r^1 and A_r^2 is approximately $d_{L-1}/d_L = n^{\sigma_{L-1}-\sigma_L}$ if the edge is a real edge. Also, for a pseudo edge, because of our choices of ρ_L and ρ_{L-1}, the probability that this pseudo edge belongs to the subgraph at the lower level is approximately $\rho_{L-1}/\rho_L = n^{\sigma_{L-1}-\sigma_L}$.

We say an edge is distinguishable when the algorithm can infer a bias in this probability, conditioned on the queried subgraph.

Definition 4.5.24 (p_e^{inner} and Distinguishability of an Edge)**.** *Let e be a real edge or pseudo edge queried by the algorithm, and let p_e^{inner} denote the probability that if e is a level $L-1$ edge that belongs to subgraph between A_r^1 and A_r^2, conditioned on all queries made by the algorithm so far and assuming either input distribution. We say edge e is distinguishable if $p_e^{inner} > 10n^{\sigma_{L-1}-\sigma_L}$.*

Lemma 4.5.25. *With high probability, the total number of edges that the algorithm discovers is at most $O(n^{1-\delta+\sigma_L})$.*

Proof. Since there are ρn ground edges between every two vertices and the algorithm makes $O(n^{2-\delta})$ pair queries, the total number of ground edges the algorithm queries is $O(\rho n^{3-\delta})$. Moreover, each of these ground edges is classified as either a pseudo or real edge with an independent probability of at most $1/n$, as they must be marked as pseudo edges of level L, which occurs with probability $1/n$. Therefore, in expectation, the algorithm finds $O(\rho n^{2-\delta})$ such edges.

Let X_i be the indicator random variable for the event that the i-th queried ground edge is a pseudo or real edge. Let $X = \sum X_i$ denote the total number of such edges found by the algorithm. We have $E[X_i] \le 1/n$, and $E[X] \le O(\rho n^{2-\delta})$. Furthermore, the random variables X_i are independent. Therefore, applying the Chernoff bound, we have:

$$\Pr\left[|X - E[X]| \ge 2\sqrt{E[X]\log n}\right] \le 2\exp\left(-\frac{(2\sqrt{E[X]\log n})^2}{3E[X]}\right) < \frac{1}{n}.$$

This implies that with probability at least $1 - 1/n$, the total number of pseudo or real edges discovered by the algorithm is $O(\rho n^{2-\delta})$. Plugging $\rho = n^{\sigma_L - 1}$ completes the proof. □

Definition 4.5.26 (Direction of an Edge)**.** *Let (u, v) be an edge queried by the algorithm. Suppose the algorithm has already discovered some edges of u, but has not discovered any edges of v. When we refer to the direction of the edge (u, v), we mean that the edge is considered to go from u to v. If the algorithm has already discovered edges for both u and v, or has not discovered any edges for either, we assign the direction of the edge randomly.*

Claim 4.5.27. *Each vertex in the queried subgraph has at most $3\sqrt{\log n}$ indegree with high probability.*

Proof. Let v be an arbitrary vertex. Suppose that v has at least one incoming edge. Also, let $\hat{V}$ be the set of vertices in the graph for which the algorithm finds at least one edge. According to the way we direct the edges in Definition 4.5.26, the rest of the incoming edges of v are queried at the time when the other endpoint already has an existing edge. Therefore, there are $|\hat{V}|$ possible pairs between $\hat{V}$ and v, each containing ρn ground edges, and each being a pseudo or real edge with probability at most $1/n$. Thus, the expected number of edges between $\hat{V}$ and v is:

$$E[X] = |\hat{V}| \cdot \rho = \widetilde{O}(n^{2\sigma_L - \delta}),$$

since $|\hat{V}| = O(n^{1-\delta+\sigma_L})$ by Lemma 4.5.25. Let X_i be the indicator random variable for the event that the i-th ground edge between $\hat{V}$ and v is a pseudo or real edge, and let $X = \sum X_i$ denote the total number of such edges. We have $E[X] = O(n^{2\sigma_L - \delta}) < 1$ for large enough n. Furthermore, the random variables X_i are

independent. Applying the Chernoff bound, we obtain:

$$\Pr\left[|X - E[X]| \geq 3\sqrt{E[X]\log n}\right] \leq 2\exp\left(-\frac{(3\sqrt{E[X]\log n})^2}{3E[X]}\right) < \frac{1}{n^2}.$$

Thus, with probability at least $1 - 1/n^2$, the total number of incoming pseudo or real edges to v is $3\sqrt{\log n}$. Applying the union bound over all vertices concludes the proof. □

Definition 4.5.28 (Shallow Subgraph)**.** *For a vertex v, we define v's shallow subgraph as the set of vertices that are reachable from v via directed paths of length at most $10\log n$ in the queried subgraph. We denote v's shallow subgraph by $T(v)$.*

Lemma 4.5.29. *Each vertex belongs to at most $\widetilde{O}(1)$ shallow subgraphs with high probability.*

Proof. Let v be an arbitrary vertex in the graph. Let V_i be the set of vertices at a distance i from v for $i \in [10\log n]$ using edges in the reverse direction. We will show that, with high probability, $|V_i| \leq 3i\sqrt{\log n}$ using induction.

For the base case of the induction, $i = 1$, the claim holds by Claim 4.5.27. Suppose the claim holds for all $i' < i$. Suppose that each vertex in V_{i-1} has at least one incoming edge. Note that this only increases the size of V_i. Let $u \in V_{i-1}$. Also, let $\hat{V}$ be the set of vertices that the algorithm has found at least one edge in the queried subgraph. If u has more than one incoming edge, it should be between a vertex that already has an edge because of the way we defined the direction of edges in Definition 4.5.26. Therefore, there are $|\hat{V}|$ possible pairs between $\hat{V}$ and u, each containing ρn ground edges, and each being a pseudo or real edge with probability at most $1/n$. Thus, the expected number of edges between $\hat{V}$ and u is $|\hat{V}| \cdot \rho = O(n^{2\sigma_L - \delta})$, since $|\hat{V}| = O(n^{1-\delta+\sigma_L})$ by Lemma 4.5.25. Additionally, $|V_{i-1}| \leq 3(i-1)\sqrt{\log n}$ by induction hypothesis. Hence, the expected number of edges between $\hat{V}$ and V_{i-1} is $\widetilde{O}(n^{2\sigma_L - \delta})$.

Let X_i be the indicator random variable for the event that the i-th ground edge between $\hat{V}$ and V_{i-1} is a pseudo or real edge, and let $X = \sum X_i$ denote the total number of such edges. We have $E[X] = \widetilde{O}(n^{2\sigma_L - \delta}) < 1$ for large enough n. Furthermore, the random variables X_i are independent. Applying the Chernoff bound, we obtain:

$$\Pr\left[|X - E[X]| \geq 3\sqrt{E[X]\log n}\right] \leq 2\exp\left(-\frac{(3\sqrt{E[X]\log n})^2}{3E[X]}\right) < \frac{1}{n^2}.$$

Thus, with probability at least $1 - 1/n^2$, the total number of incoming pseudo or real edges to v is $3\sqrt{\log n}$. Moreover, we assume that each vertex in V_{i-1} contains at least one incoming edge. Hence, we have $|V_i| \leq |V_{i-1}| + 3\sqrt{\log n} =\leq 3i\sqrt{\log n}$ which completes the induction step. Therefore, we have $|V_i| \leq 3i\sqrt{\log n}$ for all $i \leq 10\log n$. As a result, the total number of shallow subgraphs that contain v is bounded by

$$1 + \sum_{i=1}^{10\log n} |V_i| \leq 1 + \sum_{i=1}^{10\log n} 3i\sqrt{\log n} \leq (10\log n) \cdot (30\log n\sqrt{\log n}),$$

which completes the proof. □

Corollary 4.5.30. *Each edge belongs to at most $\widetilde{O}(1)$ shallow subgraphs with high probability.*

Proof. For directed edge (u, v), since u is in at most $\widetilde{O}(1)$ shallow subgraphs by Lemma 4.5.29, then (u, v) is in at most $\widetilde{O}(1)$ shallow subgraphs. □

Definition 4.5.31 (Spoiler Vertex)**.** *A vertex u is called a spoiler vertex if an edge (u, v) is discovered by the algorithm at a time when both u and v already have a non-zero degree.*

Claim 4.5.32. *There are at most $O(n^{1-2\delta+3\sigma_L})$ spoiler vertices with high probability.*

Proof. Note that each query between two vertices that already have non-zero degree in the queried subgraph, resulting in a pseudo or real edge, creates two spoiler vertices. By Lemma 4.5.25, there are at most $O(n^{1-\delta+\sigma_L})$ vertices with non-zero degree. Therefore, the total number of such vertex pairs is at most $O(n^{2-2\delta+2\sigma_L})$. If the algorithm queries all these pairs, the total number of ground edges queried is at most $O(\rho n^{3-2\delta+2\sigma_L})$, where each forms a pseudo or real edge with probability at most $1/n$. Consequently, the expected number of edges discovered by the algorithm between pairs of vertices with a non-zero degree is at most $O(\rho n^{2-2\delta+2\sigma_L}) = O(n^{1-2\delta+3\sigma_L})$. Therefore, using Chernoff bound, we can show that with high probability, there are at most $O(n^{1-2\delta+3\sigma_L})$ spoiler vertices. □

Definition 4.5.33 (Spoiled Vertex)**.** *A vertex v is called a spoiled vertex if its shallow subgraph contains any of the following:*

(i) a spoiler vertex; or

(ii) at least $n^{\delta-2\sigma_L}$ vertices.

Observation 4.5.34. *Let v be a vertex that is not spoiled. Then, the shallow subgraph of v forms a rooted tree with at most $n^{\delta-2\sigma_L}$ vertices. Additionally, for every edge (u, w) in the shallow subgraph of v, when the algorithm queries this edge, vertex w is a singleton.*

Proof. When the algorithm encounters an edge (u, w) in the shallow subgraph of v, vertex w must be a singleton, as per Definition 4.5.31 and Definition 4.5.33. This ensures that the shallow subgraph of v is indeed a rooted tree. □

Lemma 4.5.35. *There are at most $O(n^{1-2\delta+4\sigma_L})$ spoiled vertices with high probability.*

Proof. According to Definition 4.5.33, at least one of the two conditions is required for a vertex to be spoiled. For condition (i), by Claim 4.5.32, there are $O(n^{1-2\delta+3\sigma_L})$ spoiler vertices. Moreover, each vertex is in $\widetilde{O}(1)$ shallow subgraphs by Lemma 4.5.29, which implies that there are at most $O(n^{1-2\delta+4\sigma_L})$ vertices that have the condition (i) for large enough n.

For condition (ii), by Corollary 4.5.30, each edge in the queried subgraph appears in $\widetilde{O}(1)$ shallow subgraphs. Thus, for non-isolated vertices in the queried subgraph, by Lemma 4.5.25, we have $\sum_v |T(v)| \leq \widetilde{O}(n^{1-\delta+\sigma_L})$. Consequently, the total number of vertices whose shallow subgraph contains more than $n^{\delta-2\sigma_L}$ vertices is at most $O(n^{1-2\delta+4\sigma_L})$. □

Definition 4.5.36 (Spoiled Edge)**.** *Let (u, v) be a directed edge. We say (u, v) is a spoiled edge if $u \in A_r$ and at least one of the following condition holds:*

(i) v is a spoiled vertex; or

(ii) u has at least $n^{\sigma_L}/3$ spoiled neighbors in the queried subgraph.

Lemma 4.5.37. *There are at most $O(n^{1-2\delta+5\sigma_L})$ spoiled edges with high probability.*

Proof. Each vertex in the queried subgraph has an indegree of $\widetilde{O}(1)$ with high probability by Claim 4.5.27. Also, by Lemma 4.5.35, we have at most $O(n^{1-2\delta+4\sigma_L})$ spoiled vertices. Therefore, there are at most $O(n^{1-2\delta+5\sigma_L})$ edges that satisfy condition (i) of Definition 4.5.36.

Further, if a vertex u satisfies condition (ii), it must have at least $n^{\sigma L}/4$ outgoing edges (u, w) where w is spoiled. However, from Claim 4.5.27, each spoiled vertex has indegree of $\widetilde{O}(1)$. Also, the total number of spoiled vertices is $O(n^{1-2\delta+4\sigma L})$ which implies that the number of edges that satisfy condition (ii) of Definition 4.5.36 is at most $O(n^{1-2\delta+5\sigma L})$. □

We defer the proof of the following lemma to a later section, as it is involved, lengthy, and mostly independent of the flow of this section.

Lemma 4.5.38. *Let v be a vertex that is not spoiled and belongs to $\{A_r, B_r, D_r\}$. Let $\mathcal{L}(v)$ and $\mathcal{L}'(v)$ represent an arbitrary label for v from $\{A_r, B_r, D_r\}$ and the entire queried subgraph all available labels at level L, excluding the shallow subgraph of v and isolated vertices. Then, we have:*

$$\Pr[T(v) \mid \mathcal{L}(v)] \leq \left(1 + O(n^{\sigma_L - \delta})\right)^{|T(v)|} \cdot \Pr[T(v) \mid \mathcal{L}'(v)].$$

Lemma 4.5.39. *Let e be a directed edge that is not spoiled, then it holds that $p_e^{inner} \leq 10n^{\sigma_{L-1}-\sigma_L}$.*

Proof. Let $e = (u, v)$ be the directed edge (directed from u to v). First, if $u \notin A_r$, we have $p_e^{\text{inner}} = 0$. Also, if e is a pseudo edge, by the construction, the probability that we mark e as level $L-1$ is $\rho_{L-1}/\rho_L = n^{\sigma_{L-1}-\sigma_L}$ independent at random from other edges.

Thus, suppose that $u \in A_r$ and e is a real edge for the rest of the proof. According to the construction, u has at least $6n^{\sigma_L}/7$ real edges such that the other endpoint has label $A_r \cup B_r$. Also, u has at most $\widetilde{O}(1)$ incoming edge by Claim 4.5.27. Since e is not a spoiled edge, u has at most $n^{\sigma_L}/3$ neighbors in the queried subgraph that are spoiled. Let V_u be the set of neighbors of u using real edges in the original graph such that their label is $A_r \cup B_r$ and either they are singleton or direct children of u in the queried subgraph. By the above argument, we have $|V_u| \geq n^{\sigma_L}/2$. Also, note that $v \in V_u$. Now we provide an upper bound on the probability that e belongs to level $L-1$, or in other words, $v \in A_r$. We prove this upper bound using Lemma 4.5.38.

Consider a labeling profile $\mathcal{P}$ of all vertices V_u such that $\mathcal{P}(v) = A_r$. By the construction of our input distribution, since $u \in A_r$, at most $O(d_{L-1}) = O(n^{\sigma_{L-1}})$ vertices in V_u belong to A_r. We generate $\Omega(n^{\sigma_L})$ new profiles $\mathcal{P}'$ where $\mathcal{P}'(v) \neq A_r$. For each vertex $w \in V_u$ with $\mathcal{P}(w) = B_r$, we create a new profile $\mathcal{P}'$ by setting $\mathcal{P}'(z) = \mathcal{P}(z)$ for $z \notin \{v, w\}$, $\mathcal{P}'(w) = A_r$, and $\mathcal{P}'(v) = B_r$.

By Lemma 4.5.38, the probability of querying the same shallow subgraphs $T(v)$ and $T(w)$ in the new labeling profile changes only by factors of

$$\left(1 + O(n^{\sigma_L - \delta})\right)^{|T(v)|} \quad \text{and} \quad \left(1 + O(n^{\sigma_L - \delta})\right)^{|T(w)|},$$

respectively. Since v and w are not spoiled vertices, Definition 4.5.33 gives $|T(v)|, |T(w)| \le n^{\delta - 2\sigma_L}$. Therefore, the probability of generating profiles $\mathcal{P}$ and $\mathcal{P}'$ differs by at most

$$\begin{aligned}\left(1 + O(n^{\sigma_L - \delta})\right)^{|T(v)|} \cdot \left(1 + O(n^{\sigma_L - \delta})\right)^{|T(w)|} &\le \left(1 + O(n^{\sigma_L - \delta})\right)^{2n^{\delta - 2\sigma_L}} \\ &\le 1 + o(1).\end{aligned}$$

Now, construct a bipartite graph $H = (P_1, P_2, E_P)$ of labeling profiles, where P_1 consists of all profiles $\mathcal{P}$ with $\mathcal{P}(v) = A_r$, and P_2 contains all profiles $\mathcal{P}'$ with $\mathcal{P}'(v) = B_r$. Add an edge between $\mathcal{P} \in P_1$ and $\mathcal{P}' \in P_2$ if $\mathcal{P}$ can be transformed into $\mathcal{P}'$ through the process described earlier.

For any profile $\mathcal{P} \in P_1$, we have $\deg_H(\mathcal{P}) \ge |V_u|/2 \ge n^{\sigma_L}/4$ since at least $|V_u|/2$ vertices in V_u belong to B_r. In contrast, for any profile $\mathcal{P}' \in P_2$, $\deg_H(\mathcal{P}') \le 2n^{\sigma_{L-1}}$ because, by the input distribution, at most $2d_{L-1} = 2n^{\sigma_{L-1}}$ vertices w in V_u satisfy $\mathcal{P}'(w) = A_r$. Therefore,

$$\begin{aligned}p_e^{\text{inner}} &\le (1 + o(1)) \cdot \frac{|P_1|}{|P_2|} \\ &\le (1 + o(1)) \cdot \frac{2n^{\sigma_{L-1}}}{n^{\sigma_L}/4} \\ &\le (1 + o(1)) \cdot 8n^{\sigma_{L-1} - \sigma_L} \\ &\le 10n^{\sigma_{L-1} - \sigma_L},\end{aligned}$$

which completes the proof. □

Corollary 4.5.40. *There are at most $O(n^{1-2\delta+5\sigma_L})$ edges e such that $p_e^{inner} > 10n^{\sigma_{L-1}-\sigma_L}$.*

Proof. The proof follows by combining Lemma 4.5.37 and Lemma 4.5.39. □

4.5.5 Unbiased Edges Results in Small Connected Components

In this section, we demonstrate that as the recursive construction progresses to deeper levels, the algorithm finds it increasingly difficult to form large connected components using inner-level edges. By Corollary 4.5.40, at the highest level of the construction, the algorithm can identify at most $O(n^{1-2\delta+5\sigma_L})$ edges in the queried subgraph as belonging to the inner level with a probability greater than $10n^{\sigma_{L-1}-\sigma_L}$. For the sake of analysis, we assume the algorithm can perfectly distinguish these edges.

However, for all remaining queried edges, the probability of belonging to the inner level is significantly lower due to the degree choices formalized in Lemma 4.5.39. Specifically, each additional queried edge has at most $O(n^{\sigma_{L-1}-\sigma_L})$ probability of being an inner-level edge.

Our objective is to formalize a similar result to Corollary 4.5.40 for each level in the hierarchy in the next two sections. Intuitively, Lemma 4.5.43 shows that as the construction descends through the levels, the number of edges the algorithm can confidently identify as inner-level edges diminishes. To proceed, we first extend Definition 4.5.24 to all levels of the hierarchy.

Definition 4.5.41 ($p_e^{\ell-inner}$ and Distinguishability of an Edge)**.** *Let e be a real edge or pseudo edge queried by the algorithm, and let $p_e^{\ell-inner}$ denote the probability that if e is a level $\ell-1$ edge that belongs to subgraph between A_r^1 and A_r^2, conditioned on all queries made by the algorithm so far and assuming either input*

distribution. We say edge e is distinguishable if $p_e^{\ell-inner} > 10n^{\sigma_{\ell-1}-\sigma_\ell}$. We use E_ℓ^{inner} to denote the set of edges for which $p_e^{\ell\text{-}inner} > 10n^{\sigma_{\ell-1}-\sigma_\ell}$.

We define function $g(\ell)$ for $0 \le \ell \le L$ as follows:

$$g(\ell) = (L-\ell+2)\cdot\delta - 5\left(\sum_{i=\ell}^{L}\sigma_i/\sigma_{i+1}\right) - 5\left(\sum_{i=\ell}^{L-1}\sigma_i\right),$$

where $\sigma_0 = 0$ and $\sigma_{L+1} = 1$ for the purpose of defining this function.

Observation 4.5.42. *The following statements are true regarding function g:*

(i) $g(\ell-1) = g(\ell) + \delta - 5\sigma_{\ell-1}/\sigma_\ell - 5\sigma_{\ell-1}$ *for* $\ell \in (1, L]$,

(ii) $1 - g(\ell-1) - 3\sigma_{\ell-1} = 1 - g(\ell) - \delta + 5\sigma_{\ell-1}/\sigma_\ell + 2\sigma_{\ell-1}$ *for* $\ell \in (1, L]$,

(iii) $g(1) > 2$,

(iv) $1 - g(\ell) \neq 0$ *for all* $\ell \in [L]$.

Proof. We prove each statement separately.

Proof of *(i)*: From the definition of g, we can express $g(\ell-1)$ as:

$$\begin{aligned} g(\ell-1) &= (L-\ell+3)\cdot\delta - 5\left(\sum_{i=\ell-1}^{L}\frac{\sigma_i}{\sigma_{i+1}}\right) - 5\left(\sum_{i=\ell-1}^{L-1}\sigma_i\right) \\ &= \left[(L-\ell+2)\cdot\delta - 5\left(\sum_{i=\ell}^{L}\frac{\sigma_i}{\sigma_{i+1}}\right) - 5\left(\sum_{i=\ell}^{L-1}\sigma_i\right)\right] + \delta - 5\frac{\sigma_{\ell-1}}{\sigma_\ell} - 5\sigma_{\ell-1} \\ &= g(\ell) + \delta - 5\frac{\sigma_{\ell-1}}{\sigma_\ell} - 5\sigma_{\ell-1}. \end{aligned}$$

Proof of *(ii)*: Using the expression for $g(\ell-1)$ derived in part (i), we get:

$$1 - g(\ell-1) - 3\sigma_{\ell-1} = 1 - g(\ell) - \delta + 5\frac{\sigma_{\ell-1}}{\sigma_\ell} + 2\sigma_{\ell-1}$$

Proof of *(iii)*: Evaluating $g(1)$ with the given definition and using the values of parameters in Table 4.2, we obtain:

$$\begin{aligned} g(1) &= (L+1)\cdot\delta - 5\left(\sum_{i=1}^{L}\frac{\sigma_i}{\sigma_{i+1}}\right) - 5\left(\sum_{i=1}^{L-1}\sigma_i\right) \\ &= (L+1)\cdot\delta - \frac{\delta L}{2} - 5\left(\sum_{i=1}^{L-1}\left(\frac{\delta}{10}\right)^{L+1-i}\right) \\ &> (L+1)\cdot\delta - \frac{\delta L}{2} - \delta \\ &= 2. \end{aligned}$$

Proof of *(iv)*: If there exists some ℓ such that $g(\ell) = 0$, the parameters in Table 4.2 can be slightly perturbed to satisfy all conditions while ensuring $g(\ell) \neq 0$. □

When we consider the level ℓ graph and we refer to the inner level, we specifically mean the union of subgraph of real edges between A_r^1 and A_r^2 at level $\ell - 1$, and pseudo edges of level $\ell - 1$ between A_r^1 and A_r^2. In this section, we denote the edges of the inner level as *inner edges*, while all other edges are referred to as *outer edges*. We prove that the algorithm cannot grow a large component of inner edges. We prove the following two lemmas using induction on ℓ. For the base case of $\ell = L$ in Lemma 4.5.43, we already proved the claim in the previous section (Corollary 4.5.40). To prove Lemma 4.5.44 for a fixed ℓ, we first apply the bound from Lemma 4.5.43 at level ℓ. We then use this result to establish Lemma 4.5.43 for $\ell - 1$. In this section, our primary focus is on proving Lemma 4.5.44 using Lemma 4.5.43. The remainder of this subsection is dedicated to detailing the steps involved in proving Lemma 4.5.44.

Lemma 4.5.43. *The following statements hold with high probability:*

(i) If $1 - g(\ell) > 0$, then the number of edges e such that $p_e^{\ell-inner} > 10n^{\sigma_{\ell-1}-\sigma_\ell}$ is at most $O(n^{1-g(\ell)})$ with high probability.

(ii) If $1-g(\ell) < 0$, then there are no edges e such that $p_e^{\ell-inner} > 10n^{\sigma_{\ell-1}-\sigma_\ell}$ with probability $1-O(n^{1-g(\ell)})$.

(iii) If $1 - g(\ell) < 0$, the number of edges e satisfying $p_e^{\ell-inner} > 10n^{\sigma_{\ell-1}-\sigma_\ell}$ is at most $\widetilde{O}(1)$ with high probability.

Lemma 4.5.44. *Let $C_1, C_2, \ldots, C_c$ denote the underlying undirected connected components of inner edges, where each component contains at least one edge from E_ℓ^{inner}. Then, the following statements hold:*

(i) If $1 - g(\ell) > 0$, then with high probability,

$$\sum_{i=1}^{c} |C_i| \leq O(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell}).$$

(ii) If $1 - g(\ell) < 0$, then we have $c = 0$ with probability $1 - O(n^{1-g(\ell)})$.

(iii) If $1 - g(\ell) < 0$, then with high probability,

$$\sum_{i=1}^{c} |C_i| \leq \widetilde{O}(n^{5\sigma_{\ell-1}/\sigma_\ell})$$

Claim 4.5.45. *There are at most $O(n^{1-\delta+\sigma_{\ell-1}})$ inner edges in the queried subgraph with high probability.*

Proof. Since there are ρn ground edges between every pair of vertices and the algorithm makes $O(n^{2-\delta})$ pair queries, the total number of ground edges queried by the algorithm is $O(\rho n^{3-\delta})$. Each queried ground edges is classified as an inner edge with an independent probability of at most $\rho_{\ell-1}/(\rho n)$, as it must be marked as a pseudo edge of level $\ell - 1$, which occurs with probability $\rho_{\ell-1}/(\rho n)$ independently for all queried ground edges. Therefore, the expected number of such edges found by the algorithm is $O(\rho_{\ell-1} n^{2-\delta})$.

Let X_i be an indicator random variable for the event that the i-th queried ground edge is a pseudo or real edge, and let $X = \sum X_i$ denote the total number of such edges found by the algorithm. We have

$E[X] \leq O(\rho_{\ell-1} n^{2-\delta})$. Moreover, the random variables X_i are independent. Applying the Chernoff bound, we get:

$$\Pr\left[|X - E[X]| \geq 2\sqrt{E[X]\log n}\right] \leq 2\exp\left(-\frac{(2\sqrt{E[X]\log n})^2}{3E[X]}\right) < \frac{1}{n}.$$

Thus, with probability at least $1 - 1/n$, the total number of pseudo or real edges discovered by the algorithm is $O(\rho_{\ell-1} n^{2-\delta})$. Finally, substituting $\rho_{\ell-1} = n^{\sigma_{\ell-1}-1}$ completes the proof. □

Claim 4.5.46. *Consider all queried inner edges, excluding those in E_ℓ^{inner}. The length of the longest directed path consisting of inner edges is less than $5/\sigma_\ell$.*

Proof. Let u be an arbitrary vertex. We first claim that the probability of a directed path of length i starting from u and ending at a vertex v is at most $n^{i(\sigma_{\ell-1}-\sigma_\ell)/2}$. We prove this claim by induction on i.

For the base case $i = 1$, if no edge exists between u and v, the probability is trivially 0. If there is an edge, Lemma 4.5.43 implies that this edge is an inner edge with probability at most $10n^{\sigma_{\ell-1}-\sigma_\ell} < n^{(\sigma_{\ell-1}-\sigma_\ell)/2}$.

Assume the claim holds for all lengths less than i. By Claim 4.5.27, the in-degree of v in the entire queried subgraph (including all edges) is at most $3\sqrt{\log n}$. Let $\{v_1, v_2, \ldots, v_k\}$ denote the vertices with directed edges to v. If there is a directed inner path of length i to v, then there must exist a path of length $i-1$ to some v_j and an inner edge from v_j to v. Let I_w^i represent the event that there exists a directed inner path of length i to vertex w. Using a union bound:

$$\begin{aligned}
\Pr[I_v^i] &\leq \sum_{j=1}^{k} \Pr[I_{v_j}^{i-1}] \cdot \Pr[(v_j, v) \text{ is inner}] \\
&\leq \sum_{j=1}^{k} n^{(i-1)(\sigma_{\ell-1}-\sigma_\ell)/2} \cdot 10n^{\sigma_{\ell-1}-\sigma_\ell} && \text{(By induction hypothesis and Lemma 4.5.43)} \\
&\leq 3\sqrt{\log n} \cdot n^{(i-1)(\sigma_{\ell-1}-\sigma_\ell)/2} \cdot 10n^{\sigma_{\ell-1}-\sigma_\ell} && (k \leq 3\sqrt{\log n} \text{ by Claim 4.5.27}) \\
&\leq 30\sqrt{\log n} \cdot n^{(i+1)(\sigma_{\ell-1}-\sigma_\ell)/2} \\
&\leq n^{i(\sigma_{\ell-1}-\sigma_\ell)/2} && (n \text{ sufficiently large enough}),
\end{aligned}$$

which completes the induction.

Next, we show that with high probability, no directed inner path of length $5/\sigma_\ell$ exists in the graph. The probability of such a path between any two vertices u and v is bounded by $n^{5(\sigma_{\ell-1}-\sigma_\ell)/(2\sigma_\ell)}$. Applying a union bound over all vertex pairs:

$$\begin{aligned}
\Pr\left[\exists \text{ directed inner path of length } 5/\sigma_\ell\right] &\leq n^2 \cdot n^{5(\sigma_{\ell-1}-\sigma_\ell)/(2\sigma_\ell)} \\
&\leq n^{-1/4}. && \left(\text{Since } \sigma_{\ell-1} < \frac{\sigma_\ell}{10}\right).
\end{aligned}$$

Thus, with high probability, the longest directed inner path has a length of at most $5/\sigma_\ell - 1$. □

Claim 4.5.47. *Consider all queried inner edges excluding those in E_ℓ^{inner}. Each vertex has at most $n^{5\sigma_{\ell-1}/\sigma_\ell}$ descendants reachable via directed inner edges with high probability. Furthermore, for each vertex, the total number of inner edges to its descendants is at most $n^{5\sigma_{\ell-1}/\sigma_\ell}$.*

Proof. Suppose that we condition on the absence of inner paths of length $5/\sigma_\ell$ by Claim 4.5.46. Also, each vertex has at most $n^{\sigma_{\ell-1}}$ inner edges in total, even those not queried. Consequently, the number of vertices reachable from u within a distance of $5/\sigma_\ell$ is bounded by $n^{5\sigma_{\ell-1}/\sigma_\ell}$.

□

Definition 4.5.48 (Strongly Connected Component)**.** *Let G be a directed graph. A subset of vertices C is called a strongly connected component of G if it is a maximal set of vertices such that, for every pair of vertices $u, v \in C$, there exists a directed path from u to v and a directed path from v to u.*

Consider the strongly connected component decomposition of the directed inner edges queried by the algorithm excluding edges in E_ℓ^{inner}. From each component with zero in-degrees, we select a representative vertex (an arbitrary vertex in the component). Denote the set of these vertices by R.

Lemma 4.5.49. *Consider all queried inner edges excluding those in E_ℓ^{inner}. Let $v \in R$. The probability that there exists a vertex $u \in R \setminus \{v\}$ such that the descendants of u intersect with those of v is at most $O(n^{5\sigma_{\ell-1}/\sigma_\ell-\delta})$.*

Proof. By Claim 4.5.47, vertex v has at most $n^{5\sigma_{\ell-1}/\sigma_\ell}$ inner descendants. This provides an upper bound on the number of descendants of v that are reachable via directed inner edges. By Claim 4.5.45, there are at most $O(n^{1-\delta+\sigma_{\ell-1}})$ vertices with inner edges. At any point during the execution of the algorithm, there are at most

$$X = O\left(n^{1-\delta+\sigma_{\ell-1}+5\sigma_{\ell-1}/\sigma_\ell}\right)$$

pairs consisting of a descendant of v and a vertex that has at least one inner edge. Since each pair has ρn ground edges between them, the total number of ground edges between these pairs is at most $\rho n X$.

For any ground edge between a pair to become a level $\ell - 1$ real edge, it must be marked as a pseudo edge of level $\ell - 1$. The probability that a given ground edge between a pair is marked as a pseudo edge of level $\ell - 1$ is

$$\frac{1}{n} \cdot \left(\frac{\rho_{\ell-1}}{\rho}\right),$$

independently at random for each of the ground edges. To compute the probability that there is an intersection between the descendants of v and the descendants of any other vertex u, we multiply the number of ground edges by the probability that each of them is a pseudo edge of level $\ell-1$. Thus, the total probability of having an intersection is bounded by

$$(\rho n X) \cdot \frac{1}{n} \cdot \left(\frac{\rho_{\ell-1}}{\rho}\right) = \rho_{\ell-1} X = O(n^{5\sigma_{\ell-1}/\sigma_\ell-\delta})$$

□

Corollary 4.5.50. *Consider all queried inner edges excluding those in E_ℓ^{inner}. Let $k < 50\sigma_\ell/\sigma_{\ell-1}$ and let $v_1, \ldots, v_k$ be k arbitrary vertices in R. The probability that there exists a vertex $u \in R \setminus \{v_1, \ldots, v_k\}$ such that the descendants of u intersect with the descendants of any vertex in $\{v_1, \ldots, v_k\}$ is at most $O(n^{5\sigma_{\ell-1}/\sigma_\ell - \delta})$.*

Proof. The proof follows the same reasoning as that of Lemma 4.5.49, noting that k is a constant. □

Claim 4.5.51. *Consider all queried inner edges excluding those in E_ℓ^{inner}. Let C be an arbitrary connected component formed by these edges. It holds that $|C| \leq O(n^{5\sigma_{\ell-1}/\sigma_\ell})$ with high probability. Furthermore, each connected component contains at most $O(|C|)$ inner edges.*

Proof. Define a graph H_R with vertex set R, where an edge (u, v) exists in H_R if the inner descendants of u and v intersect. We will show that, with high probability, the largest connected component in H_R has a size of at most $10/\delta$. By Claim 4.5.47, each vertex has at most $n^{5\sigma_{\ell-1}/\sigma_\ell}$ inner descendants (and inner edges to its descendants), so proving this claim suffices to complete the proof.

Consider the following exploration process starting from any vertex $v \in R$. Initialize a set $S = \{v\}$. At each step, examine the edges between vertices in S and those in $R \setminus S$. If there exists an edge (w, z) where $w \in S$ and $z \in R \setminus S$, add z to S and continue. The process halts when either no such edge exists or when $|S| > 10/\delta$.

Let X_i denote the event that an edge between S and $R \setminus S$ is revealed at step i. By Lemma 4.5.49, the probability of this event is bounded as:

$$\Pr[X_i \mid X_1, \ldots, X_{i-1}] \leq O(n^{5\sigma_{\ell-1}/\sigma_\ell - \delta + \sigma_{\ell-1}}).$$

Let Y_v represent the event that the process reaches $|S| > 10/\delta$. The probability of this happening is:

$$\begin{aligned}
\Pr[Y_v] &= \prod_{i=1}^{10/\delta} \Pr[X_i \mid X_1, X_2, \ldots, X_{i-1}] \\
&\leq O\left((n^{5\sigma_{\ell-1}/\sigma_\ell - \delta + \sigma_{\ell-1}})^{10/\delta}\right) \\
&= O\left(\frac{1}{n^5}\right).
\end{aligned}$$

Finally, applying a union bound over all vertices $v \in R$, the probability that any connected component in H_R exceeds size $10/\delta$ is at most $O(n^{-4})$. Therefore, with high probability, the largest connected component in H_R has size at most $10/\delta$. □

Corollary 4.5.52. *Consider all queried inner edges excluding those in E_ℓ^{inner}. Let C be any connected component formed by the intersection of descendants of k vertices in R. Then, with high probability, $k \leq 10/\delta$.*

Proof. The result follows directly from the proof of Claim 4.5.51. □

We are ready to prove Lemma 4.5.44.

Proof of Lemma 4.5.44. Assume an adversary selects the edges in E_ℓ^{inner} to connect the components. Let $\widehat{\mathcal{C}} = \{\widehat{C_1}, \widehat{C_2}, \ldots, \widehat{C_{c'}}\}$ denote the connected components prior to adding the edges in E_ℓ^{inner}. By Claim 4.5.51, with high probability, each component satisfies $|\widehat{C_i}| = O(n^{5\sigma_{\ell-1}/\sigma_\ell})$ for all $i \in [c']$.

Now, let $C_1, C_2, \ldots, C_c$ represent the connected components formed after adding the edges in E_ℓ^{inner} and discarding components that do not contain any of these edges. Each edge in E_ℓ^{inner} can merge at most two components from $\widehat{\mathcal{C}}$, so the number of components in $\widehat{\mathcal{C}}$ connected by at least one edge from E_ℓ^{inner} is bounded by $O(|E_\ell^{inner}|)$. Now we prove each statement of the lemma separately:

(i): If $1-g(\ell) > 0$, statement (i) of Lemma 4.5.43 implies that $|E_\ell^{inner}| \leq O(n^{1-g(\ell)})$ with high probability. Since each initial component has size $|\widehat{C_i}| = O(n^{5\sigma_{\ell-1}/\sigma_\ell})$, we conclude that:

$$\sum_{i=1}^{c} |C_i| \leq O(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell}),$$

which completes the proof of the first statement.

(ii): If $1 - g(\ell) < 0$, then by statement (ii) of Lemma 4.5.43, with probability $1 - O(n^{1-g(\ell)})$, the size of E_ℓ^{inner} is zero which completes the proof of the second statement.

(iii): If $1-g(\ell) < 0$, $|E_\ell^{inner}| = \widetilde{O}(1)$ with high probability according to the statement (iii) of Lemma 4.5.43. Combining this with the fact that $|\widehat{C_i}| = O(n^{5\sigma_{\ell-1}/\sigma_\ell})$, the last statement follows immediately.

□

Corollary 4.5.53. *Let $C_1, C_2, \ldots, C_k$ be the underlying undirected connected components of inner edges, where each component contains at least one edge from E_ℓ^{inner}. Let $E(C_i)$ denote the edge set of component C_i. Then, the following statements hold:*

(i) If $1 - g(\ell) > 0$, then with high probability,

$$\sum_{i=1}^{k} |E(C_i)| \leq \widetilde{O}\left(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell}\right).$$

(ii) If $1 - g(\ell) < 0$, then $k = 0$ with probability $1 - O(n^{1-g(\ell)})$.

(iii) If $1 - g(\ell) < 0$, then with high probability,

$$\sum_{i=1}^{k} |E(C_i)| \leq \widetilde{O}\left(n^{5\sigma_{\ell-1}/\sigma_\ell}\right).$$

Proof. Note that $|E(C_i)| \leq O(\log n) \cdot |C_i|$. This is because each vertex has an incoming degree of at most $3\sqrt{\log n}$ in the entire queried subgraph by Claim 4.5.27. By merging the results from Lemma 4.5.44 with the fact that $|E(C_i)| \leq O(\log n) \cdot |C_i|$, we derive each statement accordingly. □

4.5.6 Smaller Connected Components Lead to Fewer Unbiased Inner Edges

In this section, we leverage Lemma 4.5.44 to demonstrate that shrinking the size of connected components makes it increasingly difficult for the algorithm to detect inner edges. For clarity and consistency, we extend the notions of spoiler vertices, spoiled vertices, spoiled edges, and shallow subgraphs from the warm-up section, adapting the terminology accordingly for level ℓ.

Definition 4.5.54 (ℓ-Shallow Subgraph)**.** *Let the outer and inner edges be defined with respect to levels ℓ and $\ell-1$ of the construction hierarchy. For a vertex v, the ℓ-shallow subgraph of v is the set of vertices that are reachable from v by directed paths of length at most $10\log n$ using only inner edges in the queried subgraph. We denote this subgraph as $T^{\ell}(v)$.*

Lemma 4.5.55. *Each vertex belongs to at most $\widetilde{O}(1)$ ℓ-shallow subgraphs with high probability.*

Proof. The proof follows a similar structure to that of Lemma 4.5.29. Let v be an arbitrary vertex in the graph. Let V_i be the set of vertices at a distance i from v for $i \in [10\log n]$ using inner edges in the reverse direction. We will show that, with high probability, $|V_i| \leq 3i\sqrt{\log n}$ using induction.

For the base case of the induction, $i = 1$, the claim holds by Claim 4.5.27. Suppose the claim holds for all $i' < i$. Suppose that each vertex in V_{i-1} has at least one incoming edge. Note that this only increases the size of V_i. Let $u \in V_{i-1}$. Also, let $\hat{V}$ be the set of vertices that the algorithm has found at least one edge in the queried subgraph. If u has more than one incoming inner edge, it should be between a vertex that already has an edge because of the way we defined the direction of edges in Definition 4.5.26. Therefore, there are $|\hat{V}|$ possible pairs between $\hat{V}$ and u, each containing ρn ground edges, and each being marked as a pseudo edge of level $\ell-1$ with probability at most $p_{\ell-1}/(n\rho)$. Thus, the expected number of inner edges between $\hat{V}$ and u is $|\hat{V}| \cdot \rho_{\ell-1} = O(n^{\sigma_L+\sigma_{\ell-1}-\delta})$, since $|\hat{V}| = O(n^{1-\delta+\sigma_L})$ by Lemma 4.5.25. Additionally, $|V_{i-1}| \leq 3(i-1)\sqrt{\log n}$ by induction hypothesis. Hence, the expected number of inner edges between $\hat{V}$ and V_{i-1} is $\widetilde{O}(n^{\sigma_L+\sigma_{\ell-1}-\delta})$.

Let X_i be the indicator random variable for the event that the i-th ground edge between $\hat{V}$ and V_{i-1} is a pseudo of level $\ell-1$, and let $X = \sum X_i$ denote the total number of such edges. We have $E[X] = \widetilde{O}(n^{\sigma_L+\sigma_{\ell-1}-\delta}) < 1$ for large enough n. Furthermore, the random variables X_i are independent. Applying the Chernoff bound, we obtain:

$$\Pr\left[|X - E[X]| \geq 3\sqrt{E[X]\log n}\right] \leq 2\exp\left(-\frac{(3\sqrt{E[X]\log n})^2}{3E[X]}\right) < \frac{1}{n^2}.$$

Thus, with probability at least $1-1/n^2$, the total number of incoming inner edges to v is $3\sqrt{\log n}$. Moreover, we assume that each vertex in V_{i-1} contains at least one incoming inner edge. Hence, we have $|V_i| \leq |V_{i-1}| + 3\sqrt{\log n} =\leq 3i\sqrt{\log n}$ which completes the induction step. Therefore, we have $|V_i| \leq 3i\sqrt{\log n}$ for all $i \leq 10\log n$. As a result, the total number of ℓ-shallow subgraphs that contain v is bounded by

$$1 + \sum_{i=1}^{10\log n} |V_i| \leq 1 + \sum_{i=1}^{10\log n} 3i\sqrt{\log n} \leq (10\log n)\cdot(30\log n\sqrt{\log n}),$$

which completes the proof. □

Corollary 4.5.56. *Each inner edge belongs to at most $\widetilde{O}(1)$ ℓ-shallow subgraphs with high probability.*

Proof. For directed inner edge (u, v), since u is in at most $\widetilde{O}(1)$ shallow subgraphs by Corollary 4.5.56, then (u, v) is in at most $\widetilde{O}(1)$ ℓ-shallow subgraphs. □

Definition 4.5.57 (ℓ-Spoiler Vertex)**.** *For $1 < \ell \leq L$, let $\widehat{E}$ denote the set of inner edges that belong to a connected component containing at least one edge from E_{ℓ}^{inner}. Let u be a vertex in a connected component*

that contains at least one edge from E_ℓ^{inner}. We say a vertex u is an ℓ-spoiler *if an edge (u, v) is discovered by the algorithm at a time when both u and v already have non-zero degree.*

Claim 4.5.58. *Suppose that $1 - g(\ell-1) - 2\sigma_{\ell-1} \geq 0$. Then, there are at most $O(n^{1-g(\ell-1)-2\sigma_{\ell-1}})$ ℓ-spoiler vertices with high probability.*

Proof. Let $C_1, C_2, \ldots, C_k$ be the underlying undirected connected components of inner edges, where each component contains at least one edge from E_ℓ^{inner}. Denote by $\widehat{E}$ the set of inner edges in these connected components. By statement (i) of Corollary 4.5.53, we have

$$|\widehat{E}| \leq O\left(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell}\right).$$

Now, consider the scenario where the algorithm queries a pair of vertices and discovers an edge in $\widehat{E}$. There are at most $O\left(n^{1-\delta+\sigma_{\ell-1}}\right)$ vertex pairs where one vertex is an endpoint of the queried edge, and the other already has an incident inner edge in the queried subgraph. Note that each query between two vertices that already have a non-zero degree in the queried subgraph, and one of the endpoint is in component where an edge of E_ℓ^{inner}, results in either a pseudo or real edge of level $\ell - 1$, creating two ℓ-spoiler vertices by Definition 4.5.57. Therefore, the total number of such vertex pairs is bounded by

$$O\left(|\widehat{E}| \cdot n^{1-\delta+\sigma_{\ell-1}}\right).$$

If edges are revealed between all these pairs, the total number of ground edges would be at most

$$O\left(|\widehat{E}| \cdot \rho n^{2-\delta+\sigma_{\ell-1}}\right),$$

where each pair forms a pseudo or real edge of level $\ell - 1$ with probability at most $\rho_{\ell-1}/(\rho n)$. Hence, the expected number of edges discovered by the algorithm between such pairs is bounded by $O\left(|\widehat{E}| \cdot \rho_{\ell-1} n^{1-\delta+\sigma_{\ell-1}}\right)$. Applying the Chernoff bound, we can show that with high probability, the number of ℓ-spoiler vertices is at most $O\left(|\widehat{E}| \cdot \rho_{\ell-1} n^{1-\delta+\sigma_{\ell-1}}\right)$. Combining this with the fact that $\rho_{\ell-1} = O\left(n^{1-\sigma_{\ell-1}}\right)$ and the bound on $\widehat{E}$, the total number of ℓ-spoiler vertices is bounded by

$$\widetilde{O}\left(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell-\delta+2\sigma_{\ell-1}}\right),$$

with high probability. Finally, by statement (ii) of Observation 4.5.42, we have,

$$1 - g(\ell) - \delta + 5\sigma_{\ell-1}/\sigma_\ell + 2\sigma_{\ell-1} = 1 - g(\ell-1) - 3\sigma_{\ell-1},$$

which implies that the total number of ℓ-spoiler vertices is at most $O(n^{1-g(\ell-1)-2\sigma_{\ell-1}})$. □

Claim 4.5.59. *Suppose that $1 - g(\ell-1) - 2\sigma_{\ell-1} < 0$. Then, the probability of having an ℓ-spoiler vertex is at most $O(n^{1-g(\ell-1)-2\sigma_{\ell-1}})$. Also, there are at most $\widetilde{O}(1)$ ℓ-spoiler vertices with high probability.*

Proof. Let $C_1, C_2, \ldots, C_k$ be the underlying undirected connected components of inner edges, where each component contains at least one edge from E_ℓ^{inner}. Denote by $\widehat{E}$ the set of inner edges in these connected components. We prove the claim by considering the following two cases (note that $1 - g(\ell) \neq 0$ by statement (iv) Observation 4.5.42):

- $1 - g(\ell) > 0$**:** In this case, by statement (i) of Corollary 4.5.53, we have

$$|\widehat{E}| \leq \widetilde{O}\left(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell}\right),$$

with high probability. With the exact same proof as the proof of Claim 4.5.58, the probability of having a ℓ-spoiler vertices is bounded by

$$\widetilde{O}\left(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell-\delta+2\sigma_{\ell-1}}\right).$$

Finally, by statement (ii) of Observation 4.5.42, we have,

$$1 - g(\ell) - \delta + 5\sigma_{\ell-1}/\sigma_\ell + 2\sigma_{\ell-1} = 1 - g(\ell-1) - 3\sigma_{\ell-1},$$

which implies that the probability of having a ℓ-spoiler vertices is at most $O(n^{1-g(\ell-1)-2\sigma_{\ell-1}})$.

- $1 - g(\ell) < 0$**:** In this case, by statement (ii) of Corollary 4.5.53, we have $k = 0$ with probability $1 - O(n^{1-g(\ell)})$ which implies that there is no ℓ-spoiler vertex. Now suppose that we condition on $k > 0$. Then, by statement (iii) of Corollary 4.5.53, we have

$$|\widehat{E}| \leq \widetilde{O}\left(n^{5\sigma_{\ell-1}/\sigma_\ell}\right).$$

With the exact same proof as the proof of Claim 4.5.58, the probability of having an ℓ-spoiler vertex is bounded by

$$\widetilde{O}\left(n^{5\sigma_{\ell-1}/\sigma_\ell+2\sigma_{\ell-1}-\delta}\right).$$

Since the probability of having a component containing an edge from E_ℓ^{inner} is $O\left(n^{1-g(\ell)}\right)$, the overall probability of having an ℓ-spoiler vertex is upper bounded by

$$O\left(n^{1-g(\ell)}\right) \cdot \widetilde{O}\left(n^{5\sigma_{\ell-1}/\sigma_\ell+2\sigma_{\ell-1}-\delta}\right) = \widetilde{O}\left(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell-\delta+2\sigma_{\ell-1}}\right).$$

Thus, combining with statement (ii) of Observation 4.5.42, the probability of having an ℓ-spoiler vertex is at most $O\left(n^{1-g(\ell-1)-2\sigma_{\ell-1}}\right)$.

In both cases, since the expected number of ℓ-spoiler vertices is smaller than 1, then, using Chernoff bound, with high probability there are at most $\widetilde{O}(1)$ ℓ-spoiler vertices which concludes the proof. □

Definition 4.5.60 (ℓ-Spoiled Vertex)**.** *For $1 < \ell \leq L$, let $\widehat{E}$ denote the set of inner edges that belong to a connected component containing at least one edge from E_ℓ^{inner}. Let u be a vertex in a connected component that contains at least one edge from E_ℓ^{inner}. We say a vertex v is called an* ℓ-spoiled *vertex if its shallow subgraph contains any of the following:*

(i) an ℓ-spoiler vertex; or

(ii) at least $n^{\delta-2\sigma_{\ell-1}}$.

Observation 4.5.61. *Let v be a vertex that is not ℓ-spoiled. Then, the ℓ-shallow subgraph of v forms a rooted tree with at most $n^{\delta-2\sigma_{\ell-1}}$ vertices. Additionally, for every edge (u,w) in the ℓ-shallow subgraph of v, when the algorithm queries this edge, vertex w is a singleton.*

Proof. Proof is the same as Observation 4.5.34 by adapting Definition 4.5.57 and Definition 4.5.60. □

Lemma 4.5.62. *Suppose that $1-g(\ell-1)-2\sigma_{\ell-1} \geq 0$. Then, there are at most $O(n^{1-g(\ell-1)-\sigma_{\ell-1}})$ ℓ-spoiled vertices with high probability.*

Proof. Let $C_1, C_2, \ldots, C_k$ denote the underlying undirected connected components of inner edges, where each component contains at least one edge from E_ℓ^{inner}. Additionally, let $\widehat{V}$ represent the set of vertices in these components, and let $E(C_i)$ be the set of edges in component C_i. By statement (i) of Corollary 4.5.53, we have:

$$\sum_{i=1}^{c} |E(C_i)| \leq \widetilde{O}\left(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell}\right).$$

Applying Corollary 4.5.56, it follows that

$$\sum_{u\in\widehat{V}} |T^\ell(u)| \leq \widetilde{O}(1)\cdot\sum_{i=1}^{k} |E(C_i)| \leq \widetilde{O}\left(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell}\right).$$

Hence, the number of vertices for which $|T^\ell(v)| > n^{\delta-\sigma_{\ell-1}}$ is at most

$$\begin{aligned}
\frac{\sum_{u\in\widehat{V}} |T^\ell(u)|}{n^{\delta-2\sigma_{\ell-1}}} &\leq \frac{\widetilde{O}\left(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell}\right)}{n^{\delta-2\sigma_{\ell-1}}} \\
&\leq \widetilde{O}\left(n^{1-g(\ell)-\delta+5\sigma_{\ell-1}/\sigma_\ell+2\sigma_{\ell-1}}\right) \\
&\leq O\left(n^{1-g(\ell-1)-2\sigma_{\ell-1}}\right) \qquad \text{(statement (ii) of Observation 4.5.42),}
\end{aligned}$$

which implies that there are at most $O(n^{1-g(\ell-1)-2\sigma_{\ell-1}})$ vertices that satisfy condition (ii) of Definition 4.5.60.

On the other hand, by Claim 4.5.58, there are at most $O(n^{1-g(\ell-1)-2\sigma_{\ell-1}})$ ℓ-spoiler vertices. By Lemma 4.5.55, each vertex is in at most $\widetilde{O}(1)$ ℓ-shallow subgraphs. Thus, there are at most $O(n^{1-g(\ell-1)-\sigma_{\ell-1}})$ vertices that satisfy condition (i) of Definition 4.5.60 which completes the proof. □

Lemma 4.5.63. *Suppose that $1-g(\ell-1)-2\sigma_{\ell-1} < 0$. Then, the probability of having a ℓ-spoiled vertex is at most $O(n^{1-g(\ell-1)-2\sigma_{\ell-1}})$. Also, there are at most $\widetilde{O}(1)$ ℓ-spoiled vertices with high probability.*

Proof. By Claim 4.5.59, the probability of having an ℓ-spoiler vertex is at most $O(n^{1-g(\ell-1)-2\sigma_{\ell-1}})$ which implies that the probability of satisfying condition (i) of Definition 4.5.60 is upper bounded by $O(n^{1-g(\ell-1)-2\sigma_{\ell-1}})$. Also, there are at most $\widetilde{O}(1)$ ℓ-spoiler vertices with high probability. By Lemma 4.5.55, each vertex is in at most $\widetilde{O}(1)$ ℓ-shallow subgraphs. Hence, there are at most $\widetilde{O}(1)$ vertices that satisfy condition (i) of Definition 4.5.60 with high probability.

Let $C_1, C_2, \ldots, C_k$ denote the underlying undirected connected components of inner edges, where each component contains at least one edge from E_ℓ^{inner}. Additionally, let $\widehat{V}$ represent the set of vertices in these components, and let $E(C_i)$ be the set of edges in component C_i. We prove the claim by considering the following two cases (note that $1-g(\ell)\neq 0$ by statement (iv) Observation 4.5.42):

- $1 - g(\ell) > 0$: by statement (i) of Corollary 4.5.53, we have

$$\sum_{i=1}^{k} |E(C_i)| \leq \widetilde{O}\left(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell}\right),$$

with high probability. Then, by Corollary 4.5.56, we obtain

$$\sum_{u\in\widehat{V}} |T^\ell(u)| \leq \widetilde{O}(1) \cdot \sum_{i=1}^{k} |E(C_i)| \leq \widetilde{O}\left(n^{1-g(\ell)+5\sigma_{\ell-1}/\sigma_\ell}\right).$$

Further,

$$\begin{aligned} 1 - g(\ell) + 5\sigma_{\ell-1}/\sigma_\ell &= 1 - g(\ell-1) + \delta - 5\sigma_{\ell-1} && \text{(By statement (i) of Observation 4.5.42)} \\ &= (1 - g(\ell-1) - 2\sigma_{\ell-1}) + (\delta - 3\sigma_{\ell-1}) \\ &< \delta - \sigma_{\ell-1} && \text{(Since } 1 - g(\ell-1) - 2\sigma_{\ell-1} < 0\text{)}, \end{aligned}$$

which implies that there is no component with an edge of E_ℓ^{inner} with size $n^{\delta-\sigma_{\ell-1}}$ with high probability. Therefore, no vertex satisfies condition (ii) of Definition 4.5.60 with high probability.

- $1 - g(\ell) < 0$: by statement (iii) of Corollary 4.5.53, we have $\sum_{i=1}^{c} |E(C_i)| \leq \widetilde{O}(n^{5\sigma_{\ell-1}/\sigma_\ell})$ with high probability. Since $\delta - 2\sigma_{\ell-1} > 5\sigma_{\ell-1}/\sigma_\ell$, it follows that, with high probability, no component containing an edge from E_ℓ^{inner} has size $n^{\delta-2\sigma_{\ell-1}}$, which implies that with high probability, no vertex satisfies condition (ii) of Definition 4.5.60.

□

Claim 4.5.64. *Consider the connected components $C'_1, C'_2, \ldots, C'_{k'}$ formed by inner edges that do not contain any edges from E_ℓ^{inner}. With probability $1 - O(n^{-\delta+2\sigma_{\ell-1}+10\sigma_{\ell-1}/\sigma_\ell})$, all such components are acyclic, i.e., they form trees.*

Proof. By Claim 4.5.51, each component C'_i satisfies $|C'_i| \leq O(n^{5\sigma_{\ell-1}/\sigma_\ell})$ with high probability for all i. Also, since the total number of inner edges is at most $O(n^{1-\delta+\sigma_{\ell-1}})$ by Claim 4.5.45, it follows that $k' \leq O(n^{1-\delta+\sigma_{\ell-1}})$. Hence, we obtain the bound

$$\sum_{i=1}^{k'} |C'_i|^2 \leq O(n^{1-\delta+2\sigma_{\ell-1}+10\sigma_{\ell-1}/\sigma_\ell}).$$

Fix a component C'_i. Now, consider the process of adding edges to this component one by one in the order they are queried by the algorithm. To form a cycle, at some point during this process, there must be an inner edge within a component that has not yet been queried. Each newly added edge merges two existing components. If the sizes of these components are x and y, then apart from the inner edge discovered by the algorithm, there are potentially xy other new pairs within the newly created component that could contain an inner edge which could potentially create a cycle. Ultimately, the total number of such pairs in a component is $O(|C'_i|^2)$.

Since there are ρn ground edges between each of these pairs, and each is marked as a pseudo edge of level $\ell-1$ with probability $\rho_{\ell-1}/(\rho n)$, the expected number of psuedo edges of level $\ell-1$ in the component, ignoring

those found by the algorithm, is at most $O(\rho_{\ell-1}|C_i'|^2)$. Summing over all components, this expectation is given by

$$\sum_{i=1}^{k'} O(\rho_{\ell-1}|C_i'|^2) = O(n^{-\delta+2\sigma_{\ell-1}+10\sigma_{\ell-1}/\sigma_\ell}).$$

This completes the proof. □

For the remainder, we assume that every connected component of inner edges without an edge from E_ℓ^{inner} is acyclic. By Claim 4.5.64, the probability of this assumption failing is

$$O(n^{-\delta+2\sigma\ell-1+10\sigma_{\ell-1}/\sigma_\ell}) = o(1).$$

Furthermore, since the hierarchy has a constant number of levels, this holds for all levels with probability $1 - o(1)$.

Definition 4.5.65 (ℓ-Spoiled Edge)**.** *Let (u, v) be a directed inner edge. We say (u, v) is a spoiled edge if $u \in A_r$ in level $\ell - 1$ of hierarchy and at least one of the following condition holds:*

(i) v is an ℓ-spoiled vertex; or

(ii) u has at least $n^{\sigma_{\ell-1}}/3$ ℓ-spoiled neighbors in the queried subgraph.

Lemma 4.5.66. *The following two statements hold regarding the number of ℓ-spoiled edges:*

(i) If $1 - g(\ell - 1) > 0$, then the number of ℓ-spoiled edges is at most $O(n^{1-g(\ell-1)})$ with high probability.

(ii) If $1 - g(\ell - 1) < 0$, then no ℓ-spoiled edge exists with probability $1 - O(n^{1-g(\ell-1)})$.

(iii) If $1 - g(\ell - 1) < 0$, then the number of ℓ-spoiled edge is at most $\widetilde{O}(1)$ with high probability.

Proof. Let $\widehat{E}$ be the set of ℓ-spoiled edges. We prove each statement separately:

(i): First consider the case that $1-g(\ell-1)-2\sigma_{\ell-1} \geq 0$. By Lemma 4.5.62, there are at most $O(n^{1-g(\ell-1)-\sigma_{\ell-1}})$ ℓ-spoiled vertices with high probability. Furthermore, each vertex has an indegree of $\widetilde{O}(1)$ by Claim 4.5.27 with high probability. Thus, there are at most $O(n^{1-g(\ell-1)})$ edges that satisfy condition (i) of Definition 4.5.65.

On the other hand, if vertex u satisfies condition (ii) of Definition 4.5.65, it must have at least $n^{\sigma_{\ell-1}}/4$ outgoing edges (u, w) where w is ℓ-spoiled. But using Claim 4.5.27, each ℓ-spoiled vertex has an indegree of $\widetilde{O}(1)$. Combining with the fact that the total number of ℓ-spoiled vertices is $O(n^{1-g(\ell-1)-\sigma_{\ell-1}})$ implies that the total number of edges satisfy condition (ii) of Definition 4.5.65 is at most $O(n^{1-g(\ell-1)})$.

(ii): If $1 - g(\ell - 1) < 0$ then we have $1 - g(\ell - 1) - 2\sigma_{\ell-1} < 0$. Then, the probability of having a ℓ-spoiled vertex is at most $O(n^{1-g(\ell-1)-2\sigma_{\ell-1}})$ by Lemma 4.5.63. Since $2\sigma_{\ell-1} > 0$ and according to Definition 4.5.65, the probability of having an ℓ-spoiled edge is at most $O(n^{1-g(\ell-1)})$.

(iii): If $1 - g(\ell - 1) < 0$ then we have $1 - g(\ell - 1) - 2\sigma_{\ell-1} < 0$. Then, by Lemma 4.5.63, there are at most $\widetilde{O}(1)$ ℓ-spoiled vertices with high probability. Therefore, with high probability, there are at most $\widetilde{O}(1)$ ℓ-spoiled edges.

□

Lemma 4.5.67. *Let (u, v) be a directed inner edge within a connected component C that does not contain any edge from E_ℓ^{inner}. Suppose $u \in A_r$ and v belongs to $\{A_r, B_r, D_r\}$ at level $\ell - 1$ of the hierarchy. Define $\overline{C}$ as the component containing v after removing the edge (u, v). Let $\mathcal{L}(v)$ and $\mathcal{L}'(v)$ represent an arbitrary label for v from $\{A_r, B_r, D_r\}$ and the entire queried subgraph of inner edges, excluding $\overline{C}$. Then, we have:*

$$\Pr[\overline{C} \mid \mathcal{L}(v)] \leq \left(1 + O(n^{\sigma_{\ell-1}-\delta})\right)^{|\overline{C}|} \cdot \Pr[\overline{C} \mid \mathcal{L}'(v)].$$

Lemma 4.5.68. *Let v be a vertex that is not ℓ-spoiled and belongs to a connected component that contains at least one edge from E_ℓ^{inner}. Suppose $v \in \{A_r, B_r, D_r\}$ at level $\ell - 1$ of the hierarchy. Let $\mathcal{L}(v)$ and $\mathcal{L}'(v)$ represent an arbitrary label for v from $\{A_r, B_r, D_r\}$ and the entire queried subgraph of inner edges, excluding the ℓ-shallow subgraph of v. Then, we have:*

$$\Pr[T^\ell(v) \mid \mathcal{L}(v)] \leq \left(1 + O(n^{\sigma_{\ell-1}-\delta})\right)^{|T^\ell(v)|} \cdot \Pr[T^\ell(v) \mid \mathcal{L}'(v)].$$

The proofs of these two lemmas are postponed to a later section as they are intricate, lengthy, and largely independent of the discussion in this section.

Lemma 4.5.69. *Let e be a directed inner edge that is not ℓ-spoiled. Then, it holds that $p_e^{(\ell-1)-inner} \leq 10n^{\sigma_{\ell-2}-\sigma_{\ell-1}}$.*

Proof. Let $e = (u, v)$ be a directed inner edge from u to v. First, if $u \notin A_r$, we have $p_e^{(\ell-1)-inner} = 0$. Also, if e is a pseudo edge of level $\ell - 1$, then by construction, the probability that we mark e as level $\ell - 2$ is $\rho_{\ell-2}/\rho_{\ell-1} = n^{\sigma_{\ell-2}-\sigma_{\ell-1}}$ independently from other edges.

Hence, suppose that $u \in A_r$ and e is a real edge for the rest of the proof. According to the construction, u has at least $6n^{\sigma_{\ell-1}}/7$ inner edges that are real edges such that the other endpoint has label $A_r \cup B_r$. Also, u has at most $\widetilde{O}(1)$ incoming edge by Claim 4.5.27. Since e is not an ℓ-spoiled edge, u has at most $n^{\sigma_{\ell-1}}/3$ neighbors in the queried subgraph that are ℓ-spoiled. Let V_u be the set of neighbors of u using inner edges that are real edges in the original graph such that their label is $A_r \cup B_r$ and either they are singleton or direct children of u in the queried subgraph. By the above argument, we have $|V_u| \geq n^{\sigma_{\ell-1}}/2$. Also, note that $v \in V_u$. Now we provide an upper bound on the probability that e belongs to level $\ell - 2$, or in other words, $v \in A_r$. We prove this upper bound using Lemma 4.5.67 and Lemma 4.5.68.

Consider a labeling profile $\mathcal{P}$ of all vertices V_u such that $\mathcal{P}(v) = A_r$. By the construction of our input distribution, since $u \in A_r$, at most $O(d_{\ell-2}) = O(n^{\sigma_{\ell-2}})$ vertices in V_u belong to A_r. We generate $\Omega(n^{\sigma_{\ell-1}})$ new profiles $\mathcal{P}'$ where $\mathcal{P}'(v) \neq A_r$. For each vertex $w \in V_u$ with $\mathcal{P}(w) = B_r$, we create a new profile $\mathcal{P}'$ by setting $\mathcal{P}'(z) = \mathcal{P}(z)$ for $z \notin \{v, w\}$, $\mathcal{P}'(w) = A_r$, and $\mathcal{P}'(v) = B_r$.

Since v and w are not ℓ-spoiled vertices, they either belong to a component with no edges in E_ℓ^{inner} or have an ℓ-shallow subgraph that satisfies the conditions in Definition 4.5.60. In both cases, the probability of querying the same shallow subgraph or connected component in the new labeling profile remains the same up to a factor of

$$\left(1 + O(n^{\sigma_{\ell-1}-\delta})\right)^{n^{\delta-2\sigma_{\ell-1}}},$$

by Lemma 4.5.68 and Lemma 4.5.67 since either $|T^\ell(v)| \le n^{\delta-2\sigma_{\ell-1}}$ (resp. $|T^\ell(w)| \le n^{\delta-2\sigma_{\ell-1}}$) or the component that v or w belongs to has size of at most $O(n^{5\sigma_{\ell-1}/\sigma_\ell})$ which is smaller than $O(n^{\delta-2\sigma_{\ell-1}})$ for large enough n. Therefore, the probability of generating profiles $\mathcal{P}$ and $\mathcal{P}'$ differs by at most

$$\begin{aligned}\left(1+O(n^{\sigma_{\ell-1}-\delta})\right)^{n^{\delta-2\sigma_{\ell-1}}} \cdot \left(1+O(n^{\sigma_{\ell-1}-\delta})\right)^{n^{\delta-2\sigma_{\ell-1}}} &\le \left(1+O(n^{\sigma_{\ell-1}-\delta})\right)^{2n^{\delta-2\sigma_L}} \\ &\le 1+o(1).\end{aligned}$$

Now, construct a bipartite graph $H = (P_1, P_2, E_P)$ of labeling profiles, where P_1 consists of all profiles $\mathcal{P}$ with $\mathcal{P}(v) = A_r$, and P_2 contains all profiles $\mathcal{P}'$ with $\mathcal{P}'(v) = B_r$. Add an edge between $\mathcal{P} \in P_1$ and $\mathcal{P}' \in P_2$ if $\mathcal{P}$ can be transformed into $\mathcal{P}'$ through the process described earlier.

For any profile $\mathcal{P} \in P_1$, we have $\deg_H(\mathcal{P}) \ge |V_u|/2 \ge n^{\sigma_{\ell-1}}/4$ since at least $|V_u|/2$ vertices in V_u belong to B_r. In contrast, for any profile $\mathcal{P}' \in P_2$, $\deg_H(\mathcal{P}') \le 2n^{\sigma_{\ell-2}}$ because, by the input distribution, at most $2d_{\ell-2} = 2n^{\sigma_{\ell-2}}$ vertices w in V_u satisfy $\mathcal{P}'(w) = A_r$. Therefore,

$$\begin{aligned}p_e^{\text{inner}} &\le (1+o(1)) \cdot \frac{|P_1|}{|P_2|} \\ &\le (1+o(1)) \cdot \frac{2n^{\sigma_{\ell-2}}}{n^{\sigma_{\ell-1}}/4} \\ &\le (1+o(1)) \cdot 8n^{\sigma_{\ell-2}-\sigma_{\ell-1}} \\ &\le 10n^{\sigma_{\ell-2}-\sigma_{\ell-1}},\end{aligned}$$

which concludes the proof. □

Now we finish the proof of Lemma 4.5.43.

Proof of Lemma 4.5.43. We need to prove the lemma for $\ell-1$ using Lemma 4.5.44. According to Lemma 4.5.69, which relies on Lemma 4.5.44 for ℓ, if an edge is not an ℓ-spoiled edge, then we have $p_e^{(\ell-1)-inner} \le 10n^{\sigma_{\ell-2}-\sigma_{\ell-1}}$. Thus, by applying each statement of Lemma 4.5.66, we obtain the proof for the corresponding item in the lemma. □

4.5.7 Identical Distribution for Acyclic Subgraphs

In this section, we prove Lemma 4.5.38, Lemma 4.5.68, and Lemma 4.5.67. The main difference is that, in our setting, we need to incorporate pseudo edges into the coupling argument. Furthermore, instead of a regular graph, we consider an Erdős–Rényi graph, which necessitates slight adjustments to the proof. We will demonstrate that the same result holds under this construction. To do so, we step by step revisit the original proof, making the necessary modifications to account for pseudo edges. For now, suppose we are at a fixed level ℓ in the hierarchy. We begin by introducing the relevant notation and establishing the essential tools required to achieve the main objective of this section. Throughout the remainder of the section, assume that $d = d_\ell/d_{\ell-1} = \Theta(n^{\sigma_\ell-\sigma_{\ell-1}})$. Also, when we say a vertex belongs to layer i, we mean it belongs to $A_i \cup B_i \cup D_i$. Moreover, we consider vertices of S as layer 0.

Definition 4.5.70 (Special Edge)**.** *We define an edge (u, v) as* special *if any of the following conditions hold:*

- (u, v) *belongs to gadget between* S *and* B_1*,*
- (u, v) *belongs to gadgets between* B_i *and* A_{i-1} *for* $i \in (1, r]$*.*
- (u, v) *belongs to a gadget that exists only in* $\mathcal{D}^{\ell}_{\mathsf{YES}}$ *or* $\mathcal{D}^{\ell}_{\mathsf{NO}}$*,*
- (u, v) *belongs to a gadget between* D^j_i *and* $D^{j'}_i$ *for* $i \in [r]$*, where* $j \in \{1, 3\}$ *and* $j' \in \{2, 4\}$*. At the base level, we consider edges between* D^1_i *and* D^2_i *for* $i \in [r]$*.*

Definition 4.5.71 (Special Pseudo Edge of Layer i)**.** *Let* T *be a rooted tree with root* u*, where* $u \in \{A_r, B_r, D_r\}$*. Consider a vertex* v *and suppose there exists a path in* T *leading to* v *such that it contains no mixer vertices (as defined in Definition 4.5.73) and has exactly* k *special crossings. Additionally, assume that* $v \notin B_{r-k}$*. Now, consider the pseudo edges between* v *and* A_{i-1}*. By construction, there are no real edges connecting* v *to vertices in* A_{i-1}*. However, in expectation, there are* $\rho_\ell N_\ell$ *pseudo edges of level* ℓ *between* v *and vertices in* A_{i-1}*. Each pseudo edge between* v *and* A_{i-1} *is classified as a special pseudo edge of layer* i *with probability* $\log n/\rho_\ell$*. Thus, in expectation,* v *has* $\log n$ *special pseudo edges of layer* i *in* A_{i-1}*. Furthermore, if* $v \in B_{r-k}$*, the expected number of pseudo edges between* v *and vertices in layers no lower than* $r - k$ *is* $\log n$ *greater than for other labels in layer no lower than* $r - k$*. Therefore, we randomly select* $\log n$ *edges in expectation to be marked as special crossings.*

Definition 4.5.72 (Special Crossing)**.** *Let* T *be a rooted tree with root* u*. Let* e *be an edge in the tree. We call edge* e *as* special crossing *if any of the following hold:*

- e *is a special edge by Definition 4.5.70,*
- e *is a special pseudo edge of layer* i *for some* $i \in [r]$ *by Definition 4.5.71.*

Definition 4.5.73 (Mixer Vertex)**.** *Let* T *be a rooted tree with root* u*, where* $u \in \{A_r, B_r, D_r\}$*. Consider a vertex* v *in* T*, and suppose there are* k *special crossings along the path from* u *to* v*. We say* v *is a* mixer vertex *if and only if* $k < r - 1$ *and* $v \in \bigcup_{i=1}^{r-k-1}(A_i, B_i, D_i)$*.*

Note that the definition of a mixer vertex relies on special pseudo edges, while the definition of special pseudo edges depends on mixer vertices. However, each of these definitions uses the other only up to a distance of i from the root of the tree to establish the definition for distance $i + 1$.

Claim 4.5.74. *Let* T *be a rooted tree with root* u*, where* $u \in \{A_r, B_r, D_r\}$*. Let* v *be a vertex in the tree where* v *belongs to layer* i*. Any path from* u *to vertex* v *that does not pass through a mixer vertex must contain at least* $r - i$ *special crossings.*

Proof. We claim that any path reaching a vertex v in a layer smaller than or equal i, without passing through a mixer vertex, must contain at least $r - i$ special crossings. We prove this claim by induction. For the base case $r = i$, the statement is trivial. Now, suppose the claim holds for all $j > i$, and we aim to prove it for i. Consider the first edge in the path where the algorithm reaches a vertex in layer i or smaller for the first time—let this vertex be w, reached via edge e. We analyze two possible scenarios:

- e is a real edge: By the construction of the gadgets, either e is a special edge between B_{i+1} and A_i, or w belongs to D_i, in which case w is a mixer vertex by Definition 4.5.73.

- e is a pseudo edge: If e is not a special pseudo edge of layer i, then w must be a mixer vertex according to Definition 4.5.73.

In both cases, when the algorithm reaches a vertex in layer i or lower for the first time, it requires at least one additional special crossing. Applying the induction hypothesis for $i+1$ completes the proof. □

Lemma 4.5.75. *Let T be a rooted tree queried by the algorithm, with its root in $\{A_r, B_r, D_r\}$. With probability at least $1 - \widetilde{O}(|V(T)|/d^{r-1})$, any path from the root to a vertex in T that does not cross a mixer vertex contains at most $r-2$ special crossings.*

Proof. We show that with probability $\widetilde{O}(1/d^{r-1})$, any path found by the algorithm that contains at least $r-1$ special crossings does not pass through a mixer vertex. Let i denote the index of a mixer vertex located in layer i. Suppose there exists an oracle that, whenever the algorithm discovers a path with at least $r-1$ special crossings, performs one of the following actions:

- confirms that the path does not contain any mixer vertices, or
- reveals the mixer vertex with the smallest index along the path.

This oracle is introduced purely for analytical purposes; the algorithm itself does not have access to this additional power. However, we demonstrate that even with this advantage, it remains difficult for the algorithm to find a path with $r-1$ special crossings without encountering a mixer vertex on the path.

Now, consider the first path discovered by the algorithm that contains $r-1$ special crossings. Let us examine the moment during the query process when the algorithm detects $r-2$ special crossings for the first time. Suppose that, up to this point, the path does not contain a mixer vertex with index 1. This implies, by Claim 4.5.74, that the path does not reach any vertex in layer 1 or below. At this step, when the algorithm queries the next edge (ignoring queries that do not result in either a pseudo edge or a real edge), the probability of encountering another special crossing is $\widetilde{O}(1/d)$, based on the construction and the definition of special crossings in Definition 4.5.72. However, the probability of encountering a mixer vertex with index 1 is $\Theta(1)$, according to the same construction. Consequently, the probability that the path reaches another special crossing before encountering a mixer vertex with index 1 is $\widetilde{O}(1/d)$.

For the sake of analysis, we introduce an additional power for the algorithm: if the oracle has already revealed half of the mixer vertices of a given index i that are direct children of a vertex v, then the oracle will either:

- reveals a mixer vertex closer to the root with an index larger than i, if such a mixer vertex exists, or
- construct a path with $r-1$ special crossings that avoids mixer vertices altogether, immediately terminating the process.

From the previous argument, we conclude that an $\widetilde{O}(1/d)$ fraction of paths does not have a mixer vertex with index 1. Furthermore, when the algorithm finds $\Omega(d)$ direct children of a vertex that are mixer vertices with index 1, the oracle reveals the next mixer vertex with a higher index. As a result, the proportion of paths discovered by the algorithm that do not contain a mixer vertex of index 1 remains at most an $\widetilde{O}(1/d)$ fraction of all paths. Further, observe that once a mixer vertex z is revealed by the oracle, any further queries below z become not useful for constructing a path with $r-1$ special crossings while avoiding mixer

vertices. This is because, for any path that crosses z, the mixer vertex with the highest index that the oracle can reveal will always be w itself.

Let us consider the set of paths that do not pass through a mixer vertex of index 1. By a similar argument, an $\widetilde{O}(1/d)$ fraction of these paths also avoids encountering a mixer vertex of index 2. Specifically, the probability that a path reaches the $(r-2)$-th special crossing before encountering a mixer vertex of index 2 is $\widetilde{O}(1/d)$. As a result, the fraction of paths that do not contain a mixer vertex of index 2 or lower is at most $\widetilde{O}(1/d^2)$.

Applying the exact same argument, the probability that a path avoids all mixer vertices with an index up to i is at most $\widetilde{O}(1/d^i)$. Therefore, the fraction of paths that do not contain any mixer vertex is bounded by $\widetilde{O}(1/d^{r-1})$. Since the total number of paths originating from the root is at most $O(|V(T)|)$, it follows that with probability at least $1-\widetilde{O}(|V(T)|/d^{r-1})$, every path with more than $r-2$ special crossings must pass through at least one mixer vertex. □

Lemma 4.5.76 (Modification of Lemma 6.7 of [41] with Pseudo Edges)**.** *Consider the highest level of hierarchy. Let T be a rooted tree that is queried by the algorithm where the root of the tree is in $\{A_r, B_r, D_r\}$. Also, suppose that we condition on having a mixer vertex on all paths that contain at least $r-1$ special crossings. Then, the probability of seeing the same tree is equal for all possible roots in $\{A_r, B_r, D_r\}$ up to $(1+O(n^{\sigma_L-\delta}))^{|T|}$ multiplicative factor.*

Proof. The proof follows the same outline as Lemma 6.7 in [41], with a modification to account for pseudo edges. It is based on a one-to-one coupling argument: for each tree queried by the algorithm, if l_1 is the label of the tree, the algorithm will observe the same tree with almost the same probability if it starts from label l_2.

First, note that according to our definition of special crossings Definition 4.5.72, if we query a pair of vertices (u,v) and another pair (w,z), the probability of each being a special crossing is the same. This is because, at any given moment, each vertex has $2\log n$ expected special crossing neighbors, assuming we condition on the fact that no labels have been revealed yet in the coupling.

Let $L_i = \{A_i, B_i, D_i\}$ for each i. Also, define

$$G_i = G\left[\bigcup_{j=i}^{r} L_j\right].$$

Let u and v be two distinct vertices in G_i. An important property of our input distribution is that if we query a neighbor of u and v via a real edge, and the queried edge is not a special edge, then the probability that the queried neighbor belongs to G_i is the same for both u and v. Similarly, if we query a neighbor of u and v via a pseudo edge, and the queried edge is not a special pseudo edge, then the probability that the queried neighbor belongs to G_i remains the same for both u and v. These properties follow directly from the structure of the input construction, as formalized in Corollary 4.5.21 and Corollary 4.5.22.

For a vertex u in a tree T, if there is no mixer vertex on its path to the root, we define the *progress* of u, denoted as p_u, which represents the number of special crossings along the path from the root to u. Under the assumptions of the lemma statement, we have

$$0 \le p_u < r-1, \quad \text{for all } u \in T.$$

We claim that for a vertex u where there is no mixer vertex on its path to route, we have $u \in V(G_{r-p_u})$. This is because, according to the construction and Definition 4.5.72, the only way to go down one layer is through special crossings or mixer vertices.

Let $\mathcal{L}_1$ be a labeling of vertices in T where the root is labeled with l_1. We construct the labeling $\mathcal{L}_2$ from top to the bottom of the tree such that the label of root is l_2. Consider the edge (u, v) with direction from u to v when considering the rooted tree. So at this point we have a label for u in $\mathcal{L}_2$ and we want to chose the label of v in $\mathcal{L}_2$. We maintain the invariant that progress of each vertex is the same in both labeling $\mathcal{L}_1$ and $\mathcal{L}_2$. We consider five possible cases for edge e for our coupling step:

- Edge e is special crossing: in this case, we know v is not labeled S in $\mathcal{L}_1$ as in order to reach S vertex, at least $r-1$ special crossings are needed or the path contains mixer vertex. Thus, suppose that v is not an S vertex. In labeling $\mathcal{L}_2$, we assume that edge (u, v) is a special crossing and we choose the label of v accordingly based on label of u in $\mathcal{L}_2$. Since each vertex has $2 \log n$ expected number of special crossings ($\log n$ for special edges and $\log n$ for special pseudo edges), no matter what is the label of u in $\mathcal{L}_1$ and $\mathcal{L}_2$, the distribution of their special edges are the same. Also, the invariant remains valid since, in this case, $p_v = p_u + 1$, and the progress of vertex u is the same in both $\mathcal{L}_1$ and $\mathcal{L}_2$.

- Vertex v is a mixer vertex in $\mathcal{L}_1$ and e is a real edge: in this case, label of v must be among $\bigcup_{i=1}^{r-p_u-1} D_i$ as the only possible gadgets that add real edges between vertices of G_{r-p_u} and vertices of layers lower than p_u are those to vertices in $\bigcup_{i=1}^{r-p_u-1} D_i$. Since the distribution of neighbors of vertices in G_{r-p_u} to $\bigcup_{i=1}^{r-p_u-1} D_i$ are the same, we assign the same label in $\mathcal{L}_1$ to $\mathcal{L}_2$ for vertex v. Moreover, for the subtree rooted at v, we assume that all labels remain the same since the label of v is identical at this point in both $\mathcal{L}_1$ and $\mathcal{L}_2$. Furthermore, the invariant remains valid since we have not used any special crossings in labeling $\mathcal{L}_2$.

- Vertex v is a mixer vertex in $\mathcal{L}_1$ and e is a pseudo edge: in this case, we assign the same label in $\mathcal{L}_1$ to $\mathcal{L}_2$ for vertex v. The reason is that the distribution of neighbors of vertices in G_{r-p_u} to any type of label in layer lower than p_u is the same if we ignore special pseudo edges. Moreover, for the subtree rooted at v, we assume that all labels remain the same since the label of v is identical at this point in both $\mathcal{L}_1$ and $\mathcal{L}_2$. Furthermore, the invariant remains valid since we have not used any special crossings in labeling $\mathcal{L}_2$.

- Edge e is a real edge which is not special and belongs to G_{r-p_u}: since the distribution of neighbors of vertices in induced subgraph of G_{r-p_u} using real edges is the same if we ignore special crossings, we randomly choose one of the real edge of u as the possible label for v in $\mathcal{L}_2$. Furthermore, the invariant remains valid since we have not used any special crossings in labeling $\mathcal{L}_2$.

- Edge e is a pseudo edge which is not special and belongs to G_{r-p_u}: since the distribution of neighbors of vertices in induced subgraph of G_{r-p_u} using pseudo edges is the same if we ignore special crossings, we randomly choose one of the pseudo edges of u as the possible label for v in $\mathcal{L}_2$. Furthermore, the invariant remains valid since we have not used any special crossings in labeling $\mathcal{L}_2$.

It is also important to note that the number of non-singleton vertices in level L is at most $O(n^{\sigma_L - \delta})$ since there are at most $O(n^{1+\sigma_L-\delta})$ non-singleton vertices by Lemma 4.5.25. Since the total number of steps

is at most $|T|$, the probability that the new labeling still forms a forest differs by a multiplicative factor of at most $(1 + O(n^{\sigma_L - \delta}))^{|T|}$. □

Proof of Lemma 4.5.38. By Observation 4.5.34, since v is not spoiled, then, the shallow subgraph of v is rooted. First, we bound the failure probability of the event in Lemma 4.5.75. More specifically, we upper bound the probability of having a path with at least $r-1$ special crossings without any mixer vertex. Since v is not a spoiled vertex, we have $|T(v)| \leq n^{\delta - 2\sigma_L}$. Therefore, the probability of having a path with $r-1$ crossing without a mixer vertex is at most

$$\begin{aligned}
\widetilde{O}\left(\frac{|T(v)|}{d^{r-1}}\right) &= \widetilde{O}\left(\frac{n^{\delta-2\sigma_L}}{d^{r-1}}\right) && \text{(Since } |T(v)| \leq n^{\delta-2\sigma_L}\text{)} \\
&= \widetilde{O}\left(\frac{n^{\delta-2\sigma_L}}{n^{(\sigma_L-\sigma_{L-1})\cdot(r-1)}}\right) && \text{(Since } d = \Theta(n^{\sigma_L-\sigma_{L-1}})\text{)} \\
&= \widetilde{O}\left(\frac{n^{\delta-2\sigma_L}}{n^{r\sigma_L/4}}\right) && \text{(Since } \sigma_L - \sigma_{L-1} \geq \sigma_L/2 \text{ and } r-1 \geq r/2\text{)} \\
&= \widetilde{O}\left(n^{\delta-2\sigma_L-10/(4\delta)}\right) && \text{(Since } r\sigma_L \geq 10/\delta\text{)} \\
&= O\left(n^{\delta-2\sigma_L-2}\right) && \text{(Since } \delta \leq 1 \text{ and } n \text{ is sufficiently large enough)} \\
&= O(n^{-1})
\end{aligned}$$

Note that we condition on the labels of all vertices except those in the shallow subgraph of vertex v. However, in the coupling described in Lemma 4.5.76, there is no such conditioning on vertex labels. Since the total number of vertices whose labels we are conditioning on is $O(n^{1-\delta+\sigma_L})$, the probability shift in each step of the coupling in Lemma 4.5.76 is at most $O(n^{1-\delta+\sigma_L}/n) = O(n^{\sigma_L-\delta})$. Also, we condition on the high probability event of Lemma 4.5.75. Moreover, since the number of steps in the coupling is $|T(v)| - 1$, the total probability shift is upper bounded by

$$\left(1 + O(n^{\sigma_L-\delta})\right)^{|T(v)|} \cdot \left(1 + O(n^{\sigma_L-\delta})\right)^{|T(v)|-1} \cdot \left(1 + O(n^{-1})\right) \leq \left(1 + O(n^{\sigma_L-\delta})\right)^{|T(v)|},$$

where the inequality is followed by the fact that $-1 < \sigma_L - \delta$, which completes the proof. □

Lemma 4.5.77 (Modification of Lemma 6.7 of [41] with Pseudo Edges)**.** *Consider the level ℓ of the hierarchy. Let T be a rooted tree that is queried by the algorithm where the root of the tree is in $\{A_r, B_r, D_r\}$. Also, suppose that we condition on having a mixer vertex on all paths that contain at least $r-1$ special crossings. Then, the probability of seeing the same tree is equal for all possible roots in $\{A_r, B_r, D_r\}$ up to $(1 + O(n^{\sigma_{\ell-1}-\delta}))^{|T|}$ multiplicative factor.*

Proof. Proof is the same as Lemma 4.5.76 with proper parameters for level ℓ. □

Proof of Lemma 4.5.68. By Observation 4.5.61, since v is not ℓ-spoiled, then, the ℓ-shallow subgraph of v is rooted. With the exact same argument as proof of Lemma 4.5.38, the failure probability of the event in Lemma 4.5.75 is at most

$$\widetilde{O}\left(\frac{|T(v)|}{d^{r-1}}\right) = \widetilde{O}\left(\frac{n^{\delta-2\sigma_{\ell-1}}}{d^{r-1}}\right) \qquad \text{(Since } |T(v)| \leq n^{\delta-2\sigma_{\ell-1}}\text{)}$$

$$
\begin{aligned}
&= \widetilde{O}\left(\frac{n^{\delta-2\sigma_{\ell-1}}}{n^{(\sigma_\ell-\sigma_{\ell-1})\cdot(r-1)}}\right) && \text{(Since } d=\Theta(n^{\sigma_\ell-\sigma_{\ell-1}})\text{)}\\
&= \widetilde{O}\left(\frac{n^{\delta-2\sigma_\ell}}{n^{r\sigma_\ell/4}}\right) && \text{(Since } \sigma_\ell-\sigma_{\ell-1}\geq\sigma_\ell/2 \text{ and } r-1\geq r/2\text{)}\\
&= \widetilde{O}\left(n^{\delta-2\sigma_\ell-10/(4\delta)}\right) && \text{(Since } r\sigma_\ell\geq 10/\delta\text{)}\\
&= O\left(n^{\delta-2\sigma_\ell-2}\right) && \text{(Since } \delta\leq 1 \text{ and } n \text{ is sufficiently large enough)}\\
&= O(n^{-1})
\end{aligned}
$$

Note that we condition on the labels of all vertices except those in the ℓ-shallow subgraph of vertex v. However, in the coupling described in Lemma 4.5.77, there is no such conditioning on vertex labels. Since the total number of vertices whose labels we are conditioning on is $O(n^{1-\delta+\sigma_{\ell-1}})$, the probability shift in each step of the coupling in Lemma 4.5.76 is at most $O(n^{1-\delta+\sigma_{\ell-1}}/n) = O(n^{\sigma_{\ell-1}-\delta})$. Also, we condition on the high probability event of Lemma 4.5.75. Moreover, since the number of steps in the coupling is $|T(v)|-1$, the total probability shift is upper bounded by

$$\left(1+O(n^{\sigma_{\ell-1}-\delta})\right)^{|T(v)|}\cdot\left(1+O(n^{\sigma_{\ell-1}-\delta})\right)^{|T(v)|-1}\cdot\left(1+O(n^{-1})\right)\leq\left(1+O(n^{\sigma_{\ell-1}-\delta})\right)^{|T(v)|},$$

where the inequality is followed by the fact that $-1<\sigma_{\ell-1}-\delta$, which completes the proof. □

Lemma 4.5.78. *Let C be a connected component consisting of inner edges, forming a tree, and assume that C contains no edges from E_ℓ^{inner}. Then, with high probability, the longest path in C, considering its undirected edges, has a length of at most $r-1$.*

Proof. Proof is the same as Lemma 4.5.76 with proper parameters for level ℓ. □

Lemma 4.5.79 (Modification of Lemma 6.7 of [41] with Pseudo Edges)**.** *Let C be a connected component of inner edges corresponding to the edges of level lower than ℓ that is a tree such that there is no edge of E_ℓ^{inner} in C. Also, suppose that we condition on having a mixer vertex on all paths that contain at least $r-1$ special crossings. Then, the probability of seeing the same component is equal for both distributions up to $(1+O(n^{\sigma_{\ell-1}-\delta}))^{|C|}$ multiplicative factor.*

Proof of Lemma 4.5.67. Similar to the reasoning in the proofs of Lemma 4.5.38 and Lemma 4.5.68, the total shift in the probability of the coupling is bounded above by

$$\left(1+o(n^{2\delta-3\sigma_{\ell-1}-1})\right)^{|\overline{C}|}\cdot\left(1+O(n^{\sigma_{\ell-1}-\delta})\right)^{|\overline{C}|-1}\cdot\left(1+O(n^{\sigma_{\ell-1}-\delta})\right)\leq\left(1+O(n^{\sigma_{\ell-1}-\delta})\right)^{|\overline{C}|},$$

which completes the proof. □

4.5.8 Proof of Lemma 4.5.5

In this section, we complete the proof of Lemma 4.5.5.

Claim 4.5.80. *With probability $1-O(1/n)$, we have $|E_1^{inner}|=0$.*

Proof. First, note that by statement (iii) of Observation 4.5.42, we have $1 - g(\ell) < 0$. Therefore, applying statement (ii) of Lemma 4.5.43, we obtain

$$\begin{aligned}\Pr[E_1^{\text{inner}} = \emptyset] &\geq 1 - O(n^{1-g(1)}) \\ &\geq 1 - O(1/n) && \text{(Since } g(1) > 2 \text{ by statement (iii) of Observation 4.5.42)} \\ &= 1 - o(1).\end{aligned}$$

□

Claim 4.5.81. *With probability $1 - o(1)$, every connected component formed by queried edges at the base level of the hierarchy is a tree.*

Proof. By Claim 4.5.80, we know that $|E_1^{\text{inner}}| = 0$ with probability $1 - O(1/n)$. Conditioning on this event, we apply Claim 4.5.64, which states that with probability $1 - O(n^{-\delta+2\sigma_1+10\sigma_1/\sigma_2}) = 1 - o(1)$, all connected components formed by queried edges at the base level of the hierarchy are trees. This completes the proof. □

Claim 4.5.82. *Let V_B be the set of vertices for which the algorithm detects at least one incident edge at the base level. Let v be an arbitrary vertex in V_B. The total number of level-1 edges between v and other vertices of V_B, excluding level-1 edges discovered by the algorithm, is at most $\widetilde{O}(1)$ with high probability.*

Proof. By Claim 4.5.45, there are at most $O(n^{1-\delta+\sigma_1})$ vertices that have at least one level-1 edge. Hence, we have $|V_B| = O(n^{1-\delta+\sigma_1})$. Consider all pairs of vertices where one element of pair is v and the other one is from $V_B \setminus \{v\}$, excluding those the algorithm already queried. There are at most $O(n^{1-\delta+\sigma_1})$ such pairs, and each of them has ρn parallel ground edges, which means there are at most $O(\rho n^{2-2\delta+2\sigma_1})$ total ground edges between these pairs.

For a ground edge to be marked as a level-1 edge, it must be marked as a level-1 pseudo edge. Each ground edge is independently marked as a level-1 pseudo edge with probability $\rho_1/(\rho n)$. Thus, the expected number of level-1 pseudo edges between the corresponding pairs is at most

$$\begin{aligned}O\left(\rho n^{2-\delta+\sigma_1} \cdot \frac{\rho_1}{\rho n}\right) &= O\left(\rho_1 n^{1-\delta+\sigma_1}\right) \\ &= O\left(n^{-\delta+2\sigma_1}\right) && \text{(Since } \rho_1 = O(n^{\sigma_1-1})\text{)} \\ &\ll 1 && \text{(Since } \delta > 2\sigma_1\text{).}\end{aligned}$$

Since the expected number of such edges is smaller than one, the total number of level-1 edges between v and other vertices of V_B, excluding level-1 edges discovered by the algorithm, is at most $\widetilde{O}(1)$ with high probability. □

Lemma 4.5.83. *Suppose we condition on the event in Claim 4.5.80, where every connected component formed by queried edges at the base level of the hierarchy is a tree. Let $C_1, C_2, \ldots, C_k$ be the components of the forest found by the algorithm at the base level of the construction on a graph drawn from $\mathcal{D}_{\mathsf{YES}}$. Then, the probability of querying the same forest in a graph drawn from $\mathcal{D}_{\mathsf{NO}}$ is at least nearly as large, up to a multiplicative factor of $1 + o(1)$.*

Proof. By Lemma 4.5.78, the longest path in any component has length at most $r-1$. As a result, no path within any component contains $r-1$ special crossings, as defined in Definition 4.5.72. We aim to show that the probability of observing the same set of components in $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$ differs by at most a $1+o(1)$ multiplicative factor. Let $\mathcal{L}$ represent a labeling in $\mathcal{D}_{\mathsf{YES}}$. We construct a corresponding labeling $\mathcal{L}'$ in $\mathcal{D}_{\mathsf{NO}}$ and demonstrate that the probability of obtaining $\mathcal{L}'$ is nearly the same as that of $\mathcal{L}$.

We proceed by iterating over the components sequentially. Consider a component C. At each step, we condition on the labels that have already been revealed in $\mathcal{L}'$. If no edge within C connects two vertices in A_r, we assign the same labels in $\mathcal{L}'$, as all other edges remain identical in both $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$. Now, suppose there exists an edge (u,v) such that $u,v \in A_r$. Removing this edge results in two separate components, C_u and C_v. In $\mathcal{L}'$, we assign u to A_r and v to B_r. The labels of C_u and C_v are then coupled according to Lemma 4.5.79. This follows the same as the proofs of Lemma 4.5.67 and Lemma 4.5.68. Since each vertex in the component is connected to at most $\widetilde{O}(1)$ vertices with revealed labels, as stated in Claim 4.5.82, each coupling step contributes at most a $\widetilde{O}(1/n)$ shift in probability. Since the total number of steps corresponds to the number of edges queried by the algorithm at the base level of construction, which is $O(n^{1-\delta+\sigma_1})$, the overall shift in probability remains $o(1)$. Therefore, we can couple the two distributions in such a way that the probability of querying the same forest in both remains nearly identical, differing by at most a $1+o(1)$ multiplicative factor. □

Now we have all the tools to finish the proof of Lemma 4.5.5.

Lemma 4.5.5. *For every $\delta > 0$ there exists $\varepsilon > 0$ (i.e., ε is only a function of δ), take a* deterministic *sublinear algorithm that given a graph G drawn from $\mathcal{D}$, produces an εn-additive approximation of the matching size with probability $\frac{2}{3}$. That is, the algorithm computes $\tilde{\mu}(G)$ such that*

$$\Pr[\mu(G) - \varepsilon n \leq \tilde{\mu}(G) \leq \mu(G)] \geq \frac{2}{3}$$

where the probability is over the graph G drawn from $\mathcal{D}$. Then, the algorithm requires $\Omega(n^{2-\delta})$ adjacency matrix queries.

Proof. By Lemma 4.4.14, any algorithm that estimates the size of the maximum matching within an additive error of εn must be capable of distinguishing whether the input graph is drawn from $\mathcal{D}_{\mathsf{YES}}$ or $\mathcal{D}_{\mathsf{NO}}$. Moreover, Lemma 4.5.83 states that the distribution of outcomes observed by the algorithm differs by at most $o(1)$ in total variation distance between $\mathcal{D}_{\mathsf{YES}}$ and $\mathcal{D}_{\mathsf{NO}}$. Consequently, with a probability of at least $1/3$ over the randomness of the input distribution, the algorithm cannot differentiate between the supports of the two distributions. As a result, any deterministic algorithm that computes an estimate $\widetilde{\mu}$ for the size of the maximum matching in G, satisfying

$$\mu(G) - \varepsilon n \leq \widetilde{\mu} \leq \mu(G),$$

with probability $2/3$ over the randomness of the input distribution, must have a runtime of at least $\Omega(n^{2-\delta})$. □

Chapter 5

Applications of Sublinear Matching Algorithms

In this chapter, we explore various applications of the sublinear matching problem. This includes using sublinear matching in other settings of the maximum matching problem, such as the dynamic setting. We also examine the applications of maximum matching and its closely related problem, the maximal independent set (MIS), to several well-known problems, including the traveling salesman problem (TSP), Steiner tree, and Steiner forest.

In Section 5.1, we study the dynamic matching problem and its connection to sublinear matching. More specifically, for dynamic graphs, we prove that it is possible to achieve an approximation ratio of (almost) $2 - \sqrt{2} \approx 0.585$ with $\text{poly} \log n$ update time. In Section 5.2, we examine the traveling salesman problem and present an algorithm for estimating the size of TSP in two special cases: (1,2)-TSP and graphic TSP. Furthermore, in Section 5.3, we explore the relationship between the Steiner tree problem and sublinear matching, designing a faster algorithm for estimating the size of a Steiner Tree. Finally, in Section 5.4, we leverage sublinear algorithms for the maximal independent set problem to develop the first sublinear algorithm for the Steiner forest problem.

5.1 Maximum Matching in Dynamic Graphs

In this section, we prove the following results:

Theorem 5.1.1 (Formalized as Theorem 5.1.6)**.** *For any fixed $\varepsilon > 0$, there is an algorithm that maintains a $(2-\sqrt{2}-\varepsilon) \sim 0.585$-approximation of the size of the maximum matching in $\text{poly}(\log n)$ worst-case update time even against adaptive adversaries.*

Theorem 5.1.2. *For any fixed $\varepsilon > 0$, there is a deterministic two-pass streaming algorithm that finds (the edges of) a $(2-\sqrt{2}-\varepsilon) \sim 0.585$-approximate maximum matching using $O(n \log n)$ space.*

While much of the technicality of our work is on the dynamic algorithm of Theorem 5.1.1, both the streaming algorithm of Theorem 5.1.2 and its analysis are simple and clean.

5.1.1 Technical Overview

In this section, we give a brief overview of the technical challenges in designing our algorithms. We start with one of the existing algorithms for bipartite graphs and show why this algorithm does not perform well when the input graph is non-bipartite. We then discuss the new ingredients that we incorporate into our algorithm to achieve the same approximation ratio for non-bipartite graphs.

Let us consider the algorithm of Bhattacharya, Kiss, Saranurak, and Wajc [60] for bipartite graphs which achieves a 0.585-approximation. Let b_v be the capacity of vertex v. We define a b-matching to be a collection of edges of E such that each vertex v has at most b_v incident edges in the collection. Their algorithm has two main building blocks: (1) maintaining a maximal matching M, (2) a maximal b-matching B, in the bipartite graph between vertices of $V(M)$ and $V \setminus V(M)$. We let $\overline{V(M)} = V \setminus V(M)$ and let $G[V(M), \overline{V(M)}]$ be the induced bipartite graph between $V(M)$ and $\overline{V(M)}$. Since B is between matched and unmatched vertices by M, it becomes challenging to dynamically maintain B. This difficulty arises due to the fact that updates in M result in vertex updates in $G[V(M), \overline{V(M)}]$, and currently, there is no known algorithm capable of maintaining a matching under this type of update. To overcome this challenge, this algorithm uses the sublinear matching algorithm of Behnezhad [34] to estimate the size of B. Then, it is possible to estimate the size of the maximum matching of G as a function of only the sizes of M and B. For a specific value of $b = 1 + \sqrt{2}$ and sufficiently large k, when the capacity of vertices in $V(M)$ is k and the capacity of vertices in $\overline{V(M)}$ is $\lceil kb \rceil$ in the maximal b-matching, the algorithm achieves the approximation guarantee of $2 - \sqrt{2}$.

First challenge: parallel edges in the b-matching. It is important to note that the algorithm of [60] allows the b-matching to include the same edge multiple times. Consider the graph depicted in Figure 5.1. In this particular instance, our maximal matching M contains only the edge (u_1, u_2). Furthermore, the set B contains k duplicates of each of the two edges connected to u_4. Consequently, neither M nor B contains the dashed green edge, which implies that the approximation guarantee cannot be better than 0.5. In this example, it is vital for the algorithm to include unique edges in B instead of selecting an edge multiple times.

Since we store the b-matching explicitly in the streaming setting, it is easier to avoid including parallel edges. In Section 5.1.2, we provide a novel analysis of this algorithm via fractional matchings which relies on blossom inequalities. We prove that if we restrict the b-matching to pick each edge at most once, then the

Figure 5.1: This figure shows a non-bipartite graph such that if the algorithm allows multi-edges in the b-matching, it fails to achieve a large approximation guarantee. In particular. The left figure is the input graph. The right figure is a possible output of the algorithm. Here, the red edge denotes the maximal matching M, the red vertices denote $V(M)$, the blue edges show the maximal b-matching B, and the dashed green edge is not in $M \cup B$. Moreover, the number of copies of each edge in B is written next to it.

algorithm indeed obtains a $(2-\sqrt{2})$-approximation even for general graphs. More specifically, we construct a fractional matching x such that, except for the edges of two matchings, the fractional value of all edges is exceedingly small, i.e. $O(\varepsilon^3)$. In the first matching, the fractional value of each edge is precisely $1-1/b$, while in the second matching, this value is at most $1/b$. This characterization of the fractional matching, as established in the analysis, helps to show that the blossom inequality holds for vertex sets of size at most $O(1/\varepsilon)$. Consequently, we can utilize Proposition 2.2.5 to prove that $M \cup B$ contains an integral matching almost as large as the fractional matching.

However for the dynamic algorithm—specifically the part where the sublinear time algorithm of [34] is used to estimate the size of the b-matching—it is important to *allow* parallel edges. To see why this is true, note that the reduction from maximal b-matching to maximal matching relies on copying vertices with respect to their capacity. With this approach, one edge might be included in the maximal b-matching multiple times. To tackle this obstacle, we use the power of random greedy maximal matching when we find the maximal b-matching. Let M_1^* be the maximum matching edges of G that has exactly one matched endpoint in M. We show that for each edge $e \in M_1^*$, either it is included in B, or its capacity is saturated with many distinct incident edges. It is not hard to see that the following process is equivalent to constructing random greedy maximal matching: in each round, select one edge among the remaining edges uniformly at random, include it in the maximal matching, and remove both endpoints of the edge from the graph. Now, consider an edge e in the matching M_1^*. Let's examine the rounds in which one of the incident edges to e is chosen. If among these rounds, in many of them, the number of incident edges to e is small, then with high probability, at least one copy of e is going to be included in B. On the other hand, if in most of the rounds, the number of incident edges to e is large, we anticipate the copies to be distributed evenly. This is enough for us to show that $M \cup B$ contains a large matching via constructing a large fractional matching and exploiting blossom inequalities in the analysis.

Second challenge: estimating the output size. The second challenge arises when attempting to provide an estimate for the size of the maximum matching. Consider a graph that is a path of length three and its middle edge is in M and another graph that is a triangle in which one of its edges is in M. Note that $|B|$ is equal in both graphs (including the duplicate edges), i.e. in both graphs $|B|$ is equal to $2k$ since the capacity of vertices in $V(M)$ is $2k$ and we can use all their capacities. However, the maximum matching size differs: the first graph has a maximum matching size of 2, while the second graph has a maximum matching size of 1. Consequently, if the algorithm relies solely on the size of B to make its estimation, it cannot output

a value greater than 1. This results in an approximation guarantee of 0.5 for the first graph.

Nevertheless, subgraph $G[M \cup B]$ contains a large matching, i.e. in this example it contains all maximum matching edges of the original graph. Thus, all we need is to accurately estimate $\mu(G[M \cup B])$, i.e. to $(1-\varepsilon)$-approximate $\mu(G[M \cup B])$, to obtain the estimation for the maximum matching of the original graph. On the other hand, we cannot afford to construct the whole maximal b-matching B explicitly since we are using sublinear algorithms to build B. Instead, we can access to incident edges of a vertex v by spending $\widetilde{O}(n)$ time using the sublinear algorithm of [34]. Additionally, since the maximum degree of subgraph $G[M \cup B]$ is a constant, the total number of vertices in a close neighborhood of vertex v is a constant. This characteristic allows us to use maximum matching algorithms in the LOCAL model to estimate $\mu(G[M \cup B])$. By combining the previous two ideas, we can develop an algorithm that estimates $\mu(G[M \cup B])$ within a factor of $(1-\varepsilon)$ in $\widetilde{O}(n)$ time, which we can afford.

5.1.2 Warm-up: The Two-Pass Streaming Algorithm

In this section, we introduce our two-pass streaming algorithm for general graphs. We adopt a well-established framework (see [60, 140]) that has been commonly used in the literature for designing two-pass algorithms to find maximum matchings. The algorithm operates by first identifying a maximal matching in the initial pass, followed by obtaining a maximal b-matching in the subsequent pass. Specifically, during the first pass, we find a maximal matching denoted as M within the graph G. We set $b = (1 + \sqrt{2})$ and choose a large integer k that we will specify later. Moving on to the second pass, we find a maximal b-matching B in the bipartite graph $G[V(M), \overline{V(M)}]$ with a capacity of k and $\lceil kb \rceil$ for the vertices in $V(M)$ and $\overline{V(M)}$, respectively. Crucially, we do not allow B to contain multiple copies of an edge. Finally, we output the maximum matching obtained from the union of M and B.

Now let us discuss some of the key distinctions between our algorithm and the previous algorithms in this framework, which contribute to achieving an approximation ratio of $(2 - \sqrt{2})$ for general graphs. The first notable difference lies in our approach to selecting the value of k. It is crucial to choose k sufficiently large in our algorithm. This decision stems from the fact that if k is too small, many edges in the maximal b-matching might not effectively contribute to augmenting the maximal matching M. To illustrate this, consider the scenario where $k = 1$ and $(u, v) \in M$. Now, for a vertex $w \in \overline{V(M)}$, it is possible that both edges (u, w) and (v, w) are included in our maximal b-matching. However, this inclusion of both edges does not lead to any length-three augmenting paths, which are necessary for expanding the matching. To overcome this issue, it becomes essential to let k be sufficiently large. By doing so, we can avoid the problem mentioned above and ensure that the maximal b-matching includes edges that are truly beneficial for augmenting the maximal matching in the second pass.

In our proof, we construct a fractional matching and utilize Proposition 2.2.5 to prove that $M \cup B$ has a large matching. It is worth noting that in a recent work by Bhattacharya, Kiss, Saranurak, and Wajc [60], they also adopt the approach of selecting a sufficiently large value for k. However, a notable distinction arises in the second pass of their algorithm, specifically during the computation of the maximal b-matching. In their approach, they allow their algorithm to select an edge multiple times, whereas we do not. Allowing multiple copies of the same edge can lead to a situation where an edge belonging to the maximal matching has neighboring edges in the b-matching that are chosen multiple times, while certain other edges (i.e. edges of the optimal matching M^*) are not chosen at all. Consequently, this poses a challenge when attempting to

construct a fractional matching that satisfies the blossom inequality for subsets of vertices with small sizes.

Our algorithm is formalized in Algorithm 18. In the rest of this section, we prove the approximation guarantee of Algorithm 18.

Notation: Throughout this section, we let $G = (V, E)$ be the original graph. We use M to show the maximal matching that our algorithm finds in the first pass of the stream. Let $V(M)$ be the endpoints of M and $\overline{V(M)} = V \setminus V(M)$. Finally, let M^* be an arbitrary maximum matching of G, $M_1^* = M^* \cap (V(M) \times \overline{V(M)})$, and $M_2^* = M^* \cap (V(M) \times V(M))$.

Algorithm 18: Two-pass Streaming Algorithm for General Graphs

Parameter: let $b = 1 + \sqrt{2}$ and k be an integer larger than $\frac{1}{b\varepsilon^3}$.
First Pass: $M \leftarrow$ maximal matching of G. ▷ Finding maximal matching
Second Pass: ▷ Finding b-matching
Let $B = \emptyset$.
for $(u, v) \in G[V(M), \overline{V(M)}]$ *where* $u \in V(M)$ **do**
  if $\deg_B(u) < k$ *and* $\deg_B(v) < \lceil kb \rceil$ **then**
    $B \leftarrow B \cup (u, v)$.
return maximum matching of $M \cup B$.

Theorem 5.1.2. *For any fixed $\varepsilon > 0$, there is a deterministic two-pass streaming algorithm that finds (the edges of) a $(2 - \sqrt{2} - \varepsilon) \sim 0.585$-approximate maximum matching using $O(n \log n)$ space.*

Proof outline: To prove the theorem, we construct a fractional matching x on $M \cup B$. In Claim 5.1.3, we show the sum of x on M is at least $\left(1 - \frac{1}{b}\right)\left(|M_2^*| + \frac{1}{2}|M_1^*|\right)$. In Claim 5.1.4, we show that the sum of x on $G[V(M), \overline{V(M)}]$ is (almost) at least $\frac{1}{b+1}|M_1^*|$. As a result, we can conclude x has size (almost) at least $(2 - \sqrt{2})\mu(G)$. In Claim 5.1.5, we show that $(1 - \varepsilon)x$ satisfies the conditions of Proposition 2.2.5 to prove $M \cup B$ has an integral matching (almost) as large as x. Finally, we put all this together to complete the proof.

First, we describe the construction of the fractional matching x on $M \cup B$. For all edges $e \in M$, we let $x_e = 1 - \frac{1}{b}$. For all the edges $e \in B \setminus M_1^*$, we let $x_e = \frac{1}{\lceil kb \rceil}$. Finally, for every edge $(u, v) \in B \cap M_1^*$ with $u \in V(M)$ and $v \in \overline{V(M)}$, we let $x_e = \frac{t}{\lceil kb \rceil}$ where t is equal to $\min(k - \deg_B(u), \lceil kb \rceil - \deg_B(v))$. Informally, for the analysis, we keep adding copies of (u, v) to B as long as B remains a b-matching. This is a key fact in the proof of Claim 5.1.4.

Notice that any vertex in $V(M)$ is adjacent to at most one edge from M and k edges from B. Therefore, the sum of x on the adjacent edges in M is at most $1 - \frac{1}{b}$, and the sum on adjacent edges in B is at most $k \cdot \frac{1}{\lceil kb \rceil} \leq \frac{1}{b}$. Similarly, any vertex in $\overline{V(M)}$ is adjacent to at most $\lceil kb \rceil$ edges from B, and is not adjacent to any edges of M. Therefore, the sum of x on the adjacent edges is at most $\lceil kb \rceil \cdot \frac{1}{\lceil kb \rceil} = 1$. Hence, x is a fractional matching.

Claim 5.1.3. *It holds that $x(M) \geq \left(1 - \frac{1}{b}\right)\left(|M_2^*| + \frac{1}{2}|M_1^*|\right)$.*

Proof. Consider the vertices of $V(M)$ and how they are covered by the edges of M^*. There are three possibilities: the vertex is also covered by M_1^*; the vertex is also covered by M_2^*; or the vertex is not covered

by M^*. Notice that there are $|M_1^*|$ vertices of the first type, and $2|M_2^*|$ of the second type. Therefore, it holds:

$$|V(M)| \geq |M_1^*| + 2|M_2^*|,$$

and since $|M| = \frac{1}{2}|V(M)|$:

$$|M| \geq |M_2^*| + \frac{1}{2}|M_1^*|.$$

Given that the value of x on the edges of M is $1 - \frac{1}{b}$, the claim follows. □

Claim 5.1.4. *It holds that* $x(B) \geq (1-\varepsilon)\frac{1}{b+1}|M_1^*|$.

Proof. First, we introduce some definitions. For every edge $e \in B$ define t_e equal to $x_e \cdot \lceil kb \rceil$, i.e. t_e is an integer such that $x_e = \frac{t_e}{\lceil kb \rceil}$. Also, define $t(u)$ as the sum of t on its adjacent edges, that is:

$$t(u) = \sum_{e \in \delta_B(u)} t_e.$$

Notice, for every $u \in V(M)$, it holds that $t(u) \leq k$, and for every $v \in \overline{V(M)}$, it holds that $t(v) \leq \lceil kb \rceil$. Furthermore, for every $(u,v) \in M_1^*$ with $u \in V(M)$ and $v \in \overline{V(M)}$, it holds that $t(u) = k$ or $t(v) = \lceil kb \rceil$.

We use a charging argument. We order the edges of B arbitrarily as $e_1, \ldots, e_N$, and let B_i be the set of first i edges. With respect to B_i, we define a potential ϕ_i on every edge $(u,v) \in M_1^*$ with $u \in V(M)$ and $v \in \overline{V(M)}$:

$$\phi_i(u,v) = \max\left(\frac{\sum_{e \in \delta_{B_i}(u)} t_e}{k}, \frac{\sum_{e \in \delta_{B_i}(v)} t_e}{\lceil kb \rceil} \right),$$

which is equal to the maximum fraction of the used capacity on its endpoints. We also let:

$$\phi_i = \sum_{(u,v) \in M_1^*} \phi_i(u,v).$$

For an edge e_i, we charge it $c_i = \phi_i - \phi_{i-1}$. For each edge e_i, it holds that $c_i \leq t_{e_i} \cdot \left(\frac{1}{k} + \frac{1}{\lceil kb \rceil}\right)$. Because it is adjacent to at most two edges of M_1^*, and it can increase the potential on either one by at most $\frac{t_{e_i}}{k}$ and $\frac{t_{e_i}}{\lceil kb \rceil}$ respectively. Therefore, we have:

$$\phi_N = \sum_{i=1}^{N} c_i \leq \left(\frac{1}{k} + \frac{1}{\lceil kb \rceil}\right) \sum_{i=1}^{N} t_{e_i}. \tag{5.1}$$

It also holds that $\phi_N(u,v) = 1$ for every $(u,v) \in M_1^*$. To show this, we examine two cases. If $(u,v) \in B$, that is $(u,v) = e_i$ for some i, then $\phi_j(u,v)$ is equal to one for all $j \geq i$. If $(u,v) \notin B$, then at the point in the stream that (u,v) arrived, at least one endpoint must have been saturated by the edges of B, i.e. $\phi_i(u,v)$ is equal to one whenever B_i includes all the adjacent edges (u,v) in B. Therefore, we have:

$$\phi_N = |M_1^*|. \tag{5.2}$$

Putting (5.1) and (5.2) together we get:

$$|M_1^*| \leq \left(\frac{1}{k} + \frac{1}{\lceil kb \rceil}\right) \sum_{i=1}^{N} t_{e_i},$$

or equivalently:

$$\sum_{i=1}^{N} t_{e_i} \geq \frac{k \cdot \lceil kb \rceil}{k + \lceil kb \rceil} |M_1^*| = \frac{k + kb}{k + \lceil kb \rceil} \cdot \frac{\lceil kb \rceil}{b+1} |M_1^*| \geq (1-\varepsilon)\frac{\lceil kb \rceil}{b+1} |M_1^*|$$

Given the fact that $x(e_i) = \frac{t_{e_i}}{\lceil kb \rceil}$, it follows:

$$x(B) = \frac{1}{\lceil kb \rceil} \sum_{i=1}^{N} t_{e_i} \geq (1-\varepsilon)\frac{1}{b+1} |M_1^*|. \qquad \square$$

Claim 5.1.5. *$M \cup B$ contains an integral matching of size $(1-\varepsilon)^2 \sum_e x_e$.*

Proof. To prove the statement, we show that $(1-\varepsilon)x$ satisfies the conditions of Proposition 2.2.5. That is, we prove x satisfies $x(S) \leq \left\lfloor \frac{|S|}{2} \right\rfloor$ for every vertex set $S \subseteq V$ of size at most $\frac{1}{\varepsilon}$. Notice that since $(1-\varepsilon)x$ is a fractional matching the inequality holds for any set S with an even size. Also, notice that if x satisfies the inequality for a set S, then so does $(1-\varepsilon)x$. This leaves us with one case.

Take a vertex set S of size equal to $2s+1 \leq \frac{1}{\varepsilon}$ such that x does not satisfy the condition, i.e. $s < x(S) \leq s+1$. For any edge $e \in M$, we have $x_e \leq 1 - \frac{1}{b}$, for any edge $e \in B \cap M_1^*$, we have $x_e \leq \frac{1}{b}$, and for any edge $e \in B \setminus M_1^*$, we have $x_e \leq \frac{1}{\lceil kb \rceil}$. Given the fact that there are at most s edges of M and $B \cap M_1^*$ in S, we can conclude:

$$\begin{aligned}
x(S) &= x(M) + x(B \cap M_1^*) + x(B \setminus M_1^*) \\
&\leq \left(1 - \frac{1}{b}\right)|M| + \frac{1}{b}|B \cap M_1^*| + \frac{1}{\lceil kb \rceil}|S|^2 \\
&\leq \left(1 - \frac{1}{b}\right)s + \frac{1}{b}s + \frac{1}{\lceil kb \rceil}\frac{1}{\varepsilon^2} \\
&\leq s + \varepsilon && (k \geq \tfrac{1}{b\varepsilon^3})
\end{aligned}$$

Therefore, we have:

$$(1-\varepsilon)x(S) \leq (1-\varepsilon)(s+\varepsilon) \leq s + \varepsilon - s\varepsilon - \varepsilon^2 \leq s - \varepsilon^2.$$

The claim follows from applying Proposition 2.2.5 to $(1-\varepsilon)x$. $\square$

Proof of Theorem 5.1.2. First, we use Claims 5.1.3 and 5.1.4 to show $\sum_e x_e \geq (1-\varepsilon)(2-\sqrt{2})\mu(G)$. It holds that:

$$\begin{aligned}
\sum_e x_e &= x(M) + x(B) \\
&\geq \left(1 - \frac{1}{b}\right)\left(|M_2^*| + \frac{1}{2}|M_1^*|\right) + (1-\varepsilon)\frac{1}{b+1}|M_1^*| && \text{(Claims 5.1.3 and 5.1.4)}
\end{aligned}$$

$$\geq (1-\varepsilon)\left[\left(1-\frac{1}{b}\right)|M_2^*| + \left(\frac{1}{2}-\frac{1}{2b}+\frac{1}{b+1}\right)|M_1^*|\right].$$

Since $b = 1+\sqrt{2}$, we have $1-\frac{1}{b} = \frac{1}{2}-\frac{1}{2b}+\frac{1}{b+1} = 2-\sqrt{2}$. Therefore,

$$\sum_e x_e \geq (1-\varepsilon)(2-\sqrt{2})(|M_1^*|+|M_2^*|) = (1-\varepsilon)(2-\sqrt{2})\mu(G).$$

To complete the proof, we note that by Claim 5.1.5, $M \cup B$ contains a matching of size $(1-\varepsilon)^3(2-\sqrt{2})\mu(G) \geq (1-\varepsilon)^3 \cdot .585 \cdot \mu(G)$. Also, Algorithm 18 stores $O(n)$ edges for M and $O(n \operatorname{poly} \frac{1}{\varepsilon})$ edges for B. Hence, it uses space $O(n \operatorname{poly} \frac{1}{\varepsilon})$. Replacing ε by $\frac{\varepsilon}{3}$ gives the theorem. □

5.1.3 The Fully Dynamic Algorithm

In this section, we show how we can turn our two-pass streaming algorithm in Section 5.1.2 into a fully dynamic algorithm with polylogarithmic update time. More formally, we prove the following theorem.

Theorem 5.1.6. *For any $\varepsilon > 0$, there is a fully dynamic algorithm that maintains a $(2-\sqrt{2}-\varepsilon) \sim 0.585$-approximation of the size of maximum matching in $2^{\operatorname{poly}(1/\varepsilon)} \cdot \operatorname{poly}(\log n)$ worst-case update time in general graphs. The algorithm is randomized but works against adaptive adversaries.*

Prior to delving into the algorithm and proofs, we define a setting known as a *semi-dynamic* setting. In this context, an algorithm $\mathcal{A}$ is categorized as semi-dynamic if it only generates an estimation of the maximum matching size when prompted with a query. By known reductions [32, 60, 137], such semi-dynamic algorithms can be transferred into fully dynamic algorithms that are capable of maintaining the maximum matching size continuously and not only upon receiving a query. The following lemma from [32], formalizes this:

Proposition 5.1.7 (Lemma 4.1 in [32])**.** *For a fully dynamic graph G and $\varepsilon > 0$, suppose there is a data structure $\mathcal{A}$ that takes $U(n)$ worst-case time per update to G and provides an estimate $\widetilde{\mu}$ in $Q(n,\varepsilon)$ time upon being queried, satisfying $\alpha \cdot \mu(G) - \varepsilon n \leq \mathbf{E}[\widetilde{\mu}] \leq \mu(G)$. Then, there exists a randomized data structure $\mathcal{B}$ that maintains an estimate $\widetilde{\mu}'$ such that, throughout the updates, $(\alpha-\varepsilon)\cdot\mu(G) \leq \widetilde{\mu}' \leq \mu(G)$ with high probability. Additionally, $\mathcal{B}$ has a worst-case update time of $O\left(\left(U(n)+\frac{Q(n,\varepsilon^2)}{n}\right)\cdot\operatorname{poly}(\log n, 1/\varepsilon)\right)$. Moreover, if $\mathcal{A}$ can handle an adaptive adversary, then $\mathcal{B}$ can as well.*

The proof of the aforementioned lemma is built upon the concept of "vertex sparsification," which has been previously used in the literature [19, 137]. In this work, we show that there is a semi-dynamic algorithm that for any general graph G achieves $U(n) = \operatorname{poly}(\log n)$, $Q(n,\varepsilon) = n \cdot 2^{\operatorname{poly}(1/\varepsilon)} \cdot \operatorname{poly}(\log n)$, $\alpha = 2-\sqrt{2}$, and works against adaptive adversaries. Plugged into Proposition 5.1.7, this implies our desired Theorem 5.1.1.

The first step to design such a dynamic algorithm is to simulate the first pass of the original streaming algorithm. The problem of maintaining maximal matching in a dynamic setting has been extensively studied in the field and there exists a rich literature that leads to fully dynamic algorithms with polylogarithmic update time (see [31, 49, 170]). We use the following result as one of the building blocks of our algorithm.

Proposition 5.1.8 ([31])**.** *There exists a data structure that maintains a maximal matching in a fully dynamic graph with $\operatorname{poly}(\log n)$ worst-case update time against an oblivious adversary.*

In order to simplify our algorithm, we incorporate the assumption of having an oblivious adversary as stated in Proposition 5.1.8. However, it is important to note that this assumption is only employed at this

specific point in the outline of our algorithm. Toward the end, we will explain how we can eliminate this assumption.

Let M be the maximal matching that we maintain and H be the induced bipartite subgraph between matched and unmatched vertices, i.e. $G[V(M), \overline{V(M)}]$. To simulate the second pass of our streaming algorithm, we incorporate a sublinear algorithm for estimating the size of the maximum matching. A similar approach was utilized in [32, 60] to achieve an equivalent approximation ratio for bipartite graphs. For a more comprehensive understanding of the reduction from a dynamic matching algorithm to a sublinear matching algorithm, refer to [32, 60]. In the second pass of the streaming algorithm for bipartite graphs, even if we select the same edge multiple times in the maximal b-matching, we can still demonstrate the same approximation guarantee. However, when dealing with general graphs, it is necessary to choose distinct edges in order to attain a significant approximation ratio. Let B be a maximal b-matching in H. In Section 5.1.2, we proved that $M \cup B$ contains a 0.585-approximate matching. A second technical challenge arises at this point. Unlike the algorithm presented in [60] for bipartite graphs, where the size of B is adequate for estimation, we now need to estimate the size of the maximum matching of $M \cup B$ accurately. More formally, we need a $(1-\varepsilon)$-approximation of $\mu(G[M \cup B])$ to achieve our approximation guarantee.

Our primary technical contribution in implementing our two-pass streaming algorithm in the fully dynamic setting involves addressing the above two challenges. If we are not restricted in selecting distinct edges for maximal b-matching, we can simplify the process by creating multiple copies of $\overline{V(M)}$ (specifically, $\lceil kb \rceil$ copies) and $V(M)$ (k copies). This reduction allows us to convert the maximal b-matching into an instance of maximal matching, for which we already have a fast sublinear algorithm available [34] (similar to the approach used in [60]).

Proposition 5.1.9 ([34, 32])**.** *Let $\varepsilon > 0$, v be a random vertex in graph G, and π be a random permutation over edges of G. There exists an algorithm that determines if v is matched in $\mathrm{GMM}(G, \pi)$ that works in $\widetilde{O}(n/\varepsilon)$ expected time with a success probability of $1-\varepsilon$. Moreover, if v is matched, the algorithm returns the matching edge.*

However, with this reduction, the possibility of selecting an edge multiple times arises, preventing us from obtaining the desired approximation guarantee for general graphs. It is worth noting that the algorithm in [34] estimates the size of the randomized greedy maximal matching. To overcome the aforementioned challenge, we leverage the observation that when we run the randomized greedy maximal matching on the maximal b-matching instance, each edge in M_1^* will either be selected at least once or has one endpoint that is nearly saturated with distinct edges in the maximal b-matching. This observation allows us to achieve the same approximation guarantee, disregarding some dependence on ε.

For the second challenge, we need to design an oracle that, given a vertex v as input, can determine whether v is part of an approximately optimal maximum matching of $M \cup B$. We will then apply this oracle to several randomly selected vertices to estimate $\mu(G[M \cup B])$. To design the oracle, we can exploit the fact that the maximum degree of $G[M \cup B]$ is constant. This enables us to utilize existing LOCAL algorithms for maximum matching, as the number of vertices within a bounded distance from the queried vertex is at most a certain constant.

Proposition 5.1.10 ([120])**.** *For $\varepsilon > 0$, there exists a $O(\varepsilon^{-3} \log \Delta)$-round LOCAL algorithm that outputs $(1-\varepsilon)$-approximate maximum matching in expectation.*

Note that we do not have direct access to the adjacency matrix of graph $G[M \cup B]$. However, we can utilize Proposition 5.1.9 to identify all the maximal b-matching edges of a vertex v. This allows us to obtain the neighbors of v in $M \cup B$ within a time complexity of $\widetilde{O}_\varepsilon(n)$. Consequently, by spending $\widetilde{O}_\varepsilon(n)$ time, we can obtain all the vertices that are at a distance of $O(\varepsilon^{-3} \log \Delta)$ from a given vertex.

In the rest of this section, we provide formal proof of the approximation guarantee and running time of Algorithm 19.

Notation: Throughout this section, we let $G = (V, E)$ be the original graph that undergoes edge deletion and insertion. We use M to show the maximal matching that our algorithm maintains. Let $V(M)$ be the endpoints of M and $\overline{V(M)} = V \setminus V(M)$. Let $H \coloneqq G[V(M), \overline{V(M)}]$ and $\widetilde{H}$ be the graph constructed by having k copies of vertices of $V(M)$ and $\lceil kb \rceil$ copies of $\overline{M}$. Additionally, let $\widetilde{B}$ denote the random greedy maximal matching of $\widetilde{H}$ and B be the corresponding b-matching on H. Finally, let M^* be an arbitrary maximum matching of G, $M_1^* = M^* \cap (V(M) \times \overline{V(M)})$, and $M_2^* = M^* \cap (V(M) \times V(M))$.

Algorithm 19: Semi-Dynamic Algorithm for General Graphs

1 Let M be the maximal matching of G that we maintain using Proposition 5.1.8.
2 Let $H = G[V(M), \overline{V(M)}]$, and $\widetilde{H}$ be the auxiliary graph based on H ($\widetilde{H}$ is not constructed explicitly).
3 Let π be a random permutation over edges of $\widetilde{H}$ (π is not constructed explicitly).
4 Let $\widetilde{B}$ be a random greedy maximal matching on $\widetilde{H}$ with respect to π, and let B be the corresponding b-matching on H ($\widetilde{B}$ and B are not constructed explicitly).
5 Sample $r = 24\varepsilon^{-2} \log n$ random vertices $v_1, \ldots, v_r$ from V.
6 Let X_i be the indicator variable for the event that v_i is matched in the $(1-\varepsilon)$-approximate maximum matching of $M \cup B$ computed via Proposition 5.1.10.
7 Let $X = \sum_{i=1}^{r} X_i$ and $\widetilde{\mu} = \frac{nX}{2r} - \frac{\varepsilon n}{2}$.

Approximation Ratio

This subsection is devoted to proving the following claim, that is, $G[M \cup B]$ approximates the maximum matching of G.

Claim 5.1.11. *It holds that* $(2 - \sqrt{2} - \varepsilon)\mu(G) \leq \mathbf{E}\left[\mu(G[M \cup B])\right] \leq \mu(G)$.

Remark 10. *Throughout this subsection, to make the proof more simple, we assume that M is a maximal matching in G, and B is a maximal b-matching in H. Whereas, M is an almost maximal matching and the edges of B may be "missed" with probability ε. Toward the end, we show how these assumptions can be lifted.*

To prove the claim, we adopt a similar strategy to Section 5.1.2. We construct a large fractional matching x on $M \cup B$ and then show $M \cup B$ has an integral matching almost as large as x. Finally, we conclude that $\mu(G[M \cup B])$ approximates $\mu(G)$. Since we now allow B to contain multiple copies of each edge and G may be non-bipartite, the same argument no longer works. However, we can show that B has certain properties and the claim still holds. We prove:

Claim 5.1.12. *For every edge $e \in M_1^*$, the following holds with probability at least $1 - \varepsilon$: Either e appears in B, or for an endpoint u of e, there exists a multiset of edges $F \subseteq B$ such that $|F| \geq (1 - 2\varepsilon)b(u)$ and*

no edge appears in F more than $\varepsilon^3\lceil kb\rceil$ times. Where $b(u)$ is the capacity of u in the b-matching, i.e. k if $u \in V(M)$ and $\lceil kb\rceil$ if $u \in \overline{V(M)}$.

We can view the process of finding a random greedy maximal matching in $\widetilde{H}$ as follows: In each step, a corresponding copy of edge $(u, v) \in H$ is picked with probability proportional to its weight $w(u, v) = r(u)r(v)$. Where $r(u)$ is the number of remaining vertices in $\widetilde{H}$ corresponding to u. Afterward, both endpoints are deleted from $\widetilde{H}$. Note that $r(u)$ can also be regarded as the unused capacity of u.

Now, we fix an edge (u, v) in M_1^* and define the edge set $I \subseteq E(H)$ equal to (u, v) plus the set of edges adjacent to (u, v), and let $I' = I \setminus \{(u, v)\}$. The process alternates between picking an edge from I and picking a number of edges from $E(H) \setminus I$. We focus on the steps where an edge from I is picked and model the steps that happen outside of I with an adversary. Note that this adversary does not really exist. It only represents the complex process of picking edges from $E(H) \setminus I$.

More formally, we model the process with a chain of steps, each composed of two parts. The first part corresponds to picking edges from $E(H) \setminus I$, and the second part corresponds to picking an edge from I. In the first part, the adversary is given the number of times each edge of I has been picked so far, hereafter referred to as the load of the edges. It will then decide the weights w on I' for the next part (the weight of (u, v) is uniquely determined by the load of the edges in I). In the second part, one of the edges in I is picked at random with probability proportional to its weight.

Note that in the original process, the weights w should satisfy certain constraints. For example, the weights should be non-increasing throughout the process and the weight of any edge (u', v') should be determined by the underlying values $r(u')$ and $r(v')$ which in turn have constraints of their own based on the loads. We allow for a stronger adversary by disregarding many of these constraints and imposing only a few of them. For now, we impose:

1. In every step, every edge has an integer weight in $[0, k \cdot \lceil kb\rceil]$; and
2. the total weight of I', hereafter referred to as $w(I')$, is non-increasing throughout the process (this implies the total weight of I is also non-increasing since $w(u, v)$ is also non-increasing).

As a first step in proving Claim 5.1.12, we show the following claim is true. It roughly states that if in many steps, (u, v) has a large weight compared to the total weight of I, then (u, v) is likely to be picked by the process.

Claim 5.1.13. *For an edge $(u, v) \in M_1^*$ with $u \in V(M)$ and $v \in \overline{V(M)}$, considering the prefix of steps where $r(u) \geq \varepsilon k$ and $r(v) \geq \varepsilon\lceil kb\rceil$, if there are more than $s = 2\frac{\log(1/\varepsilon)}{\varepsilon^6}$ steps where $w(I')$ is less than $W = \lfloor\varepsilon^{-4}\rfloor \cdot k \cdot \lceil kb\rceil$, then (u, v) is picked with probability at least $(1 - \varepsilon)$.*

Proof. Notice that since $r(u) \geq \varepsilon k$ and $r(v) \geq \varepsilon\lceil kb\rceil$, it holds that $w(u, v) \geq \varepsilon^2 \cdot k \cdot \lceil kb\rceil$. Also, $w(I')$ is at most W in s of the steps. Therefore, the probability that (u, v) is *not* picked in any of these s steps is at most:

$$\left(1 - \frac{\varepsilon^2 \cdot k \cdot \lceil kb\rceil}{\varepsilon^2 \cdot k \cdot \lceil kb\rceil + W}\right)^s \leq \left(1 - \frac{\varepsilon^{-6}}{2}\right)^s \leq \exp\left(-\frac{s\varepsilon^{-6}}{2}\right) \leq \varepsilon. \qquad \square$$

To complete the proof of Claim 5.1.12, we restrict our attention to the cases where the probability of (u, v) being picked is smaller than $(1 - \varepsilon)$. Therefore, due to Claim 5.1.13 we can assume that among the steps where $r(u) \geq \varepsilon k$ and $r(v) \geq \varepsilon\lceil kb\rceil$, all but $2\frac{\log(1/\varepsilon)}{\varepsilon^6}$ of them have $w(I')$ larger than $\lfloor\varepsilon^{-4}\rfloor \cdot k \cdot \lceil kb\rceil$.

We call them the *early steps*. Note that these steps form a prefix of the steps since $w(I')$ is non-increasing throughout the process. We prove that, as a result, with probability $(1-\varepsilon)$ the maximum number of times an edge of I' is picked in these steps is at most $\varepsilon^3\lceil kb\rceil$. Intuitively, since $w(I')$ is very large, we expect each edge to be picked ε^4 fraction of the time, and no edge has the chance to be picked many times, say an ε^3 fraction of the time.

To prove this, we characterize the adversary that maximizes the probability of the maximum load being larger than $T = \varepsilon^3\lceil kb\rceil$ after the early steps. We use τ to denote this probability and we call an adversary optimal if it maximizes τ. We say an adversary is greedy if in the early steps, it assigns weight $k \cdot \lceil kb\rceil$ to the $\lfloor\varepsilon^{-4}\rfloor$ edges that have the highest loads (breaking ties arbitrarily), and assigns zero weight to the others, i.e. $w(I')$ is exactly equal to W and it is distributed among the edges with the highest loads. Claim 5.1.14 states that the greedy adversary is optimal. Informally, we are stating that the worst thing that can happen is that the $w(I')$ is always equal to W in the early steps and the weight is concentrated on the edges with the highest loads. Throughout the proof, we assume (u, v) is never picked.

Claim 5.1.14. *Among the adversaries that satisfy the following conditions:*

1. *In every step, every edge has an integer weight in* $[0, k \cdot \lceil kb\rceil]$*;*
2. $w(I')$ *is non-increasing throughout the process; and*
3. *for all but* $2\frac{\log(1/\varepsilon)}{\varepsilon^6}$ *many of the steps such that* $r(u) \geq \varepsilon k$ *and* $r(v) \geq \varepsilon\lceil kb\rceil$*, we have* $w(I')$ *larger than* $W = \lfloor\varepsilon^{-4}\rfloor \cdot k \cdot \lceil kb\rceil$*;*

the greedy adversary is optimal.

Proof. Let $|I'| = q$. Let $l_1 \geq l_2 \geq \ldots \geq l_q$ be the loads of the edges so far. We refer to the multiset of loads as the *load profile*. Note that for an optimal adversary, τ depends only on the load profile and it does not matter exactly which edge has which load. We use J to denote the set of the $\lfloor\varepsilon^{-4}\rfloor$ edges with the highest loads. An adversary is greedy if it sets $w(J) = W$ and $w(I' \setminus J) = 0$ in every step. This way, an edge from J is picked uniformly at random in every step and no other edge is ever picked. We refer to the $\lfloor\varepsilon^{-4}\rfloor$ highest loads as the *upper load profile*. For the greedy adversary, τ depends only on the upper load profile. We say an upper load profile $L_1^{(1)} \geq L_2^{(1)} \geq \ldots \geq L_{\lfloor\varepsilon^{-4}\rfloor}^{(1)}$ dominates another upper load profile $L_1^{(2)} \geq L_2^{(2)} \geq \ldots \geq L_{\lfloor\varepsilon^{-4}\rfloor}^{(2)}$ if for all i, it holds that $L_i^{(1)} \geq L_i^{(2)}$. When $L^{(1)}$ dominates $L^{(2)}$ and the adversary acts greedily, starting from a $L^{(1)}$ leads to a higher value of τ than starting from $L^{(2)}$.

We prove the claim by induction on the remaining number of early steps, n. For $n = 1$, the claim is trivial. Since there is only one step remaining, τ is maximized when the edges with higher loads have the maximum probability of being picked. Therefore, given any weight assignment, if there is an edge $e \notin J$ with positive weight, we can either transfer some of e's weight to J (when $w(J) < W$), or delete some of e's weight (when $w(J) = W$), and τ would grow.

For $n > 1$, we can assume by induction that whatever happens in this step, from the next step forward, it is optimal for the adversary to act greedily. Therefore, for the adversaries we examine in the rest of this proof, we assume they act greedily after the current step. Now, take any non-greedy weight assignment w. Given that the weights are set to w in this step, let $\tau(w)$ be the probability that after the early steps finish, the maximum load is larger than $T = \varepsilon^3\lceil kb\rceil$. We alter w slightly to obtain a weight assignment w' such that $\tau(w') \geq \tau(w)$ (where $\tau(w')$ is defined similarly to $\tau(w)$). There are two cases.

First, consider the case where $w(J) < W$. In this case, there must be a an edge $e \notin J$ with $w(e) > 0$, since we have $w(I') \geq W$. Take such an edge e with the lowest load. Also, there must be an edge $e' \in J$ with $w(e') < k \cdot \lceil kb \rceil$, otherwise it would have held $w(J) = W$. We transfer a unit of weight from e to e'. That is, we define $w'(e) = w(e) - 1$, $w'(e') = w(e) + 1$, and let w' be equal to w for every other edge. To show $\tau(w') \geq \tau(w)$, loosely we can say that except for the instances where e' is picked instead of e, the two weight assignments lead to the same outcome. Therefore, we only need to show that e' being picked instead of e in this step, leads to a better chance of the maximum load exceeding T in the rest of the early steps. This is intuitively true because e' has a greater load than e.

Formally, each of these two weight assignments leads to a process of n steps. We introduce a coupling for them as follows: Consider an outcome of the first process, starting with weights w. If the first process has picked e in the current step, then with probability $\frac{1}{w(e)}$ (i.e. overall probability $\frac{1}{w(I')}$) we assume the second process picks e' in this step and carries on independently of the first process (this corresponds to the alteration in the weight assignment). Otherwise, we let the second process have the exact same outcome as the first process. It can be easily seen that the second process created here, has the same outcome distribution as an independent process that starts with weights w'.

To show $\tau(w') \geq \tau(w)$, it suffices to prove that when e' is selected in this step, then the probability of the maximum load going over T in the next $n-1$ steps is larger than when e is selected. Because the adversary acts greedily in the next steps, we only need to examine the upper load profiles. Let L be the current upper load profile, let $L^{(1)}$ be the upper load profile resulting from picking e, and $L^{(2)}$ be the upper load profile resulting from picking e'. We use $\tau(L^{(1)})$ to denote the probability of the maximum load going over T after the next $n-1$ steps are carried out with the greedy adversary, when the initial load is $L^{(1)}$. We define $\tau(L^{(2)})$ similarly. We need to show $\tau(L^{(2)}) \geq \tau(L^{(1)})$. Let γ be $\lfloor \varepsilon^{-4} \rfloor$-th highest load, i.e. $l_{\lfloor \varepsilon^{-4} \rfloor} = \gamma$. We consider two cases. If $l(e) \leq \gamma - 1$, then when e is picked the upper load profile does not change, i.e. $L^{(1)} = L$, since there are already $\lfloor \varepsilon^{-4} \rfloor$ edges with load larger than L. Meaning that picking e is as good as picking no edges this round because it will not change the upper load profile. Also, $L^{(2)}$ dominates L. Therefore, $L^{(2)}$ dominates $L^{(1)}$, and as a result $\tau(L^{(2)}) \geq \tau(L^{(1)})$.

Now, consider the case where $l(e) = \gamma$. In this case, if e is picked, then $l(e)$ becomes $\gamma + 1$. As a result, in the upper load profile, an element γ is replaced with $\gamma + 1$. That is, we have:

$$L^{(1)} = L \setminus \{l(e)\} \cup \{l(e) + 1\} \qquad \text{and} \qquad L^{(2)} = L \setminus \{l(e')\} \cup \{l(e') + 1\}$$

In Claim 5.1.15, we prove that $\tau(L^{(2)}) \geq \tau(L^{(1)})$. To apply Claim 5.1.15, note that $l(e') \geq l(e)$. This completes the proof of $\tau(w') \geq \tau(w)$ for when $w(J) < W$.

The claim follows similarly when $w(J) = W$. There must be an edge $e \notin J$ such that $w(e) > 0$, otherwise the weight assignment would indeed be greedy. Take such an edge e with the lowest load. We define $w'(e) = w(e) - 1$ and let w' be equal to w on all the other edges. For the coupling, when the first process picks edge e, with probability $\frac{1}{w(e)}$ the second process will randomly pick an edge from I' with probability proportional to w'. Otherwise, we let the outcomes be the same. We still have to examine two cases where $l(e) \leq \gamma - 1$ and $l(e) = \gamma$. Note that in the case where another edge e' is selected instead of e, it holds that $l(e') \geq l(e)$.

By a series of the two types of alterations we have discussed, w can be transformed into the greedy assignment of weights. As proved above, with each alteration, τ will not decrease. Therefore, the greedy

adversary is optimal for this step as well. This completes the step of the induction and concludes the proof. □

Claim 5.1.15. *Let $L_1, \ldots, L_{\lfloor \varepsilon^{-4} \rfloor}$ be an upper load profile. Let i and j be indices such that $L_i \geq L_j$ and define*

$$L^{(1)} = \left\{L_1, \ldots, L_i, \ldots, L_j + 1, \ldots, L_{\lfloor \varepsilon^{-4} \rfloor}\right\},$$

and

$$L^{(2)} = \left\{L_1, \ldots, L_i + 1, \ldots, L_j, \ldots, L_{\lfloor \varepsilon^{-4} \rfloor}\right\}.$$

Then it holds that $\tau(L^{(2)}) \geq \tau(L^{(1)})$. Where $\tau(L)$ denotes the probability that the maximum load exceeds T after n steps are carried out with the greedy adversary.

Proof. We prove the claim by induction. For $n = 0$, it holds trivially since the maximum element of $L^{(2)}$ is at least as large as the maximum element of $L^{(1)}$. For $n > 0$, if $L_i = L_j$, then $L^{(1)}$ and $L^{(2)}$ are the same multisets. Hence $\tau(L^{(1)}) = \tau(L^{(2)})$.

If $L_i > L_j$, we consider the two processes starting with $L^{(1)}$ and $L^{(2)}$, and couple them so that they pick the same index in the first step and carry on independently. Let the chosen index be p, by which we mean the edge with a load equal to L_p has been picked. This leads to two new upper load profiles $L^{(3)}$ for the first process, and $L^{(4)}$ for the second process. Also, define another upper load profile L' which is equal to L, except for L'_p which is equal to $L_p + 1$. Then, L', $L^{(3)}$ and $L^{(4)}$ satisfy the conditions of the claim, with the same indices i and j. Therefore, by induction, we have $\tau(L^{(4)}) \geq \tau(L^{(3)})$. Hence, whatever happens in this step, the second process has a greater chance of exceeding the threshold. This completes the step of the induction and concludes the proof. □

With the help of Claim 5.1.14, we can prove Claim 5.1.12.

Proof of Claim 5.1.12. Take an adversary and some edge $(u, v) \in M_1^*$. If (u, v) is picked with probability at least $(1 - \varepsilon)$ then the claim holds. Therefore, we can assume that (u, v) is picked with probability smaller than $(1 - \varepsilon)$. As a result, due to Claim 5.1.13, we can assume that the adversary lets the total weight be larger than $W = \left\lfloor \varepsilon^{-4} \right\rfloor \cdot k \cdot \lceil kb \rceil$ for all but $s = 2\frac{\log(1/\varepsilon)}{\varepsilon^6}$ steps where $r(u) \geq \varepsilon k$ and $r(v) \geq \varepsilon \lceil kb \rceil$. That is, the adversary satisfies the conditions of Claim 5.1.14. Now, we bound the probability that the maximum load after these steps is larger than $T = \varepsilon^3 \lceil kb \rceil$. Since we have already established that the greedy adversary is optimal, it suffices to bound the probability for the greedy adversary.

By the characterization of Claim 5.1.14, a fixed set $\left\lfloor \varepsilon^{-4} \right\rfloor$ edges of I' have positive weight in all the early steps. This reduces the problem of bounding the maximum load to an instance of the balls into bins problem. The probability that a fixed edge $e \in I'$ is picked in a fixed set of T early steps, is at most $\left(\frac{k \cdot \lceil kb \rceil}{W}\right)^T = \left(\frac{1}{\lfloor \varepsilon^{-4} \rfloor}\right)^T$. Because the weight of each edge is exactly $k \cdot \lceil kb \rceil$ and the total weight is at least W. Therefore, by taking the union bound over all the possible edges and sets of early steps (there are at most $k + \lceil kb \rceil \leq 2\lceil kb \rceil$ steps), the probability that any edge is picked in more than T early steps, is at most:

$$\begin{aligned}
\binom{2\lceil kb \rceil}{T} \left\lfloor \varepsilon^{-4} \right\rfloor \left(\frac{1}{\lfloor \varepsilon^{-4} \rfloor}\right)^T &\leq \left(\frac{2e\lceil kb \rceil}{T}\right)^T \left\lfloor \varepsilon^{-4} \right\rfloor \left(\frac{1}{\lfloor \varepsilon^{-4} \rfloor}\right)^T \\
&\leq \left(\frac{2e\lceil kb \rceil}{T}\right)^T 2\varepsilon^{-4} \left(\frac{2}{\varepsilon^{-4}}\right)^T
\end{aligned}$$

$$\begin{aligned}
&= \left(\frac{4e\lceil kb\rceil}{T\varepsilon^{-4}}\right)^{T} 2\varepsilon^{-4} \\
&\leq (4e\varepsilon)^{\varepsilon^3\lceil kb\rceil} 2\varepsilon^{-4} \\
&\leq \varepsilon.
\end{aligned}$$

Where the first inequality holds since $\binom{n}{k} \leq \left(\frac{en}{k}\right)^k$, and the last inequality holds for small enough ε since $k \geq \varepsilon^{-8}$.

Now, assuming (u, v) was not picked during the process, we construct F. Take the endpoint z of (u, v) that first violates $r(z) \geq \varepsilon b(z)$. Before this inequality is violated, there must be at least $(1-\varepsilon)b(z)$ steps where an edge adjacent to z is picked. At most $s = \frac{\log(1/\varepsilon)}{\varepsilon^6} \leq \varepsilon k \leq \varepsilon b(z)$ of these steps have total weight smaller than W, meaning that the rest of them are early steps. Therefore, if we just let F equal to the edges adjacent to z that are picked in the early steps, it holds that $|F| \geq (1-2\varepsilon)b(z)$. Also, as proved in the last paragraph, with probability $(1-\varepsilon)$ no edge appears in F more than $\varepsilon^3\lceil kb\rceil$ times. This concludes the proof. □

Now, we can define the fractional matching x on $M \cup B$. Recall that we have fixed a maximum matching M^* of G, and let $M_1^* = M^* \cap (V(M) \times \overline{V(M)})$. For every edge $e \in B \setminus M_1^*$, let t_e be $\min(\varepsilon^3\lceil kb\rceil, B_e)$, where B_e is the number of occurrences of e in B. For every edge $(u, v) \in B \cap M_1^*$ with $u \in V(M)$ and $v \in \overline{V(M)}$, we define:

$$t_e = \min\left(k - \sum_{e' \in \delta_B(u) \setminus M_1^*} t_{e'}, \lceil kb\rceil - \sum_{e' \in \delta_B(v) \setminus M_1^*} t_{e'}\right).$$

As a result, for every edge e in $B \setminus M_1^*$ we have $t_e > \varepsilon^3\lceil kb\rceil$. Finally, for $e \in B$, we define

$$x_e = \frac{t_e}{\lceil kb\rceil},$$

and for $e \in M$, we define $x_e = 1 - \frac{1}{b}$.

Claim 5.1.16. *For every edge $(u, v) \in M_1^*$ with $u \in V(M)$ and $v \in \overline{V(M)}$, with probability $(1-\varepsilon)$ it holds that $t(u) = k$ or $t(v) = \lceil kb\rceil$.*

Proof. The claim follows from Claim 5.1.12. When (u, v) is picked in B, the claim holds by the definition of $t_{(u,v)}$. When (u, v) is not picked in B, the claim follows from the existence of multiset $F \subseteq B$ as stated in Claim 5.1.12. □

Claim 5.1.17. *It holds that $\mathbf{E}[x(B)] \geq (1-4\varepsilon)\frac{1}{b+1}|M_1^*|$.*

Proof. The charging argument that we use is quite similar to that of Claim 5.1.4. We repeat the proof in its entirety for the sake of completeness. Order the edges of B arbitrarily as $e_1, \ldots, e_N$, and let B_i be the set of first i edges. With respect to B_i, we define a potential ϕ_i on every edge $(u, v) \in M_1^*$ with $u \in V(M)$ and $v \in \overline{V(M)}$:

$$\phi_i(u, v) = \max\left(\frac{\sum_{e \in \delta_{B_i}(u)} t_e}{k}, \frac{\sum_{e \in \delta_{B_i}(v)} t_e}{\lceil kb\rceil}\right),$$

which is equal to the maximum fraction of the used capacity on its endpoints. We also let:

$$\phi_i = \sum_{(u,v)\in M_1^*} \phi_i(u,v).$$

For an edge e_i, we charge it $c_i = \phi_i - \phi_{i-1}$. For each edge e_i, it holds that $c_i \leq t_{e_i} \cdot \left(\frac{1}{k} + \frac{1}{\lceil kb \rceil}\right)$. Because it is adjacent to at most two edges of M_1^*, and it can increase the potential on either one by at most $\frac{t_{e_i}}{k}$ and $\frac{t_{e_i}}{\lceil kb \rceil}$ respectively. Therefore, we have:

$$\phi_N = \sum_{i=1}^{N} c_i \leq \left(\frac{1}{k} + \frac{1}{\lceil kb \rceil}\right) \sum_{i=1}^{N} t_{e_i}. \tag{5.3}$$

We call an edge (u,v) of M_1^* with $u \in V(M)$ and $v \in \overline{V(M)}$, *saturated* if it satisfies $t(u) \geq (1-2\varepsilon)k$ or $t(v) \geq (1-2\varepsilon)\lceil kb \rceil$, i.e. if it satisfies $\phi_N(u,v) \geq (1-2\varepsilon)$. If we let X be the number of saturated edges, then by definition it holds:

$$\phi_N \geq (1-2\varepsilon)X. \tag{5.4}$$

Putting (5.3) and (5.4) together we get:

$$(1-2\varepsilon)X \leq \left(\frac{1}{k} + \frac{1}{\lceil kb \rceil}\right) \sum_{i=1}^{N} t_{e_i},$$

or equivalently:

$$\sum_{i=1}^{N} t_{e_i} \geq (1-2\varepsilon)\frac{k \cdot \lceil kb \rceil}{k + \lceil kb \rceil} X = (1-2\varepsilon)\frac{k+kb}{k+\lceil kb \rceil} \cdot \frac{\lceil kb \rceil}{b+1} X \geq (1-3\varepsilon)\frac{\lceil kb \rceil}{b+1} X.$$

Due to Claim 5.1.16, each edge in M_1^* is saturated with probability at least $(1-\varepsilon)$. Therefore we have $\mathbf{E}[X] \geq (1-\varepsilon)|M_1^*|$. Given the fact that $x(e_i) = \frac{t_{e_i}}{\lceil kb \rceil}$, it follows:

$$\mathbf{E}[x(B)] = \frac{1}{\lceil kb \rceil} \mathbf{E}\left[\sum_{i=1}^{N} t_{e_i}\right] \geq (1-3\varepsilon)\frac{1}{b+1}\mathbf{E}[X] \geq (1-4\varepsilon)\frac{1}{b+1}|M_1^*|. \qquad \square$$

Claim 5.1.18. *It holds that* $x(M) \geq \left(1 - \frac{1}{b}\right)\left(|M_2^*| + \frac{1}{2}|M_1^*|\right)$.

Proof. The proof is identical to the proof of Claim 5.1.3 and thus we omit it. $\square$

Claim 5.1.19. *$M \cup B$ contains an integral matching of size $(1-\varepsilon)^2 \sum_e x_e$.*

Proof. The proof is very similar to that of Claim 5.1.5. The claim follows from Proposition 2.2.5 since the value of the fractional matching is at most $1 - \frac{1}{b}$ on M, $\frac{1}{b}$ on $B \cap M_1^*$, and ε^3 on $B \setminus M_1^*$. $\square$

Proof of Claim 5.1.11. First, we use Claims 5.1.17 and 5.1.18 to show $\mathbf{E}[\sum_e x_e] \geq (1-\varepsilon)(2-\sqrt{2})\mu(G)$. It holds that:

$$\mathbf{E}\left[\sum_e x_e\right] = x(M) + \mathbf{E}[x(B)]$$

$$\geq \left(1-\frac{1}{b}\right)\left(|M_2^*|+\frac{1}{2}|M_1^*|\right)+(1-4\varepsilon)\frac{1}{b+1}|M_1^*| \qquad \text{(Claims 5.1.17 and 5.1.18)}$$
$$\geq (1-4\varepsilon)\left[\left(1-\frac{1}{b}\right)|M_2^*|+\left(\frac{1}{2}-\frac{1}{2b}+\frac{1}{b+1}\right)|M_1^*|\right].$$

Since $b=1+\sqrt{2}$, we have $1-\frac{1}{b}=\frac{1}{2}-\frac{1}{2b}+\frac{1}{b+1}=2-\sqrt{2}$. Therefore,

$$\mathbf{E}\left[\sum_e x_e\right] \geq (1-4\varepsilon)(2-\sqrt{2})(|M_1^*|+|M_2^*|) = (1-4\varepsilon)(2-\sqrt{2})\mu(G). \tag{5.5}$$

To complete the proof, we note that by Claim 5.1.19, $M \cup B$ contains a matching of size $(1-\varepsilon)^2(1-4\varepsilon)(2-\sqrt{2})\mu(G) \geq (1-6\varepsilon)\cdot .585\cdot \mu(G)$. Replacing ε by $\frac{\varepsilon}{6}$ concludes the proof. □

Lifting the Assumption That the Adversary is Oblivious and M is Maximal

Thus far, we have taken M to be a maximal matching that we maintain. For this to be possible, we have relied on the assumption that the adversary is oblivious. Notably, this is the only place where we use this assumption. It is an open problem to maintain a maximal matching against an adaptive adversary within a $\text{poly}(\log n)$ update time. Therefore, to resolve this issue, we lift the assumption by taking M to be an *almost* maximal matching instead, meaning M is a maximal matching if we ignore $\varepsilon\cdot\mu(G)$ vertices. There are existing algorithms that can maintain an almost maximal matching against an adaptive adversary in $\text{poly}(\log n)$ worst-case update time [60, 172].

Proposition 5.1.20. *There exists a data structure that maintains an almost maximal matching in a fully dynamic graph with* $\text{poly}(\log n)$ *worst-case update time against an adaptive adversary.*

It remains to show that the approximation ratio does not suffer, i.e. Claim 5.1.11 holds even when M is an almost maximal matching. Note that we have only relied on the fact that M is maximal when we use $|M_1^*|+|M_2^*|=\mu(G)$ in (5.5). When M is an almost maximal matching, we have $|M_1^*|+|M_2^*|\geq(1-\varepsilon)\mu(G)$ instead. Hence inequality (5.5) still holds with an extra factor of $(1-\varepsilon)$.

Lifting The Assumption That B is Maximal

We have analyzed the algorithm so far assuming that it has access to B, a maximal matching H. However, as stated in Proposition 5.1.9, each edge of B is "missed" with probability ε. As a result, $\mathbf{E}[\mu(G[M\cup B])]$ is going to suffer a factor of $(1-\varepsilon)$. Because if we fix any matching in $M\cup B$, then the algorithm will successfully find each of its edges with probability $(1-\varepsilon)$. Assuming B is a maximal matching we have proved $\mathbf{E}[\mu(G[M\cup B])]\geq(1-O(\varepsilon))(2-\sqrt{2})(|M_1^*|+|M_2^*|)$. Therefore, when the edges are missed with probability ε, inequality (5.5) still holds with an extra factor of $(1-\varepsilon)$. Therefore, Claim 5.1.11 is true.

Implementation and Update Time Analysis

In this subsection, we provide the implementation details and the runtime analysis of Algorithm 19. To facilitate the analysis, we break it down into several smaller parts.

Oracle Access to B

To utilize Proposition 5.1.10 effectively, we require a fast method to obtain all vertices within close proximity to a random vertex v. Let $\Delta(G[M \cup B])$ be the maximum degree of the graph that only includes the edges of $M \cup B$. Thus, $\Delta(G[M \cup B]) = O(k)$. Let $d = O(\varepsilon^{-3} \log k)$ be the number of rounds needed in Proposition 5.1.10 for graph $G[M \cup B]$. The following lemma outlines a formal approach to obtaining these vertices efficiently.

Lemma 5.1.21. *Let $v_1, v_2, \ldots, v_r$ be a set of random vertices in G such that $r = O(\varepsilon^{-2} \log n)$. There exists an algorithm that runs in $\widetilde{O}(rk^d n)$ time and for each vertex v_i, returns all vertices within distance d of v_i in graph $G[M \cup B]$ with high probability.*

Proof. Let $R = \{v_1, v_2, \ldots, v_r\}$ and $v \in R$. We initiate the process from vertex v and execute a breadth-first search (BFS). Note that for vertex v we do not have the list of its incident edges in B and we only maintain the maximal matching explicitly. To retrieve the list of incident edges of vertex v in B, we invoke the oracle defined in Proposition 5.1.9 for each instance of v in $\widetilde{H}$. This enables us to obtain the adjacency list of vertex v in the graph $G[M \cup B]$. This process is repeated as we execute the BFS, utilizing the oracle as described above whenever we require the adjacency list of a vertex. The process concludes when we have reached all vertices at a distance of d. We repeat this process for all vertices of R.

Let π be the permutation that we use in the algorithm over edges of $\widetilde{H}$. Let $l(v, \pi)$ be the number of vertices in a distance of at most d from v in $G[M \cup B]$ and $u_1^v = v, u_2^v, \ldots, u_{l(v,\pi)}^v$ be all vertices that are visited by BFS if we start from vertex v. Thus, $l(v, \pi) = O(k^d)$. Let $T(u, \pi)$ be the time needed by the oracle in Proposition 5.1.9 for vertex u and permutation π. By Proposition 5.1.9, we have $\mathbf{E}_{u,\pi}[T(u, \pi)] = \widetilde{O}(n)$. While the expected total running time would be $\widetilde{O}(l(v, \pi) \cdot n)$ if $u_1^v, u_2^v, \ldots, u_{l(v,\pi)}^v$ were chosen uniformly at random (since $l(v, \pi)$ is constant), it is important to note that these vertices do not follow a random selection. Instead, they are the vertices visited by the BFS algorithm starting from a random vertex v. Let $S(v, \pi) = \sum_{i=1}^{l(v,\pi)} T(u_i^v, \pi)$. We prove that $\mathbf{E}_{R,\pi}[\sum_{v \in R} S(v, \pi)] = \widetilde{O}(rk^d n)$. Let $\widetilde{m}$ be the number of edges in $\widetilde{H}$. We have

$$\begin{aligned}
\mathbf{E}_{R,\pi}\left[\sum_{v \in R} S(v, \pi)\right] &= \sum_{\pi} \sum_{R} \sum_{v \in R} \sum_{i=1}^{l(v,\pi)} \frac{\mathbf{E}[T(u_i^v, \pi)]}{\widetilde{m}! \cdot \binom{n}{r}} \\
&\leq \sum_{\pi} \sum_{v} l(v, \pi) \cdot \binom{n-1}{r-1} \cdot \frac{\mathbf{E}[T(v, \pi)]}{\widetilde{m}! \cdot \binom{n}{r}} \\
&\leq O(k^d) \cdot \sum_{\pi} \sum_{v} \frac{\binom{n-1}{r-1} \cdot \mathbf{E}[T(v, \pi)]}{\widetilde{m}! \cdot \binom{n}{r}} \\
&\leq O(k^d) \cdot \sum_{\pi} \sum_{v} \frac{r \cdot \mathbf{E}[T(v, \pi)]}{\widetilde{m}! \cdot n} \\
&= O(rk^d) \cdot \mathbf{E}_{v,\pi}[T(v, \pi)] \\
&= \widetilde{O}(rk^d n),
\end{aligned}$$

where the last inequality follows by $\mathbf{E}_{v,\pi}[T(v, \pi)] = \widetilde{O}(n)$, which implies that for a random set R, BFS takes $\widetilde{O}(rk^d n)$ time in expectation.

In order to achieve a high probability bound on the time complexity, we sample $\Theta(\log n)$ sets of r random vertices R, along with a permutation π. For each of these $\Theta(\log n)$ samples, we execute the described BFS

instance. The termination condition is met when, in one of the instances, the BFS halts for all vertices in R. Using Markov's inequality, we can deduce that each individual instance terminates within $\widetilde{O}(rk^d n)$ time with a constant probability. Consequently, at least one of these instances terminates within $\widetilde{O}(rk^d n)$ time with high probability. This completes the proof. □

Computing the Maximum Matching in $G[M \cup B]$

Once we have all the vertices within a close distance of vertex v, we can use Proposition 5.1.10 to estimate $\mu(G[M \cup B])$. We design an algorithm that can answer to the query of whether a vertex is matched in $(1-\varepsilon)$-approximate matching of $G[M \cup B]$.

Claim 5.1.22. *Let v be a random vertex in the graph G, and let D_v represent all the vertices in $G[M \cup B]$ that are within a given distance d of v. Suppose that subgraph $G[D_v]$ is given. There exists an algorithm that determines if v is matched in $(1-\varepsilon)$-approximate matching of $G[M \cup B]$, $\mathcal{L}$, that works in $O(k^d)$ time per query such that if we let $\mathcal{L}(v)$ to be the indicator that shows matching status of v, and $\widetilde{\ell} = \frac{1}{2}\sum_{v \in V} \mathcal{L}(v)$, then we have $(1-\varepsilon) \cdot \mu(G[M \cup B]) \leq \mathbf{E}[\widetilde{\ell}] \leq \mu(G[M \cup B])$.*

Proof. First, since $\Delta(G[M \cup B]) = O(k)$, $d = O(\varepsilon^{-3} \log k)$, we have $|D_v| \leq O(k^d)$. We use the algorithm of Proposition 5.1.10. The running time of the algorithm is linear with respect to $|D_v|$. Thus, for a vertex v, the running time is $O(k^d)$. Furthermore, since $\widetilde{\ell}$ represents the size of the matching generated by the algorithm described in Proposition 5.1.10, it holds $(1-\varepsilon) \cdot \mu(G[M \cup B]) \leq \mathbf{E}[\widetilde{\ell}] \leq \mu(G[M \cup B])$. □

Lemma 5.1.23. *Computing $\widetilde{\mu}$ in Algorithm 19 takes $\widetilde{O}(rk^d n)$ time with high probability.*

Proof. Let $v_1, v_2, \ldots, v_r$ be the sampled vertices in Algorithm 19 of Algorithm 19. By Lemma 5.1.21, the total time to obtain vertices within distance d of v_i for all i in $G[M \cup B]$ takes $\widetilde{O}(rk^d n)$ time with high probability. Let D_{v_i} be all vertices in $G[M \cup B]$ that are within distance d of v_i. By Claim 5.1.22, the time needed to determine if v_i is matched is $O(k^d)$, condition on the fact that D_{v_i} is given which finishes the proof. □

Lemma 5.1.24. *Let $\widetilde{\mu}$ be the estimate in Algorithm 19 of Algorithm 19. Then, with high probability, $(2-\sqrt{2}-\varepsilon) \cdot \mu(G) - \varepsilon n \leq \mathbf{E}[\widetilde{\mu}] \leq \mu(G)$.*

Proof. Let $\mathcal{L}$ be the algorithm in Claim 5.1.22, and $v_1, v_2, \ldots, v_r$ be the sampled vertices in Algorithm 19 of Algorithm 19. Moreover, let X_i be the indicator if v_i is matched by $\mathcal{L}$, i.e. $X_i = \mathcal{L}(v_i)$. Let $\widetilde{\ell} = \frac{1}{2}\sum_{v \in V} \mathcal{L}(v)$. For any realization of B which depends on the permutation π over edges of $\widetilde{H}$, by Claim 5.1.22, we have $(1-\varepsilon) \cdot \mu(G[M \cup B]) \leq \mathbf{E}[\widetilde{\ell}] \leq \mu(G[M \cup B])$. Hence,

$$(1-\varepsilon) \cdot \mathbf{E}[\mu(G[M \cup B])] \leq \mathbf{E}[\widetilde{\ell}] \leq \mathbf{E}[\mu(G[M \cup B])]. \tag{5.6}$$

Note that $\mathbf{E}[X_i] = 2\,\mathbf{E}[\widetilde{\ell}]/n$, given that the number of matched vertices is twice the number of matching edges. Define $X = \sum_{i=1}^{r} X_i$. Thus,

$$\mathbf{E}[X] = \frac{2r \cdot \mathbf{E}[\widetilde{\ell}]}{n}. \tag{5.7}$$

Using Chernoff bound on X,

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

Since $\widetilde{\mu} = (1-\varepsilon)\cdot\frac{nX}{2r}$, with probability of $1 - 2/n^4$ we get

$$\begin{aligned}
\widetilde{\mu} &\in \frac{n(\mathbf{E}[X] \pm \sqrt{12\,\mathbf{E}[X]\log n})}{2r} - \frac{\varepsilon n}{2} \\
&\in \left(\frac{n\,\mathbf{E}[X]}{2r} \pm \frac{\sqrt{12n^2\,\mathbf{E}[X]\log n}}{2r}\right) - \frac{\varepsilon n}{2} \\
&\in \left(\mathbf{E}[\widetilde{\ell}] \pm \sqrt{\frac{6n\,\mathbf{E}[\widetilde{\ell}]\log n}{r}}\right) - \frac{\varepsilon n}{2} && \text{(By (5.7))} \\
&\in \left(\mathbf{E}[\widetilde{\ell}] \pm \sqrt{\frac{\varepsilon^2 n\,\mathbf{E}[\widetilde{\ell}]}{4}}\right) - \frac{\varepsilon n}{2} && \text{(Since } r = 24\varepsilon^{-2}\log n\text{)} \\
&\in \mathbf{E}[\widetilde{\ell}] - \frac{\varepsilon n}{2} \pm \frac{\varepsilon n}{2} && \text{(Since } \mathbf{E}[\widetilde{\ell}] \leq n\text{)},
\end{aligned}$$

thus,

$$\mathbf{E}[\widetilde{\ell}] - \varepsilon n \leq \widetilde{\mu} \leq \mathbf{E}[\widetilde{\ell}].$$

Combining with (5.6),

$$(1-\varepsilon)\cdot\mathbf{E}[\mu(G[M \cup B])] - \varepsilon n \leq \widetilde{\mu} \leq \mathbf{E}[\mu(G[M \cup B])].$$

Plugging Claim 5.1.11,

$$(1-\varepsilon)\cdot(2-\sqrt{2}-\varepsilon)\cdot\mu(G) - \varepsilon n \leq \widetilde{\mu} \leq \mu(G),$$

and

$$(2-\sqrt{2}-4\varepsilon)\cdot\mu(G) - \varepsilon n \leq \widetilde{\mu} \leq \mu(G),$$

yeilds the proof using $\varepsilon' = \varepsilon/4$ in the algorithm. □

Proof of Theorem 5.1.6. By Lemma 5.1.23 and Lemma 5.1.24, we have a semi-dynamic algorithm with query time of $\widetilde{O}(rk^d n)$ where $r = O(\varepsilon^{-2}\log n)$, $k = O(\varepsilon^{-8})$, and $d = O(\varepsilon^{-4})$. Also, the worst-case update time of the algorithm is $\text{poly}(\log n)$, which works against an adaptive adversary, and it returns an estimate $\widetilde{\mu}$ of maximum matching of G such that $(2-\sqrt{2}-\varepsilon)\cdot\mu(G) - \varepsilon n \leq \mathbf{E}[\widetilde{\mu}] \leq \mu(G)$. Therefore, the reduction in Proposition 5.1.7 yields the proof. □

5.2 Sublinear Algorithm for TSP

In this section, we prove the following results:

Theorem 5.2.1 (Formally as Theorem 5.2.37)**.** *For any $\varepsilon > 0$, there is a randomized algorithm that w.h.p. $(1/2 - \varepsilon)$-approximates the size of maximum path cover in $\widetilde{O}(n \cdot \mathrm{poly}(1/\varepsilon))$ time.*

Theorem 5.2.2. *For any $\varepsilon > 0$, there is a randomized algorithm that w.h.p. $(1.5 + \varepsilon)$-approximates the cost of $(1, 2)$-TSP in $\widetilde{O}(n \cdot \mathrm{poly}(1/\varepsilon))$ time.*

Theorem 5.2.3. *For any $\varepsilon > 0$, there is a randomized algorithm that w.h.p. $(1 + \varepsilon)(\frac{11}{6} \approx 1.833)$-approximates the cost of graphic TSP in $\widetilde{O}(n \cdot \mathrm{poly}(1/\varepsilon))$ time.*

Theorem 5.2.4. *For any $\varepsilon > 0$, there is a randomized algorithm that w.h.p. $(1 + \varepsilon)(\frac{5}{3} \approx 1.666)$-approximates the cost of graphic TSP in $n^{2-\Omega_\varepsilon(1)}$ time.*

5.2.1 Technical Overview

In this section, we give an overview of our algorithms, especially our sublinear time maximum path cover algorithm of Theorem 5.2.1 which is the key to the other results as well.

Let us start with using matchings to approximate maximum path cover. Consider a graph that has a Hamiltonian path. Here, the optimal maximum path cover has size $n-1$. On the other hand, any maximum matching can have at most $n/2$ edges, which is by a factor 2 smaller than our optimal path cover. On top of this, we only know close to $1/2$ approximations for maximum matching if we restrict the running to be close to linear in n [34, 43], thus can only achieve an approximation close to $1/4$.

Instead of a single matching, Chen, Kannan, and Khanna [68] showed how to estimate the number of edges in a *maximal matching pair* in $\widetilde{O}(n\sqrt{n})$ time, where a matching pair is simply two edge disjoint matchings. It is not hard to see that the number of edges in a maximal matching pair is at least half the number of edges in a maximum path cover. The problem, however, is that a maximal matching pair is not a collection of paths! In particular, the two matchings can form cycles of length as small as four. Therefore, one may only be able to use $3/4$ fraction of the edges of a matching pair in a path cover. This is precisely why the algorithm of [68] only obtains a $\frac{1}{2} \times \frac{3}{4} = \frac{3}{8}$ approximation for path cover, and a $2 - \frac{3}{8} = 1.625$ approximation for $(1, 2)$-TSP.

If we could modify the matching pair algorithm of [68], and avoid cycles by manually excluding edges whose endpoints are the endpoints of a path in the current matching pair, then we could avoid the $3/4$ factor loss discussed above and achieve a $1/2$-approximation. Unfortunately, checking whether the endpoints of an edge are endpoints of a path requires knowledge about whether a series of other edges belong to the solution, which seems hard to implement in sublinear time.

Instead of checking for cycles manually, we introduce the following Algorithm 20 which avoids cycles more naturally. While our final algorithm is a modified variant of Algorithm 20 described below, we start with Algorithm 20 as we believe it provides the right intuition.

Algorithm 20: A new algorithm for path cover.

Initialize $P \leftarrow \emptyset$.

Each vertex v has two *ports* that we denote by v^0 and v^1. Each of these ports throughout the algorithm will be either *free* or *occupied*. Initially, all ports are free.

Iterate over the edges in some ordering π. Upon visiting an edge $e = (u, v)$:

- If v^0 and u^0 are free, add e to P, mark v^0 and u^0 as occupied, and skip to the next edge.
- If v^1 and u^0 are free, add e to P, mark v^1 and u^0 as occupied, and skip to the next edge.
- If v^0 and u^1 are free, add e to P, mark v^0 and u^1 as occupied, and skip to the next edge.

Return P.

Two properties of Algorithm 20 are crucial. First, it prioritizes occupying (u^0, v^0) (compared to (u^1, v^0) or (u^0, v^1)) which in particular implies that any component in P must have a (u^0, v^0) edge. Second, it never occupies (u^1, v^1) with an edge (u, v). While it is easy to see that the output of Algorithm 20 has maximum degree 2, and is thus a collection of paths or cycles, the two properties above actually guarantee that it never includes any cycle. See Figure 5.2. We provide the formal proof of this later in Section 5.2.2. Additionally, we show that the output of Algorithm 20 must be at least half the size of a maximum path cover, as we prove next. Hence, if we manage to estimate the size of the output P of Algorithm 20, then we have proved Theorem 5.2.1.

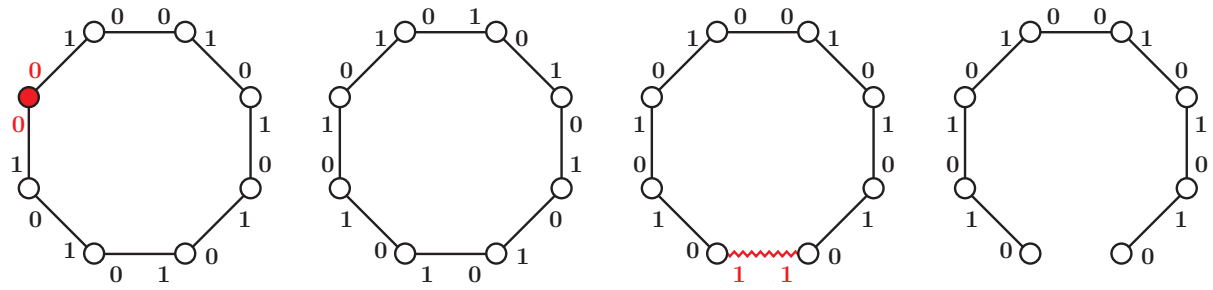


Figure 5.2: Examples of why the output of Algorithm 20 will not have cycles.

Our final algorithm is slightly different from Algorithm 20 discussed above. In particular, we slightly relax it—see Algorithm 21—so that it can be solved via a randomized greedy maximal independent set (RGMIS), for which we have a rich toolkit of sublinear time estimators. Existing approaches (particularly the algorithm of Yoshida, Yamamoto, and Ito [176] and its two-step implementation by Chen, Kannan, and Khanna [68]) can be employed to estimate the value of this modified Algorithm 21 in $\widetilde{O}(n\sqrt{n})$ time. We achieve the improved, and near tight, $\widetilde{O}(n)$ time bound guarantee of Theorem 5.2.1 by building on the analysis of Behnezhad [34] for maximal independent set on the line graphs (i.e., maximal matchings). Though we note that several new ideas are needed, because the MIS graph in our case will not be exactly a line graph. We defer more discussions about this to Sections 5.2.2 and 5.2.3.

Implications for TSP: By having an α-approximate maximum path cover algorithm, we immediately obtain a $(2-\alpha)$-approximation for $(1,2)$-TSP. Therefore, the algorithm above immediately proves Theorem 5.2.2 that we can (almost) 1.5-approximate $(1,2)$-TSP in $\widetilde{O}(n)$ time. For our Theorem 5.2.3 for graphic TSP, we first observe that our improved path cover algorithm can be employed to provide a better lower bound for the optimal TSP solution. This improves the 1.92-approximation of [68] as black-box to 1.9-approximation (Section 5.2.6). However, the final improvement to 1.83 requires more ideas, in particular, on how to better estimate the number of certain *bridges* in the graph. See Section 5.2.7 for more details about this.

5.2.2 New Meta Algorithms for Maximum Path Cover

In this section, we present a new meta algorithm for maximum path cover that obtains a 1/2-approximation. The algorithm, as we will state it in this section, will not be particularly in the sublinear time model. We discuss its sublinear time implementation later in Sections 5.2.3 and 5.2.4.

Our starting point is the Algorithm 20 described in Section 5.2.1. Let us first formally prove that it obtains a 1/2-approximation, and that no component in it is a cycle.

Claim 5.2.5. *The output of Algorithm 20 is a collection of disjoint paths.*

Proof. Since P has maximum degree two, it suffices to show none of its connected components are cycles. Property (i) above implies that at any point during the algorithm, any degree one vertex v has its port v^0 occupied. Now take an edge $e=(u,v)$ that forms a cycle if added to P. Both u and v must have degree one and so u^0 and v^0 are occupied. Since by property (ii) edge e does not occupy both v^1 and u^1, the algorithm does not add e to P thus not completing a cycle. □

Claim 5.2.6. *Let $P^\star$ be* any *path cover using weight one edges. Then the output of Algorithm 20 has size at least $\frac{1}{2}|P^\star|$.*

Proof. For any edge $e=(u,v)\in P^\star$ define $\phi(e)=\frac{1}{4}(\deg_P(u)+\deg_P(v))$. We first claim that for every edge $e=(u,v)$ in G, we have $\phi(e)\geq 1/2$ (or, equivalently, $\deg_P(u)+\deg_P(v)\geq 2$). This is clear for edges $e\in P$ due to the contribution of e itself to its endpoints' degrees, so fix $e\notin P$. Consider the time that we process $e=(u,v)$ in the algorithm and decide not to add it to P. We claim that out of v^0,v^1,u^0,u^1 at least two ports must be occupied. Suppose w.l.o.g. and for contradiction that only v^x is occupied for $x\in\{0,1\}$. Then (u,v) can occupy v^{1-x} and u^x and be added to P. This contradicts (u,v) not being added to P and proves our claim that $\phi(e)\geq 1/2$.

From the discussion above, we get that

$$\sum_{e\in P^\star}\phi(e)\geq\sum_{e\in P^\star}1/2=|P^\star|/2.$$

Moreover, because every vertex has degree at most two in $P^\star$, we get

$$\sum_{e\in P^\star}\phi(e)=\frac{1}{4}\sum_{(u,v)\in P^\star}\deg_P(u)+\deg_P(v)\leq\frac{1}{4}\cdot 2\sum_{v\in V}\deg_P(v)=|P|.$$

The two inequalities above combined imply that $|P|\geq|P^\star|/2$. □

As discussed, our final algorithm is different from Algorithm 20 discussed above. One problem with Algorithm 20 is that it cannot be cast as an instance of the randomized greedy maximal independent set (RGMIS) algorithm for which there is a rich toolkit of sublinear time estimators. To remedy this, we present a modified variant of Algorithm 20 whose output is (almost) as good, but in addition can be modeled as an instance of RGMIS. We denote the output of RGMIS on a graph G with a permutation π on its vertices by $\text{RGMIS}(G, \pi)$.

The algorithm is stated below as Algorithm 21. Similar to the output of Algorithm 20, the output of Algorithm 21 can be verified to have maximum degree two. Thus, it is a collection of paths and cycles. But unlike Algorithm 20, the output of Algorithm 21 can have cycles. This happens since, unlike Algorithm 20, each connected component of the output of Algorithm 21 is not guaranteed to have an edge (u, v) occupying both u^0 and v^0. Nonetheless, we are able to show that this bad event only happens for a small fraction of connected components of the output of Algorithm 21 in expectation, and so once we remove one edge of each of these cycles, the resulting collection of disjoint paths has almost the same size.

Algorithm 21: A modification of Algorithm 20 that uses RGMIS.

Parameter: K (think of it as a large constant integer).

Let $G = (V, E)$ be the subgraph of weight one edges. We construct a graph $H = (V_H, E_H)$ from G on which we run RGMIS.

Each vertex in H corresponds to an edge e in G and two *ports* (as in Algorithm 20) of the endpoints of e that it occupies. Formally, for any $(u, v) \in E$ we have $K + 2$ vertices in H:

- One vertex that corresponds to occuping u^0 and v^1.
- One vertex that corresponds to occuping u^1 and v^0.
- K vertices that each corresponds to occuping u^0 and v^0.

Consider two distinct vertices a and b in H corresponding to edges e_a and e_b in G:

- If $e_a = e_b$ then we add an edge between a and b in H.
- If e_a and e_b share exactly one endpoint v and both a and b occupy the same port of v, we add an edge between a and b in H.

Find a randomized greedy maximal independent set I of H.

Let P be the set of edges in G corresponding to the vertices in I.

Return P.

Observation 5.2.7. *Let C be a connected component in the output of Algorithm 21. If C is a cycle, then every edge in C occupies one 0-port and one 1-port (that is, no edge occupies two 0-ports).*

Proof. Suppose that C has n' vertices. Since each vertex in a cycle has degree two, both ports of each vertex in C must be occupied. Hence, n' 0-ports and n' 1-ports of C are occupied in total. Given that any edge occupies at least one 0-port by the algorithm, we cannot have an edge that occupies two 0-ports, or else we should occupy more 0-ports than 1-ports of C, which is a contradiction. □

Next, we show that up to a factor of $(1 + 2/k)$ which is negligible for K in the order $1/\varepsilon$, the output of Algorithm 21 is an (almost) 1/2-approximation of the maximum path cover value.

Observation 5.2.8. *Let C be a connected component in the output of Algorithm 21. If C is a path, then it contains at most one edge that occupies two 0-ports.*

Proof. Let C be the path $(v_1, v_2, \ldots, v_r)$. Since the degree of any vertex v_i for $1 < i < r$ is two in the path, both ports of v_i must be occupied. For v_1 and v_r, on the other hand, only one port is occupied. Hence, the total number of 0-ports that are occupied by C minus the number of 1-ports occupied by it is at most two. This means that there is at most one edge that occupies two 0-ports since all other types of edges occupy exactly one 0-port and one 1-port. □

Lemma 5.2.9. *let P be the output of Algorithm 21 on graph G. Then*

$$\frac{1}{2}\rho(G) \leq \mathbf{E}\,|P| \leq \left(1 + \frac{2}{K}\right)\rho(G),$$

where the expectation is taken over the randomization of computing RGMIS in Algorithm 21.

Proof. Let P^* be a maximum path cover. For any edge $e = (u, v) \in P^\star$ define $\phi(e) = \frac{1}{4}(\deg_P(u) + \deg_P(v))$. With the exact same argument as in the proof of Claim 5.2.6, we get that $\phi(e) \geq 1/2$, which implies

$$\sum_{e \in P^\star} \phi(e) \geq \sum_{e \in P^\star} 1/2 = \rho(G)/2.$$

Since the degree of each vertex in P is at most two, we get

$$\sum_{e \in P^\star} \phi(e) = \frac{1}{4} \sum_{(u,v) \in P^\star} \deg_P(u) + \deg_P(v) \leq \frac{1}{4} \cdot 2 \sum_{v \in V} \deg_P(v) = |P|.$$

By combining above inequalities we get $\frac{1}{2}\rho(G) \leq |P|$. Note that we do not need the randomization for the proof of the lower bound.

By construction of P, every vertex has degree at most two in P. Hence, all connected components of P are cycles and paths. We claim that at most $\frac{2}{K+2}$ fraction of connected components are cycles in expectation. Since the expected number of connected components is at most $\mathbf{E}\,|P|$, from this we get that the expected number of cycles is at most $2\,\mathbf{E}\,|P|/(K+2)$. By removing one edge from each cycle, we obtain a valid solution for maximum path cover problem. Thus,

$$\mathbf{E}\,|P| - \frac{2\,\mathbf{E}\,|P|}{K+2} = \frac{K}{K+2}\,\mathbf{E}\,|P| \leq \rho(G) \qquad \Rightarrow \qquad \mathbf{E}\,|P| \leq \left(1 + \frac{2}{K}\right) \cdot \rho(G).$$

So it remains to show that at most $\frac{2}{K+2}$ fraction of connected components are cycles in expectation. As we process edges one by one according to the ordering of RGMIS, let A be the set of edges that none of their incident edges are added to the solution of Algorithm 21. By definition of A, if one copy of edge (u, v) is in A, then all other copies of (u, v) are also in A. Therefore, at any point during running RGMIS, if a new component is added to the solution, the edge (u, v) that gets added to the solution occupies (u^0, v^0) with probability at least $\frac{K}{K+2}$ since K copies out of the $K + 2$ copies are for (u^0, v^0). Let C_0 be the number of times that the newly added component is an edge occupying two 0-ports, and C_1 be the number of times that the newly added component is an edge occupying one 0-port and one 1-port. By the above argument, we have

$$\frac{\mathbf{E}[C_0]}{\mathbf{E}[C_0] + \mathbf{E}[C_1]} = \frac{K}{K+2}. \tag{5.8}$$

Note that after running Algorithm 21, it is possible that the number of connected components is actually smaller than $C_0 + C_1$, since some of the components may merge as the algorithm proceeds. However, by Observation 5.2.8, two components that their first edge occupies two 0-ports will not merge together. Also, by Observation 5.2.7, none of the cycle components have an edge that occupies two 0-ports. Therefore, in the end, there exists at most $\mathbf{E}[C_0] + \mathbf{E}[C_1]$ connected components and at least $\mathbf{E}[C_0]$ of them will not be cycles. This completes the proof. □

5.2.3 A Local Query Process for the Algorithm and its Complexity

In this section, we define a query process to estimate the size of the output of Algorithm 21.

In graph H of Algorithm 21, each vertex corresponds to an edge in the original graph. More precisely, we make $K+2$ copies of each edge (u, v) such that one of the copies corresponds to an edge occupying (u^0, v^1), one for (u^1, v^0), and K for (u^0, v^0). We use $G' = (V, E')$ to show the new graph with these parallel edges. During the course of Algorithm 21, two different edges that share the same endpoint and port cannot appear in the solution together. We use the following definition to formalize this notion.

Definition 5.2.10 (Conflicting Pair of Edges)**.** *Two edges $e, e' \in E'$ that share an endpoint v are* conflicting *if both e and e' correspond to same port v^i for $i \in \{0, 1\}$. We call (e, e') a conflicting pair of edges.*

In order to estimate the size of the output of Algorithm 21, we define a vertex oracle that given a vertex v and a permutation π on E', returns the degree of vertex v in the output of Algorithm 21. These are akin to the query processes used before in the works of [34, 176], but are specific to our Algorithm 21.

Algorithm 22: "vertex oracle" $\text{VO}(u, \pi)$ to determine the degree of vertex u in $\text{RGMIS}(G', \pi)$.

1 Let $e_1 = (u, v_1), \ldots, e_r = (u, v_r)$ be the edges incident to u with $\pi(e_1) < \ldots < \pi(e_r)$.
2 $d \leftarrow 0$
3 **for** i *in* $1 \ldots r$ **do**
4 **if** $\text{EO}(e_i, v_i, \pi) = \text{TRUE}$ **then** $d \leftarrow d + 1$;
5 **return** d

Algorithm 23: "edge oracle" $\text{EO}(e, u, \pi)$ to determine an edge e is in $\text{RGMIS}(G', \pi)$. Also, u must be an endpoint of e.

1 **if** $\text{EO}(e, u, \pi)$ *computed before* **then return** the computed result.;
2 Let $e_1 = (u, v_1), \ldots, e_r = (u, v_r)$ be the edges incident to e such that $\pi(e_1) < \ldots < \pi(e_r) < \pi(e)$. Also, (e, e_i) is a conflicting pair for all $1 \leq i \leq r$.
3 **for** i *in* $1 \ldots r$ **do**
4 **if** $\text{EO}(e_i, v_i, \pi) = \text{TRUE}$ **then return** FALSE;
5 **return** TRUE

Note that in Line 2 of the Algorithm 23 we only recursively call the function on edges that their label, conflict with edge e since if other edges appear in the RMGIS subgraph, we can still have e in the RGMIS subgraph. Before analyzing the query complexity of the vertex oracle, we prove the correctness of the vertex oracle.

Claim 5.2.11. *For any edge $e = (u, z) \in E'$ that is occupying ports u^i and z^j, if $\text{EO}(e, u, \pi)$ is called while computing $\text{VO}(v, \pi)$, then $\text{EO}(e, u, \pi) = \text{TRUE}$ iff $e \in \text{RGMIS}(G', \pi)$.*

Proof. We prove the claim using induction on ranking of edge e. Assume that the claim is true for all edges with ranking smaller than $\pi(e)$. If $\mathrm{EO}(e, u, \pi)$ is called by $\mathrm{EO}(e' = (w, z), z, \pi)$ or directly by $\mathrm{VO}(v, \pi)$, then by definition of Algorithm 23 and Algorithm 22, all edges $e'' = (w', z)$ with $\pi(e'') < \pi(e')$ that are occupying z^j are queried before e' which means that none of them return TRUE. Hence, by induction hypothesis, none of the edges incident to z that are occupying z^j with lower rank are in the $\mathrm{RGMIS}(G', \pi)$. Moreover, $\mathrm{EO}(e, u, \pi)$ calls all incident edges to u with lower rank that are occupying u^i and return TRUE if none of them are in the $\mathrm{RGMIS}(G', \pi)$ by induction hypothesis. Therefore, $\mathrm{EO}(e, u, \pi) = \textsc{True}$ iff $e \in \mathrm{RGMIS}(G', \pi)$. □

Claim 5.2.12. *Let $v \in V$ and d be the output of $\mathrm{VO}(v, \pi)$. Then d is equal to the degree of vertex v in the subgraph outputted by $\mathrm{RGMIS}(G', \pi)$.*

Proof. The observation follows by combining the fact that the vertex oracle queries edges in increasing order and Claim 5.2.11. □

Let $T(v, \pi)$ denote the number of recursive calls to the edge oracle during the execution of $\mathrm{VO}(v, \pi)$.

Theorem 5.2.13. *For a randomly chosen vertex v and permutation π on E', we have that*

$$\mathbf{E}_{v,\pi}[T(v, \pi)] = O(\bar{d} \cdot \log^2 n)$$

where $\bar{d}$ is the average degree of the graph G.

Let $Q(e, v, \pi)$ be the number of $\mathrm{EO}(e, \cdot, \pi)$ calls during the execution of $\mathrm{VO}(v, \pi)$. Moreover, let $Q(e, \pi)$ be the number of $\mathrm{EO}(e, \cdot, \pi)$ calls starting from any vertex. In other words, we have that $Q(e, \pi) = \sum_{v \in V} Q(e, v, \pi)$.

Observation 5.2.14. *For every edge e and permutation π, $Q(e, \pi) \leq O(n^2)$.*

Proof. Let $e = \{x, y\}$. For a fixed vertex u, either the vertex oracle $\mathrm{VO}(u, \pi)$ queries the edge oracle for e directly, or through some incident edge e'. Hence, the edge oracle of e is called through at most $(K + 2)(\deg(x) - 1) + (K + 2)(\deg(y) - 1)$ of its incident edges ($K + 2$ appears since each edge has $K + 2$ copies), which implies that $Q(e, u, \pi) \leq (2K + 4)(n - 1) + 1$. Therefore,

$$Q(e, \pi) \leq \sum_{u \in V} Q(e, u, \pi) \leq n\left((2K + 4)(n - 1) + 1\right) \leq O(n^2)$$ □

The main contribution of this section is to show that the expected number of $\mathrm{EO}(e, \pi)$ calls over all permutations π is $O(\log^2 n)$, which is formalized in the following lemma.

Lemma 5.2.15. *For any edge $e \in E'$, we have $\mathbf{E}_\pi[Q(e, \cdot, \pi)] = O(\log^2 n)$.*

Assuming the correctness of Lemma 5.2.15, we can complete the proof of Theorem 5.2.13.

Proof of Theorem 5.2.13.

$$\begin{aligned}
\mathbf{E}_{v,\pi}[T(v, \pi)] = \frac{1}{n}\,\mathbf{E}_\pi\Big[\sum_{v \in V} T(v, \pi)\Big] &= \frac{1}{n}\,\mathbf{E}_\pi\Big[\sum_{v \in V}\sum_{e \in E'} Q(e, v, \pi)\Big] \\
&= \frac{1}{n}\,\mathbf{E}_\pi\Big[\sum_{e \in E'}\sum_{v \in V} Q(e, v, \pi)\Big] = \frac{1}{n}\,\mathbf{E}_\pi\Big[\sum_{e \in E'} Q(e, \pi)\Big] \\
&= \frac{1}{n}\sum_{e \in E'} \mathbf{E}_\pi[Q(e, \pi)] = \frac{1}{n}\sum_{e \in E'} O(\log^2 n)
\end{aligned}$$

$$= \frac{1}{n} O(|E'| \cdot \log^2 n) = O(\bar{d} \cdot \log^2 n). \qquad \square$$

During the recursive calls to the edge oracle that starts from vertex v, the edges in the stack of recursive calls create a trail.

Observation 5.2.16. *Let $S = (e_1 = (v, u), e_2, \ldots, e_r)$ be the stack of recursive calls starting from vertex v. Then $(e_1, e_2, \ldots, e_r)$ is a trail in G'.*

Proof. Since in Line 2 of Algorithm 23, edge oracle only queries incident edges, $(e_1, e_2, \ldots, e_r)$ is a walk. It remains to show that all edges are distinct. Suppose that $e_i = e_j$ for some $i < j$ which implies $\pi(e_i) = \pi(e_j)$. Since the edge oracle queries edges in decreasing order, we have $\pi(e_j) < \pi(e_i)$ which is a contradiction. $\square$

We direct the edges of the trail from v to the other endpoint. We call a trail that starts from v on the graph with edge permutation π, a (v, π)-query-trail. For an edge $e = (x, y)$, let $\vec{e}$ denote the directed edge from x to y and $\overleftarrow{e}$ denote a directed edge from y to x.

Observation 5.2.17. *Let $\vec{P} = (\vec{e_1}, \vec{e_2}, \ldots, \vec{e_k})$ be a (v, π)-query-trail; then $\pi(e_1) > \pi(e_2) > \ldots > \pi(e_k)$.*

Proof. During the answering whether an edge is in RGMIS(G', π), Algorithm 23 recursively calls on edges with π values lower than the value of the current edge. Therefore, the stack of recursive calls will be decreasing with respect to π values. $\square$

Let $Q(\vec{e}, \pi) \subseteq Q(e, \pi)$ be the set of all query trails that end at $\vec{e}$ (with the same direction). In what follows, we obtain a bound for the query complexity for $\vec{e}$. We use this lemma to prove Lemma 5.2.15.

Lemma 5.2.18. *For any edge e, we have $\mathbf{E}_\pi[Q(\vec{e}, \pi)] = O(\log^2 n)$.*

Proof of Lemma 5.2.15. Since $Q(e, \pi) = Q(\vec{e}, \pi) \cup Q(\overleftarrow{e}, \pi)$, by Lemma 5.2.18 we have

$$\mathbf{E}_\pi[Q(e, \pi)] \leq \mathbf{E}_\pi[Q(\vec{e}, \pi)] + \mathbf{E}_\pi[Q(\overleftarrow{e}, \pi)] = O(\log^2 n) + O(\log^2 n) = O(\log^2 n). \qquad \square$$

Given a permutation π and a trail $\vec{P} = (\vec{e_1}, \vec{e_2}, \ldots, \vec{e_k})$, we define $\phi(\pi, \vec{P})$ to be another permutation σ over the edges such that:

$$(\sigma(e_1), \sigma(e_2), \ldots, \sigma(e_{k-1}), \sigma(e_k)) \coloneqq (\pi(e_2), \pi(e_3), \ldots, \pi(e_k), \pi(e_1))$$
$$\pi(e') = \sigma(e') \qquad \forall e' \notin \vec{P}$$

Given an edge $\vec{e}$, by using the above mapping function we can construct a bipartite graph H with two parts A and B such that each part has $|E'|!$ vertices showing different permutations of edges. For a permutation $\pi \in A$ and a (v, π)-query-trail $\vec{P}$ that ends at $\vec{e}$ for some arbitrary vertex v, we connect π in A to $\phi(\pi, \vec{P})$ in B. Note that by construction of H, $\deg(\pi_A) = Q(\vec{e}, \pi_A)$ for all $\pi_A \in A$, since we have a unique edge for each query-trail that ends at $\vec{e}$ with permutation π_A. Hence, in order to prove Lemma 5.2.15, it is sufficient to prove that $\mathbf{E}_{\pi_A \sim A}[\deg_H(\pi_A)] = O(\log^2 n)$. Let $\mathcal{Q}(\vec{e}, \pi)$ be the set of all query-trails for permutation π that ends at $\vec{e}$. Let $\beta = c \log^2 n$ for some large c. We partition permutations into two sets of *likely* and *unlikely* permutations called L and U as follows:

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

Likely permutations are those permutations that the longest query-trail ending at $\vec{e}$ has length at most β and unlikely permutations are the remaining permutations. Let A_L be the set of vertices corresponding to the likely permutations in A and A_U be the set of vertices corresponding to the unlikely permutations. The intuition behind this partitioning is that the set of unlikely permutations makes up a tiny fraction of all permutations which is formalized in Lemma 5.2.19.

Lemma 5.2.19. *If c is a large enough constant, then we have $|A_U| \leq |E'|!/n^2$.*

Before proving Lemma 5.2.19, we introduce the parallel implementation of the greedy maximal independent set.

Parallel Randomized Greedy Maximal Independent Set: Let G be a graph and π be a permutation over its edges. In each iteration, we pick all vertices whose rank is less than all their neighbors and remove all their neighbors. We denote the number of rounds in this algorithm until G becomes empty as *round complexity* and we show it with $\rho(G, \pi)$.

It is clear that the output of the parallel randomized greedy MIS is the same as $\text{RGMIS}(G, \pi)$. We have the following known result about the round complexity of parallel randomized greedy MIS.

Lemma 5.2.20 ([61, Theorem 3.5])**.** *For a uniformly random chosen permutation π over edges of G, we have $\rho(G, \pi) = O(\log^2 n)$, with probability of at least $1 - \frac{1}{n^2}$.*

In order to use the above lemma, we need to show that for an unlikely permutation, the round complexity is large and therefore, small fraction of permutations are unlikely as a result of Lemma 5.2.20.

Claim 5.2.21. *Let $\vec{P}$ be query-trail in G' with permutation π. Then $\rho(G', \pi) \geq \lfloor \frac{|\vec{P}|}{2} \rfloor$.*

Proof. Let $\vec{P} = (\vec{e_1}, \vec{e_2}, \ldots, \vec{e_k})$ be a query-trail. By Observation 5.2.17, we have $\pi(e_1) > \pi(e_2) > \ldots > \pi(e_k)$, where e_k is the last edge on the trail. Let $\rho(e)$ show the round in which edge e is deleted by the parallel algorithm. If we can show that for $i < k-1$, $\rho(e_i) > \rho(e_{i+2})$, then we have that $\rho(e_2) \geq \lfloor \frac{k}{2} \rfloor$ which completes the proof. We prove it using a contradiction. Assume that $\rho(e_i) \leq \rho(e_{i+2})$ for some $1 < i < k-1$. Note that $\rho(e_{i+1}) \geq \rho(e_i)$, otherwise, when e_{i+1} is deleted from the graph, one of its corresponding ports that is shared with e_i and e_{i+2} was occupied which implies that at least one of e_i and e_{i+2} should be deleted at the same time. Hence, in round $\rho(e_i)$, edge e_{i+1} is still present in the graph. Therefore, e_i is not a local minimum in round $\rho(e_i)$ and is deleted due to presence of an edge e' in the solution. Note that $e' \neq e_{i+1}$ since e_{i+1} is not the minimum edge because e_{i+2} is still in the graph. If e' is only incident to e_i, $\text{EO}(e_{i-1}, \cdot, \pi)$ should call $\text{EO}(e', \cdot, \pi)$ before $\text{EO}(e_i, \cdot, \pi)$ since e' is the local minimum in round $\rho(e_i)$ and therefore $\pi(e') < \pi(e_i)$. If e' is incident to both e_i and e_{i+1}, $\text{EO}(e_i, \cdot, \pi)$ should call $\text{EO}(e', \cdot, \pi)$ before $\text{EO}(e_{i+1}, \cdot, \pi)$ since e' is local minimum at round $\rho(e_i)$ and therefore $\pi(e') < \pi(e_{i+1})$. In both cases, the edge oracle terminates and will not query edge e_{i+2}. Hence, the assumption that $\rho(e_i) \leq \rho(e_{i+2})$ leads to a contradiction and the proof is complete. □

Now we are ready to prove Lemma 5.2.19.

Proof of Lemma 5.2.19. For each unlikely permutation $\pi \in U$, there exists a query-trail of length larger than β. By Claim 5.2.21, we have $\rho(G, \pi) \geq \lfloor \frac{\beta+1}{2} \rfloor$. Since $\beta = c \log^2 n$, by choosing c large enough and Lemma 5.2.20, we have that $|U|/|\Pi| \leq 1/n^2$. Therefore, $|U| \leq |E'|!/n^2$ which implies that $|A_U| \leq |E'|!/n^2$ since A_U represents vertices that correspond to unlikely permutations. □

Next, we show that each vertex $\pi_B \in B$, has at most β neighbors between likely permutations in part A in bipartite graph H.

Lemma 5.2.22. *Let π_Y be a vertex in Y. Then π_Y has most β neighbors in X_L.*

Before proving this lemma, we show how we can prove Lemma 5.2.18 using Lemma 5.2.19 and Lemma 5.2.22.

Proof of Lemma 5.2.18. Note that by Observation 5.2.14, degree of each vertex $\pi_A \in A$ is at most $O(n^2)$. Combining Lemma 5.2.19, we have

$$E(A_U, B) \leq |E'|!/n^2 \cdot O(n^2) \leq O\big(|E'|!\big).$$

Moreover, by Lemma 5.2.22, each vertex $\pi_B \in B$ has at most $O(\beta)$ neighbors in A_L. Since H is a bipartite graph, $E(A_L, B) \leq O(\beta) \cdot |A_L|$. Therefore, sum of degrees of all vertices in A is at most

$$E(A_L, B) + E(A_U, B) \leq O(\beta) \cdot |A_L| + O(|E'|!) \leq O(\beta \cdot |E'|!).$$

For a random vertex in A, the expected degree is $\frac{O(\beta \cdot |E'|!)}{|E'|!} = O(|E'|)$. Combining with $\beta = c \log^2 n$ and $\deg(\pi_A) = Q(\vec{e}, \pi_A)$ completes the proof. □

The rest of this section, we prove Lemma 5.2.22. Before proving Lemma 5.2.22, we prove that if two different query-trails that are mapped to two different permutations of A_L to $\pi_B \in B$ by ϕ, the shorter query-trail must be subgraph of the longer one.

Lemma 5.2.23. *Let π and π' be two different permutations, and $\vec{P}$ and $\vec{P}'$ be (v, π)- and (v', π')-query-trail, respectively, that both end at edge $\vec{e}$. If $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P}')$ and $|\vec{P}| \geq |\vec{P}'|$, then $\vec{P}'$ is a subgraph of $\vec{P}$.*

We prove this lemma by series of observations and claims. Let $\vec{P} = (\vec{e_k}, \dots, \vec{e_1})$ and $\vec{P}' = (\vec{e_r}', \dots, \vec{e_1}')$ such that $e = e_1 = e'_1$. If $\vec{P}'$ is not a subgraph of $\vec{P}$, then it must branch after an edge $\vec{e_b}'$. This means that $\vec{e_i} = \vec{e_i}'$ for $i \leq b$ and $\vec{e_{b+1}} \neq \vec{e_{b+1}}'$. Note that $\vec{e_{b+1}}$ and $\vec{e_{b+1}}'$ can be copy of the same edge.

Observation 5.2.24. *Let π be a random permutation over E'. For a (u, π)-query-trail, if f and f' are two consecutive edges in the trail, then (f, f') is a conflicting pair.*

Proof. Since the edge oracle calls $\mathrm{EO}(f', \cdot, \pi)$ in $\mathrm{EO}(f, \cdot, \pi)$, (f, f') must be a conflicting pair. □

Observation 5.2.25. *Let f_1, f_2, f_3 be three different edges incident to some vertex u and let π be a random permutation over E'. Let $\vec{P_1}$ be a (x, π)-query-trail that calls $\mathrm{EO}(f_3, \cdot, \pi)$ in $\mathrm{EO}(f_1, \cdot, \pi)$. Also, let $\vec{P_2}$ be a (y, π')-query-trail that calls $\mathrm{EO}(f_3, \cdot, \pi')$ in $\mathrm{EO}(f_2, \cdot, \pi')$. Then (f_1, f_2) is a conflicting pair.*

Proof. By Observation 5.2.24, (f_1, f_3) is a conflicting pair. Assume that both f_1 and f_3 occupied port u^i. Moreover, since (f_2, f_3) is a conflicting pair, then f_2 is also occupying u^i. Therefore, (f_1, f_2) is a conflicting pair. □

Observation 5.2.26. $\pi(e_b) = \pi'(e_{b+1})$.

Proof. Since $\vec{e_{b+1}}$ is not in $\vec{P}'$, we have that $\phi(\pi', \vec{P}')(e_{b+1}) = \pi'(e_{b+1})$. Also, $\phi(\pi, \vec{P})(e_{b+1}) = \pi(e_b)$ since $\phi(\pi, \vec{P})$ shifts edges of the trail $\vec{P}$ by one. Given that permutation $\phi(\pi, \vec{P})$ is equal to $\phi(\pi', \vec{P}')$, we have $\pi(e_b) = \pi'(e_{b+1})$. □

Without loss of generality, we can assume that $\pi(e_b) \leq \pi'(e_b)$ since we did not make any difference between π and π' until this point.

Observation 5.2.27. $\pi'(e_{b+1}) < \pi'(e_b)$.

Proof. By combining Observation 5.2.26, our assumption that $\pi(e_b) \leq \pi'(e_b)$, and the fact that π' is a permutation, we have that $\pi'(e_{b+1}) < \pi'(e_b)$. □

Claim 5.2.28. *If $\pi(f) < \pi(e_b)$ or $\pi'(f) < \pi(e_b)$ for some edge $\vec{f}$, then $\pi(f) = \pi'(f)$.*

Proof. There are five different possible cases for f:

- $\vec{f} \notin \vec{P} \cup \vec{P'}$: Since ϕ only changes the edge on the query-trail and $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P'})$, we have $\pi(f) = \pi'(f)$.
- $\vec{f} \in \{\vec{e_1}, \ldots, \vec{e}_{b-1}\}$: Since $\phi(\pi, \vec{P})(e_{i+1}) = \phi(\pi', \vec{P'})(e_{i+1})$ for $1 \leq i < b$, we have $\pi(e_i) = \pi'(e_i)$. Hence, $\pi(f) = \pi'(f)$.
- $\vec{f} = \vec{e_b}$: In this case, condition $\pi(f) < \pi(e_b)$ does not hold since $\pi(f) = \pi(e_b)$. Also, $\pi'(f) = \pi'(e_b) \geq \pi(e_b)$. Therefore, condition $\pi'(f) < \pi(e_b)$ does not hold.
- $\vec{f} \in \{\vec{e}_{b+1}, \ldots, \vec{e_k}\}$: By Observation 5.2.17, we have that $\pi(f) > \pi(e_b)$. Therefore, condition $\pi(f) < \pi(e_b)$ does not hold. Let $\vec{f} = \vec{e_i}$ for $i > b$. Since $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P'})$, we have $\pi'(f) = \pi(e_{i-1}) \geq \pi(e_b)$. Therefore, none of the conditions in the claim statement holds.
- $\vec{f} \in \{\vec{e_{b+1}}', \ldots, \vec{e_r}'\}$: By Observation 5.2.17, we have that $\pi'(f) > \pi'(e_b)$. Combining by our assumption that $\pi'(e_b) \geq \pi(e_b)$, we have $\pi'(f) \geq \pi(e_b)$. Let $\vec{f} = \vec{e_i}'$ for $i > b$. Since $\phi(\pi, \vec{P}) = \phi(\pi', \vec{P'})$, we have that $\pi(f) = \pi'(e'_{i-1}) \geq \pi'(e_b) \geq \pi(e_b)$. Therefore, none of the conditions in the claim statement holds.

The proof is thus complete. □

Claim 5.2.29. $e_{b+1} \in \text{RGMIS}(G', \pi')$.

Proof. We prove the claim by contradiction. Assume that $e_{b+1} \notin \text{RGMIS}(G', \pi')$. Hence, there exists an edge f which is incident to e_{b+1} such that $\pi'(f) < \pi'(e_{b+1})$. Thus, $\text{EO}(e_{b+1}, \cdot, \pi')$ will recursively call $\text{EO}(f, \cdot, \pi')$. Let f be incident to e_i and e_{i+1} for $i \in \{b, b+1\}$. In the query-trail $\vec{P}$, $\text{EO}(e_{i+1}, \cdot, \pi)$ calls $\text{EO}(e_i, \cdot, \pi)$. Therefore, using the Observation 5.2.25, we have that (f, e_i) is a conflicting pair. Note that by Observation 5.2.26, we have $\pi'(f) < \pi(e_b)$. Hence, $\pi(f) = \pi'(f) < \pi(e_b)$ by Claim 5.2.28. Since both permutations are identical for ranks lower than $\pi(e_b)$, edge f must appear in $\text{RGMIS}(G', \pi)$ and the query-trail $\vec{P}$ is not a valid query-trail since $\text{EO}(e_i, \cdot, \pi)$ terminates upon calling $\text{EO}(f, \cdot, \pi)$ (see Figure 5.3). □

Proof of Lemma 5.2.23. We prove that query-trail $\vec{P'}$ is not a valid (v, π')-query-trail. Note that by Observation 5.2.27, $\text{EO}(e'_{b+1}, \cdot, \pi')$ calls $\text{EO}(e_{b+1}, \cdot, \pi')$ before $EO(e_b, \cdot, \pi')$. Thus, by Claim 5.2.29, $\text{EO}(e_{b+1}, \cdot, \pi')$ will return TRUE and execution of $\text{EO}(e'_{b+1}, \cdot, \pi')$ terminates at this point. Therefore, query-trail $\vec{P'}$ is a subgraph of query-trail $\vec{P}$. □

Now we are ready to complete the proof of Lemma 5.2.22.

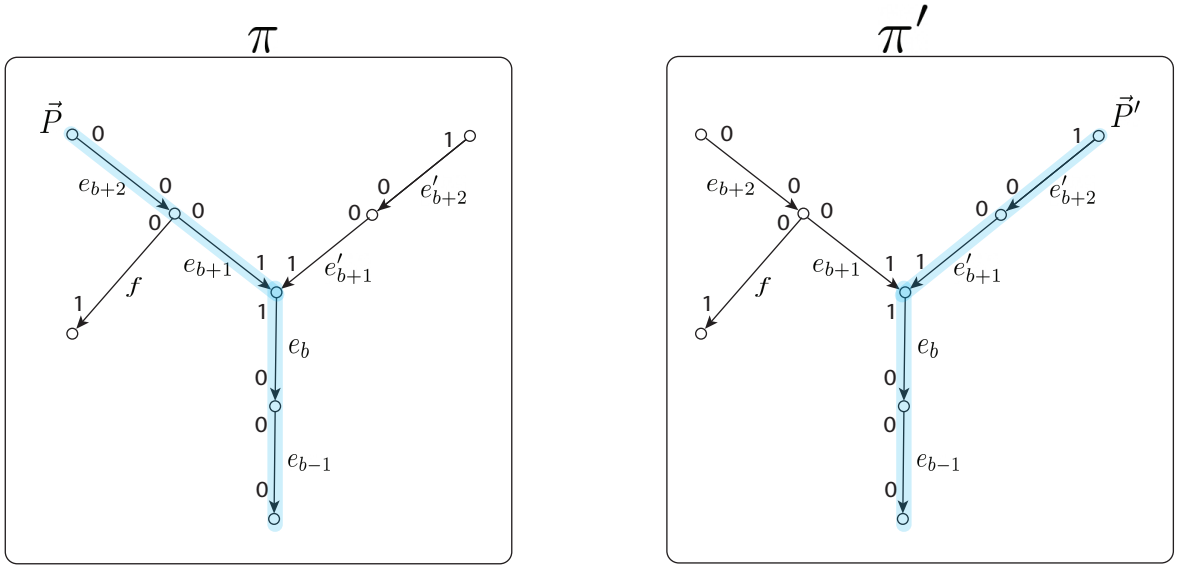


Figure 5.3: Illustration of proof of Claim 5.2.29. The highlighted blue trails show query-trails $\vec{P}$ and $\vec{P}'$. Query-trail $\vec{P}$ is not valid since $\text{EO}(e_i, \cdot, \pi)$ terminates upon calling $\text{EO}(f, \cdot, \pi)$.

Proof of Lemma 5.2.22. For each edge between $\pi_A \in A_L$ and $\pi_B \in B$ in graph H, we write a label $\chi(\pi_A, \pi_B)$ on the edge which is equal to the length of the query-trail corresponding to this edge in H. By Lemma 5.2.23, all the labels for edges of a fixed vertex $\pi_B \in B$ that are incident to A_L should be different. Moreover, by the definition of likely permutations, all query-trails of permutation A_L have length less than or equal to β. Thus, each vertex $\pi_B \in B$ has at most β neighbors in A_L. □

5.2.4 Our Estimator for Maximum Path Cover

In this section, we use the oracle of the previous section to estimate the number of edges in the output of Algorithm 21. In Section 5.2.3, we provide a lower bound on the number of recursive calls to our local query process. Note that this bound does not necessarily imply the same running time algorithm. For example, if we generate the whole permutation over all copies of edges before running the algorithm, it takes $\Theta(m)$ which is no longer sublinear. Using by now standard ideas of the literature, we show in Section 5.2.10 how we can implement the query process in almost the same running time (multiplied by a polylogarithmic factor) which is formalized in the following lemma.

Lemma 5.2.30. *There exists a data structure that given a graph G in the adjacency list format, (implicitly) fixes a random permutation π over its edges. Then for any vertex v, the data structure returns the degree of vertex v in the subgraph P produced by Algorithm 21 according to a random permutation π. Each query v to the data structure is answered in $\tilde{O}(T(v, \pi))$ time w.h.p. where $T(v, \pi)$ is as defined in Section 5.2.3.*

Note that in our local query process, we need access to the adjacency list of weight-one edges. So the challenge that arises here is how to estimate the size of the output of Algorithm 21 in the adjacency matrix model. We present a reduction from adjacency matrix to adjacency list that appeared in the literature [34].

In this reduction, each query to the adjacency list can be implemented with $O(1)$ queries to the adjacency matrix and still we are able to estimate the maximum path cover with some additive error.

Let $\gamma = 16Kn$. We construct a graph $\hat{G} = (V_{\hat{G}}, E_{\hat{G}})$ for weight-one edges of graph G as follows:

- $V_{\hat{G}}$ is the union of V_1, V_2 and $U_1, U_2, \ldots, U_n$ such that:
  - V_1 and V_2 are two copies of the vertex set of the original graph G.
  - U_i is a vertex set of size γ for each $i \in [n]$.
- We define the edge set such that degree of each vertex is in $\{1, n, n+\gamma\}$:
  - Degree of each vertex $v \in V_1$ is n. The i-th neighbor of v is the i-th vertex in V_1 if $(v,i) \in E$, otherwise its i-th neighbor is the i-th vertex in V_2 for $i \leq n$. Note that graph $(V_1, E_H \cap (V_1 \times V_1))$ is isomorphic to G.
  - Degree of each vertex $v \in V_2$ is $n+\gamma$. The i-th neighbor of v is the i-th vertex in V_2 if $(v,i) \in E$, otherwise, its i-th neighbor is the i-th vertex in V_1 for $i \leq n$. For all $n < i \leq n+\gamma$, the i-th neighbor of v is i-th vertex in U_v.
  - Degree of each vertex $u \in U_i$ is one for $i \in [n]$. The only neighbor of u is the i-th vertex of V_2.

By the construction of $\hat{G}$, the only neighbor of $v \in \bigcup_{i=1}^{n} U_i$ can be determined without any query to the adjacency matrix. Also, the i-th neighbor of each vertex in $V_1 \cup V_2$ can be determined with one query.

Observation 5.2.31. *For each vertex $v \in V_{\hat{G}}$ and $i \in [\deg_{\hat{G}}(v)]$, the i-th neighbor of vertex v can be determined using at most one query to the adjacency matrix.*

Fix a vertex $v \in V_2$. When we run Algorithm 21, intuitively with high probability the first edge that is incident to v and occupies port v^0 is between v and $u \in U_v$. Furthermore, with high probability the first two edges that are incident to v and occupies port v^1 are between v and $u \in U_v$. A vertex $v \in V_2$ is an *abnormal* vertex if the above properties do not hold for v. Let $R \in V_2$ be the set of abnormal vertices. In the following observation, we show that for each vertex $v \in V_2 \setminus R$, all incident edges of v in the output of Algorithm 21 are between v and vertices of U_v.

Claim 5.2.32. $\mathbf{E}_\pi |R| \leq n/(4K)$

Proof. Fix a vertex $v \in V_2$. For a random permutation over copies of edges of $\hat{G}$, the first incident edge to v that occupies port v^0 is between v and U_v with a probability of at least $\frac{(K+1)\gamma}{(n+\gamma)(K+1)} \geq 1 - \frac{1}{8K}$. Moreover, the first two edges that occupy v^1 are between v and U_v with probability of at least $\frac{\gamma(\gamma-1)}{(n+\gamma)(n+\gamma-1)} \geq 1 - \frac{1}{8K}$. Since both events are independent, the probability of v not being an abnormal vertex is at least

$$\left(1 - \frac{1}{8K}\right)^2 \geq 1 - \frac{1}{4K},$$

which implies $\mathbf{E}_\pi |R| \leq n/(4K)$. □

Claim 5.2.33. *For each $v \in V_2 \setminus R$, all incident edges of v in the output of Algorithm 21 are between v and vertices of U_v.*

Proof. By definition of an abnormal vertex, let the first edge in the permutation incident to v be between v and $w \in U_v$ which occupies v^0. Since all copies of edges incident to w are between v and w, this edge will be added to the solution of Algorithm 21. Moreover, we know that the first two edges that are incident to v and occupy port v^1 are between v and U_v. Let these two edges be (v, u_1) and (v, u_2) where $u_1, u_2 \in U_v$. Note that the only way that (v, u_1) is not added to the solution of Algorithm 21 is when $u_1 = w$. In this case, since there is only one copy for each edge that occupied port v^1, then $u_2 \neq w$. Therefore, Algorithm 21 adds (v, u_2) to its output if it has not added (v, u_1). Since both ports of v are occupied in this case, all incident edges of v in the output of Algorithm 21 are between v and vertices of U_v. □

Observation 5.2.34. *Let P be the output of Algorithm 21 on $\hat{G}$. Then*

$$\frac{1}{2}\rho(\hat{G}[V_1 \cup R]) \leq \mathbf{E}\,|P \cap (V_1 \cup R) \times (V_1 \cup R)| \leq (1 + \frac{2}{K}) \cdot \rho(\hat{G}[V_1 \cup R]).$$

Proof. By Claim 5.2.33, if we run Algorithm 21 on $\hat{G}$, for any vertex $v \in V_2 \setminus R$, all incident edges of v in the output are between v and U_v. Hence, none of the edges between $V_2 \setminus R$ and $V_1 \cup R$ will be added to the output of Algorithm 21. Since, the permutation over edges of $V_1 \cup R$ is uniformly at random, by Lemma 5.2.9, we obtain the claimed bound. □

In the above sequence of observations, we show that there are few abnormal vertices in V_2, which implies that most of the incident edges to vertices of V_1 in the output of Algorithm 21 are in $\hat{G}[V_1]$ (only those between V_1 and R violate this property). Therefore, a natural way to estimate the number of edges in the output of Algorithm 21 on G, is to estimate the number of edges in $\hat{G}[V_1]$ in the output of Algorithm 21 on $\hat{G}$. With this intuition in mind, we need to bound the query complexity of the algorithm for a random vertex in V_1.

Claim 5.2.35. *Let v be a random vertex in V_1 and π be a random permutation over edges of graph that is created by copying $E_{\hat{G}}$ according to Algorithm 21. Then*

$$\mathbf{E}_{v \sim V_1, \pi}[T(v, \pi)] = \tilde{O}(n).$$

Proof. By Theorem 5.2.13, we have that

$$\mathbf{E}_{v \sim V_{\hat{G}}, \pi}[T(v, \pi)] = O(\frac{K \cdot |E_{\hat{G}}|}{|V_{\hat{G}}|} \cdot \log^2 |V_{\hat{G}}|).$$

Summing over all vertices in $V_{\hat{G}}$, we obtain

$$\sum_{v \in V_{\hat{G}}} \mathbf{E}_{\pi}[T(v, \pi)] = O(K \cdot |E_{\hat{G}}| \cdot \log^2 |V_{\hat{G}}|) = \tilde{O}(n^2),$$

since $|V_{\hat{G}}| = O(n^2)$, $K = O(1/\varepsilon)$, and $|E_{\hat{G}}| = O(n^2)$. Therefore, for a random vertex in V_1, we get

$$\mathbf{E}_{v \sim V_1, \pi}[T(v, \pi)] \leq \left(\sum_{v \in V_{\hat{G}}} \mathbf{E}_{\pi}[T(v, \pi)] \right) / |V_1| = \tilde{O}(n).$$ □

Algorithm 24: Final algorithm for maximum path cover.

Let $\hat{G} = (V_{\hat{G}}, E_{\hat{G}})$ as described above.
$r \leftarrow 192 \cdot K^2 \cdot \log n$.
Sample r vertices $u_1, u_2, \ldots, u_r$ uniformly at random from V_1 with replacement.
Sample r ports $p_1, p_2, \ldots, p_r$ uniformly at random from $\{0, 1\}$.
Run vertex oracle for each u_i and let X_i be the indicator if port $u_i^{p_i}$ is occupied with an edge in $\hat{G}[V_1]$ in output of Algorithm 21.
Let $X = \sum_{i \in [r]} X_i$ and $f = X/r$.
Let $\tilde{\rho} = \frac{K}{2(K+2)} \cdot (f \cdot n - \frac{n}{4K})$.
return $\tilde{\rho}$

Lemma 5.2.36. *Let $\tilde{\rho}$ be the output of Algorithm 24 on input graph G. With high probability,*

$$\left(\frac{1}{2} - \frac{1}{K}\right) \cdot \rho(G) - \frac{n}{K} \leq \tilde{\rho} \leq \rho(G),$$

where K is the parameter which is defined in Algorithm 21.

Proof. Let $\hat{P}$ be the set of edges outputted by Algorithm 21 on $\hat{G}$ with both endpoints in V_1. By Lemma 5.2.9, we have that $\mathbf{E}\,|\hat{P}| \leq (1 + \frac{2}{K}) \cdot \rho(G)$. Furthermore, by Claim 5.2.32 and the fact that the degree of each vertex in the output of Algorithm 21 is at most two, in the output of Algorithm 21 on $\hat{G}[V_1 \cup R]$ we have at most $n/(2K)$ edges with one endpoint in R. Hence, combining with Lemma 5.2.9 and Observation 5.2.34 we get

$$\frac{1}{2}\rho(G) - \frac{n}{2K} \leq \mathbf{E}\,|\hat{P}| \leq (1 + \frac{2}{K}) \cdot \rho(G). \tag{5.9}$$

Since each edge in the output of Algorithm 21 counted twice in Algorithm 24, we have

$$\mathbf{E}[X_i] = \Pr[X_i = 1] = \frac{2\,\mathbf{E}\,|\hat{P}|}{n},$$

and,

$$\mathbf{E}[X] = \frac{2r\,\mathbf{E}\,|\hat{P}|}{n}. \tag{5.10}$$

Since X is sum of r independent random variables, by Chernoff bound (Proposition 2.3.1) we get

$$\Pr[|X - \mathbf{E}[X]| \leq \sqrt{6\,\mathbf{E}[X] \log n}] \leq 2 \exp\left(-\frac{6\,\mathbf{E}[X] \log n}{3\,\mathbf{E}[X]}\right) = \frac{2}{n^2}.$$

Combining $fn = Xn/r$ and the above bound, with probability of at least $1 - 2/n^2$ we have

$$\begin{aligned} fn \in \frac{n(\mathbf{E}[X] \pm \sqrt{6\,\mathbf{E}[X] \log n})}{r} &= \frac{n\,\mathbf{E}[X]}{r} \pm \sqrt{\frac{6n^2\,\mathbf{E}[X] \log n}{r^2}} \\ &= 2\,\mathbf{E}\,|\hat{P}| \pm \sqrt{\frac{12n\,\mathbf{E}\,|\hat{P}| \log n}{r}} \qquad \text{(By (5.10))} \end{aligned}$$

$$= 2\,\mathbf{E}\,|\hat{P}| \pm \sqrt{\frac{n\,\mathbf{E}\,|\hat{P}|}{16K^2}} \qquad \text{(Since } r = 192 \cdot K^2 \cdot \log n\text{)}$$
$$\in 2\,\mathbf{E}\,|\hat{P}| \pm \frac{n}{4K} \qquad \text{(Since } \mathbf{E}\,|\hat{P}| \leq n\text{)}.$$

Since, $\tilde{\rho} = \frac{K}{2(K+2)} \cdot (f \cdot n - \frac{n}{4K})$, hence

$$\frac{K}{K+2}\left(\mathbf{E}\,|\hat{P}| - \frac{n}{2K}\right) \leq \tilde{\rho} \leq \frac{K}{K+2} \cdot \mathbf{E}\,|\hat{P}|.$$

Combining with (5.9), implies the claimed bound. □

Theorem 5.2.37. *Given an adjacency matrix access for input graph G, there exists a randomized algorithm that w.h.p. runs in $\widetilde{O}(n)$ time and produces an estimate $\tilde{\rho}$, such that*

$$\left(\frac{1}{2} - \varepsilon\right) \cdot \rho(G) - \varepsilon n \leq \tilde{\rho} \leq \rho(G).$$

Proof. Let $K = \frac{1}{\varepsilon}$ and $\tilde{\rho}$ be the output of Algorithm 24 on G. In Algorithm 24, by combining Lemma 5.2.30 and Claim 5.2.35, the running time for each sample is $\tilde{O}(n)$. Since the number of samples is $r = 192K^2 \log n$, and K is a constant, the total running time of the algorithm is $\tilde{O}(n)$. Moreover, by Lemma 5.2.36 we get the approximation ratio in the statement. □

5.2.5 Our Estimator for (1,2)-TSP

In this section, we use the algorithm for estimating the size of maximum path cover as a black box to estimate the size of (1,2)-TSP. First, note that if there is no Hamiltonian cycle with weight one edges of the graph, then the set of weight-one edges of the graph (1,2)-TSP is a solution for maximum path cover of graph G. Also, in the case that there exists a Hamiltonian cycle, then the size of maximum path cover is $n-1$. Moreover, if P^* is the maximum path cover of a graph G, then it is possible to create a TSP by connecting these paths using edges with weight two. This intuition helps to formalize the following observation.

Observation 5.2.38. *Let $\tau(V)$ be the cost of (1,2)-TSP of graph $G = (V, E)$. Then, we have*

$$2n - \rho(G) - 1 \leq \tau(V) \leq 2n - \rho(G).$$

Now we are ready to present the final algorithm for estimating (1,2)-TSP.

Algorithm 25: Final algorithm for (1,2)-TSP.

1 Construct $\hat{G} = (V_{\hat{G}}, E_{\hat{G}})$ implicitly as desribed in Section 5.2.4.
2 Let $\tilde{\rho}$ be the output of Algorithm 24 on $\hat{G}$.
3 $\tilde{\tau} = 2n - \tilde{\rho}$
4 **return** $\tilde{\tau}$

Lemma 5.2.39. *Let $\tilde{\tau}$ be the output of Algorithm 25 and $\tau(V)$ be the cost of (1,2)-TSP of graph $G = (V, E)$. With high probability,*

$$\tau(V) \leq \tilde{\tau} \leq (\frac{3}{2} + \frac{4}{K}) \cdot \tau(V),$$

where K is the parameter which is defined in Algorithm 21.

Proof. By Observation 5.2.38, we have $2n-\rho(G)-1 \leq \tau(V) \leq 2n-\rho(G)$. Algorithm 25 outputs $\tilde{\tau} = 2n-\tilde{\rho}$ as the estimate, where $\tilde{\rho}$ is the output of Algorithm 24. Hence, by Lemma 5.2.36, we have $2n-\tilde{\rho} \geq 2n-\rho(G)$. Also, by Lemma 5.2.36, we have

$$\begin{aligned}
2n-\tilde{\rho} &\leq 2n-(\frac{1}{2}-\frac{1}{K})\cdot\rho(G)+\frac{n}{K} \\
&\leq 3n-\frac{3\rho(G)}{2}+\frac{4n}{K}-\frac{2\rho(G)}{K}-1 && \text{(Since } \rho(G)<n\text{)} \\
&\leq (\frac{3}{2}+\frac{4}{K})(2n-\rho(G)-1) && \text{(Since } K \ll n\text{)} \\
&\leq (\frac{3}{2}+\frac{4}{K})\cdot\tau(V) && \text{(Since } \tau(V)=2n-\rho(G)\text{)},
\end{aligned}$$

which completes the proof. □

Theorem 5.2.40. *Let $\tau(V)$ be the cost of (1,2)-TSP of graph $G=(V,E)$. For any $\varepsilon>0$, there exists an algorithm that estimate the cost of (1,2)-TSP, $\tilde{\tau}$, such that*

$$\tau(V) \leq \tilde{\tau} \leq (\frac{3}{2}+\varepsilon)\cdot\tau(V),$$

w.h.p in $\tilde{O}(n)$ running time.

Proof. We choose $K=\frac{4}{\varepsilon}$. By Lemma 5.2.39, if $\tilde{\tau}$ is the output of Algorithm 25, we get

$$\tau(V) \leq \tilde{\tau} \leq (\frac{3}{2}+\varepsilon)\cdot\tau(V).$$

Also, since the running time of Algorithm 25 is the same as the running time of Algorithm 24, by Theorem 5.2.37, the total running time is $\tilde{O}(n)$, which completes the proof. □

5.2.6 Our Estimator for Graphic TSP

In this section, we use our algorithm for estimating the size of maximum path cover to estimate the size of graphic TSP. In a recent work, Chen et al. [68] showed that it is possible to obtain a (27/14)-approximate algorithm for graphic TSP by estimating the matching size and the number of biconnected components in the graph. Since the size of graphic TSP is at most $2n$ (the cost of MST is $n-1$), they proved that if a graph has large matching and a few biconnected components, the cost of graphic TSP is significantly lower than $2n$. Since estimating the number of biconnected components is not an easy task in sublinear time, they use a proxy quantity that can be estimated in sublinear time.

We show that if we use our estimator for maximum path cover as a black-box instead of matching estimator in algorithm of [68], we can improve the approximation ratio to 19/10. Moreover, we show that we can estimate the number of bridges in $\tilde{O}(n)$. We exploit this estimation for further improvement to get a 11/6-approximate algorithm for graphic TSP.

Chen et al. [68] introduced the following definition of *bad vertex* as a proxy for estimating the number of biconnected components.

Definition 5.2.41 (Bad Vertex)**.** *We say a vertex $v \in V$ is a bad vertex, if one of the following holds:*

- *degree of v is 1,*
- *v is a cut vertex with degree 2.*

In the following series of lemmas, we bound the cost of graphic TSP based on the size of maximum path cover and number of bad vertices. Almost all the steps of this part are similar to the algorithm for graphic TSP of [68] — except the path cover subroutine that we use instead of maximal matching subroutine. We restate some of the useful lemmas to achieve the approximation bound that the black-box algorithm can get, and in the next subsection we improve this bound. First, we prove that if the size of the maximum path cover is small, the cost of graphic TSP is bounded away from n.

Claim 5.2.42. *If the size of maximum path cover of graph G is at most ρ, then the cost of graphic TSP is at least $2n - \rho$.*

Proof. Let $(v_0, v_2, \ldots, v_n = v_0)$ be the optimal graphic TSP of graph G. Note that the subgraph induced by weight-one edges of this cycle is a solution for path cover. Hence, at most ρ edges in cycle $(v_0, v_2, \ldots, v_n = v_0)$ have weight one. All the remaining edges have a weight of at least two which implies the claimed bound. □

Furthermore, the following lemma from [68], provides a lower bound for a graphic TSP of graph in terms of number of bad vertices.

Lemma 5.2.43 ([68, Lemma 2.8])**.** *If the number of bad vertices of graph G is at least β, then the cost of graphic TSP is at least $n + \beta - 2$.*

Chen et al. [68] showed that in a biconnected graph, if there exists a matching of large size, the cost of graphic TSP is significantly smaller than $2n$.

Lemma 5.2.44 ([68, Lemma 2.7])**.** *Let G be a graph and M be a matching of G. Then the cost of graphic TSP is at most $2n - |M|$.*

Lemma 5.2.45 ([68, Lemma 2.11])**.** *Let G be a graph and M' be a matching that none of its edges is a bridge. Then the cost of graphic TSP is at most $2n - \frac{2}{3}|M'|$.*

We now upper bound the cost of graphic TSP in terms of size of maximum path cover and number of bad vertices.

Lemma 5.2.46. *If the size of maximum path cover of graph G is $\rho(G)$ and it has β bad vertices, then the cost of graphic TSP is at most $2n - \frac{1}{5}(\rho(G) - 2\beta)$.*

Proof. Let l be the number of non-trivial biconnected components and M' be a maximum matching in graph G that none of its edges is a bridge. Also, let B be the number of bridges in G. By the proof of Lemma 2.9 of [68], the cost of graphic TSP is at most $\min\{2n - \frac{2}{3}|M'|, 2n - l\}$. Note that there are at least $\rho(G) - B$ edges of the maximum path cover that are not bridge. Since all non-bridge edges of the maximum path cover are still union of several disjoint paths, there exists a matching with size of at least half of the edges

of these paths. Hence, there exist a matching of size at least $\frac{1}{2}(\rho(G) - B)$. On the other hand, in the proof of the same lemma, they showed that $l \geq \frac{B}{2} - \beta$ which implies that the cost of graphic TSP is at most

$$\min\left\{2n - \frac{2}{3}|M'|, 2n - l\right\} \leq \min\left\{2n - \frac{1}{3}(\rho(G) - B), 2n - \frac{B}{2} + \beta\right\}.$$

There are two possible cases:

- If $B \leq \frac{2}{5}\rho(G) + \frac{6}{5}\beta$, then we have

$$2n - \frac{1}{3}(\rho(G) - B) \leq 2n - \frac{1}{3}(\rho(G) - \frac{2}{5}\rho(G) - \frac{6}{5}\beta) = 2n - \frac{1}{5}(\rho(G) - 2\beta).$$

- If $B > \frac{2}{5}\rho(G) + \frac{6}{5}\beta$, then we have

$$2n - \frac{B}{2} + \beta \leq 2n - \frac{1}{5}\rho(G) - \frac{3}{5}\beta + \beta = 2n - \frac{1}{5}(\rho(G) - 2\beta).$$

Therefore, the cost of graphic TSP is at most

$$\min\left\{2n - \frac{1}{3}(\rho(G) - B), 2n - \frac{B}{2} + \beta\right\} \leq 2n - \frac{1}{5}(\rho(G) - 2\beta). \quad \square$$

Now we are ready to introduce the first algorithm for estimating the cost of graphic TSP, which uses our maximum path cover subroutine instead of the matching subroutine as a black-box. In Algorithm 26, we first estimate the size of the maximum path cover and the number of bad vertices of the graph and report the graphic TSP cost in terms of the two estimations. The subroutine used for counting number of bad vertices is similar to the one in section 2.2 of [68].

Lemma 5.2.47 ([68])**.** *Let β be the number of bad vertices. For any constant $\varepsilon > 0$, there exists an algorithm that w.h.p estimates the number of bad vertices $\tilde{\beta}$, such that $\beta \leq \tilde{\beta} \leq \beta + \varepsilon n$, in $\tilde{O}(n)$ running time.*

Algorithm 26: First algorithm for graphic TSP.

1 Construct $\hat{G} = (V_{\hat{G}}, E_{\hat{G}})$ implicitly as desribed in Section 5.2.4.
2 Let $\tilde{\rho}$ be the output of Algorithm 24 on $\hat{G}$.
3 Let $\tilde{\beta}$ be the estimate of number of bad vertices.
4 $\tilde{\tau} = 2n - \frac{1}{5}(\tilde{\rho} - 2\tilde{\beta})$
5 **return** $\tilde{\tau}$

Lemma 5.2.48. *Let $\tilde{\tau}$ be the output of Algorithm 26 and $\tau(V)$ be the cost of graphic TSP of graph $G = (V, E)$. With high probability,*

$$\tau(V) \leq \tilde{\tau} \leq (\frac{19}{10} + \frac{1}{K}) \cdot \tau(V),$$

where K is the parameter which is defined in Algorithm 21.

Proof. Let β be the number of bad vertices. By Lemma 5.2.36 and Lemma 5.2.47 $\tilde{\rho} \leq \rho(G)$ and $\beta \leq \tilde{\beta}$. Hence, we have $\tau(V) \leq \tilde{\tau}$ by Lemma 5.2.46.

By Lemma 5.2.36 and Lemma 5.2.47, we can estimate $\rho(G)$ and β such that $\left(\frac{1}{2}-\frac{1}{K}\right)\rho(G)-\frac{n}{K}\leq\tilde{\rho}$ and $\tilde{\beta}\leq\beta+\frac{n}{K}$, if we choose $\varepsilon=\frac{1}{K}$. Thus, we have

$$\begin{aligned}\tilde{\tau} &\leq 2n-\frac{1}{5}\left((\frac{1}{2}-\frac{1}{K})\cdot\rho(G)-\frac{n}{K}-2(\beta+\frac{n}{K})\right)\\ &\leq 2n-\frac{1}{5}(\frac{\rho(G)}{2}-2\beta)+\frac{4n}{5K}. && \text{(Since } \rho(G)\leq n\text{).}\end{aligned}$$

On the other hand, assume that the approximation ratio that the algorithm obtains is $\alpha+1/K$ for some $\alpha\leq 2$. Thus, we get

$$\begin{aligned}(\alpha+\frac{1}{K})\cdot\tau(V) &\geq \alpha\cdot\tau(V)+\frac{n}{K} && \text{(Since } \tau(V)\geq n\text{)}\\ &\geq \alpha\cdot\max\{2n-\rho(G),n+\beta-2\}+\frac{n}{K} && \text{(By Claim 5.2.42 and Lemma 5.2.43)}\\ &\geq \alpha\cdot\max\{2n-\rho(G),n+\beta\}+\frac{n}{K}-4.\end{aligned}$$

So in order to show that $\tilde{\tau}\leq(\alpha+\frac{1}{K})\cdot\tau(V)$, it is sufficient to show that

$$2n-\frac{1}{5}(\frac{\rho(G)}{2}-2\beta)+\frac{4n}{5K}\leq\alpha\cdot\max\{2n-\rho(G),n+\beta\}+\frac{n}{K}-4.$$

If n is large enough, then we have $\frac{4n}{5K}\leq\frac{n}{K}-4$, which implies that we need to prove

$$2n-\frac{1}{5}(\frac{\rho(G)}{2}-2\beta)\leq\alpha\cdot\max\{2n-\rho(G),n+\beta\}.$$

Now, let $\rho(G)=xn$ and $\beta=yn$ for $0\leq x\leq 1$ and $0\leq y\leq 1$. To obtain α, it suffices to solve the following program

$$\begin{aligned}\text{maximize}\quad & \alpha\\ \text{subject to}\quad & \tfrac{2-\frac{1}{5}(\frac{x}{2}-2y)}{\max\{2-x,1+y\}}\leq\alpha,\\ & 0\leq x\leq 1,\\ & 0\leq y\leq 1.\end{aligned}$$

This is a constant size program that can be easily solved; the solution is 19/10.[1] This completes the proof. □

Theorem 5.2.49. *Let $\tau(V)$ be the cost of graphic TSP of graph $G=(V,E)$. For any $\varepsilon>0$, there exists an*

[1] See e.g. this WolframAlpha link.

algorithm that estimate the cost of graphic TSP, $\tilde{\tau}$, such that

$$\tau(V) \leq \tilde{\tau} \leq (\frac{19}{10} + \varepsilon) \cdot \tau(V),$$

w.h.p in $\tilde{O}(n)$ running time.

Proof. Let $\tilde{\tau}$ be the output of Algorithm 26. If we choose $K = \frac{1}{\varepsilon}$, then by Lemma 5.2.48, we have

$$\tau(V) \leq \tilde{\tau} \leq (\frac{19}{10} + \varepsilon) \cdot \tau(V).$$

Also, by Theorem 5.2.37 and Lemma 5.2.47, estimating $\tilde{\rho}$ and $\tilde{\beta}$ can be done in $\tilde{O}(n)$ time. □

5.2.7 Further Improvement for Graphic TSP

In this section, we design an algorithm to estimate the number of bridges in given graph G. Equipped with this tool, we are able to estimate the number of non-bridge edges in the path cover which helps to improve the approximation ratio. Before describing the techniques for estimating the number of bridges, we prove the following lemma that provides a lower bound on the cost of graphic TSP based on the number of bridges in the graph.

Claim 5.2.50. *If the number of bridges in the graph G is at least B, then the cost of the graphic TSP is at least $n + B$.*

Proof. Since the metric in the graphic TSP is corresponding to the shortest path distances in graph G, then a TSP tour is corresponding to a closed walk that contains all vertices. Thus, each bridge should be crossed at least two times in this walk in order for the walk to be closed and cover all vertices. Therefore, the cost of graphic TSP is at least $n + B$. □

In the following series of lemmas, first, we prove that there are a few bridges that both of their endpoints have a high degree and then we show an efficient way to estimate the number of bridges that have at least one endpoint with a low degree. Combining the above arguments is the main idea to estimate the number of bridges.

Lemma 5.2.51. *For any integer $c \geq 2$, there exists at most $\frac{2n}{c}$ bridges that both of their endpoints have a degree larger than c.*

Proof. Let $\mathcal{B}$ be the set of bridges that both of their endpoints have a degree larger than c. We construct a tree, $T_{\mathcal{B}}$, with edge set equal to $\mathcal{B}$ such that each vertex of $T_{\mathcal{B}}$ corresponds to a component of vertices that are compressed to a single vertex. We construct $T_{\mathcal{B}}$ iteratively. In the beginning, we consider the bridge-block tree of the original graph. In each step, if there exists a bridge $e = (u, v)$ (note that u and v are vertices of the tree and corresponding to a set of vertices of the original graph) such that at least one of its endpoints has a degree less than or equal to c, we merge u with v and add all edges of u to v. We continue this process until there is no bridge with an endpoint of degree less than or equal to c.

Now, we prove that $|\mathcal{B}| \leq \frac{2n}{c}$. Let x_v denote the number of vertices in the original graph that are compressed to vertex $v \in T_{\mathcal{B}}$. We remove vertices of $T_{\mathcal{B}}$ one by one until there is no vertex in the tree. At each step, we remove a leaf $v \in T_{\mathcal{B}}$ and at the end when only one vertex is remaining, we remove that vertex. Let y_v be the number of incident edges to v in $T_{\mathcal{B}}$ that are removed before removing v. At the time that we

are removing leaf v, we have $x_v + y_v + 1 \geq c + 1$, since the endpoint of the leaf that is the component of v has at most x_v incident edges in the same component in the original graph, y_v incident edges to the other components that are removed before, and there is only one remaining incident edge to other components (the other endpoint of the leaf). Thus,

$$\sum_{v \in T_{\mathcal{B}}} x_v \geq \sum_{v \in T_{\mathcal{B}}} (c - y_v) = (|\mathcal{B}| + 1)c - \sum_{v \in T_{\mathcal{B}}} y_v. \tag{5.11}$$

Since vertices of each component are disjoint, we have $\sum_{v \in T_{\mathcal{B}}} x_v = n$. Moreover, we have $\sum_{v \in T_{\mathcal{B}}} y_v = |\mathcal{B}|$ since each edge of $\mathcal{B}$ counted when one of its endpoints is deleted from the tree. Combining above bounds and inequality (5.11), we have

$$n = \sum_{v \in T_{\mathcal{B}}} x_v \geq (|\mathcal{B}| + 1)c - |\mathcal{B}|$$

Therefore,

$$|\mathcal{B}| \leq \frac{n - c}{c - 1} \leq \frac{2n}{c},$$

where the last inequality holds for sufficiently large n. □

Lemma 5.2.52. *Let $c \geq 2$ be a constant and u is a vertex such that $\deg(u) \leq c$. Then we can test if each of incident edges of u is a bridge in $O(n)$ total running time.*

Proof. We can query all neighbors of u in $O(n)$. Assume that $\{v_1, v_2, \ldots, v_r\}$ are neighbors of u for $r \leq c$. Now we divide the vertices of the graph except u into r sets $V_1, V_2, \ldots, V_r$. For each vertex $w \neq u$, we query the distance of w to all $\{v_1, v_2, \ldots, v_r\}$. Let v_i be the closest one to w (if there is a tie, choose the one with the lowest index). Then we put w in V_i. Note that since c is a constant and $r \leq c$, this step can be done in $O(n)$.

Now we claim that (u, v_j) is a bridge iff the following conditions hold:

- For each $w \in V_j$ and $i \neq j$, $d(w, v_i) - d(w, v_j) = 2$.
- For each $w \in V_i$ such that $i \neq j$, $d(w, v_j) - d(w, v_i) = 2$.

Suppose that $e = (u, v_j)$ is a bridge. Since removing e creates two connected components C_u and C_{v_j}, all vertices in C_{v_j} (resp. C_u) have a closer distance to v_j (resp. u). In other words, all shortest paths between $w \in V_j$ to v_i for $i \neq j$, cross edges (v_j, u) and (u, v_i). In addition, all the shortest paths between $w \in V_i$ and v_j for $i \neq j$, cross edges (v_j, u) and (u, v_i). Therefore, both conditions hold.

Now suppose that $e = (u, v_j)$ is not a bridge. In this case, there must be an edge between V_j and at least one of V_i as otherwise, V_j will be disconnected from the rest of the graph by removing e. Without loss of generality, assume that this edge is (w, w') such that $w \in V_j$, $w' \in V_i$, and $i \neq j$. Also, w.l.o.g., we assume $d(w, v_j) \leq d(w', v_i)$. Since there is an edge between w and w', we have $d(w', v_j) \leq 1 + d(w, v_j) \leq 1 + d(w', v_i)$, which contradicts the conditions.

To test whether the conditions hold, we need to query the distance of each vertex to all $\{v_0, v_1, \ldots, v_r\}$ which can be done in $O(n)$ in total since r is a constant. □

Lemma 5.2.53. *Let B be the number of bridges in graph G. For any $\varepsilon > 0$, there exists an algorithm that outputs an estimate $\tilde{B}$ in $\tilde{O}(n)$ such that $B \leq \tilde{B} \leq B + \varepsilon n$.*

Proof. By Lemma 5.2.51, there are at most $\frac{\varepsilon n}{2}$ bridges with both endpoints have degree larger than $\frac{4}{\varepsilon}$. Let $\hat{B}$ be the number of bridges that at least one of their endpoint has degree of at most $\frac{4}{\varepsilon}$. Thus,

$$B - \frac{\varepsilon n}{2} \leq \hat{B} \leq B. \tag{5.12}$$

We sample $r = 256 \cdot \varepsilon^{-4} \cdot \log n$ vertices uniformly at random with replacement. Let u be the i-th sampled vertex. If the degree of the vertex is larger than $\frac{4}{\varepsilon}$, we let $X_i = 0$. Otherwise, let $\{v_1, v_2, \ldots, v_k\}$ be the neighbors of u where $k \leq \frac{4}{\varepsilon}$. By Lemma 5.2.52, we can test if each of the incident edges of u is a bridge in $O(n)$ total running time. For each edge (u, v_j) if $\deg(u) < \deg(v_j)$ or $\deg(u) = \deg(v_j)$ and index of u is smaller than v_j, we test if the edge is a bridge or not. Let X_i show the number of successful tests for incident edges of u. Note that in the above algorithm, each bridge with low-degree endpoints only counted once.

Let $\bar{X} = (\sum_i^r X_i)/r$ and $n\bar{X} + \frac{3\varepsilon n}{4}$ be our final estimate of the number of bridges. Hence, $\mathbf{E}[\bar{X}] = \hat{B}/n$. Since $\bar{X}$ is the average of r independent random variables such that $0 \leq X_i \leq 4/\varepsilon$, by Hoeffding's inequality (Proposition 2.3.2) we obtain

$$\Pr\left[\left|\bar{X} - \mathbf{E}[\bar{X}]\right| \geq \frac{\varepsilon}{4}\right] \leq 2\exp\left(-\frac{r\varepsilon^4}{128}\right) = \frac{2}{n^2},$$

where the last inequality follows from $r = 256 \cdot \varepsilon^{-4} \cdot \log n$. Therefore, with probability of $1 - \frac{2}{n^2}$,

$$\begin{aligned} n\bar{X} &\in n\,\mathbf{E}[\bar{X}] \pm \frac{n\varepsilon}{4} \\ &= \hat{B} \pm \frac{n\varepsilon}{4} && \text{(Since } \mathbf{E}[\bar{X}] = \hat{B}/n\text{).} \end{aligned}$$

Combining above range and inequality (5.12), we get

$$B \leq n\bar{X} + \frac{3\varepsilon n}{4} \leq B + \varepsilon n.$$

Since the number of sampled vertices is $r = 256 \cdot \varepsilon^{-4} \cdot \log n$, the total running time is $\tilde{O}(n)$. □

Now we are ready to introduce the improved algorithm for graphic TSP.

Algorithm 27: Improved algorithm for graphic TSP.

1 Construct $\hat{G} = (V_{\hat{G}}, E_{\hat{G}})$ implicitly as described in Section 5.2.4.
2 Let $\tilde{\rho}$ be the output of Algorithm 24 on $\hat{G}$.
3 Let $\tilde{B}$ be the estimate of the number of bridges.
4 $\tilde{\tau} = 2n - \frac{1}{3}(\tilde{\rho} - \tilde{B})$
5 **return** $\tilde{\tau}$

Lemma 5.2.54. *Let $\tilde{\tau}$ be the output of Algorithm 27 and $\tau(V)$ be the cost of graphic TSP of graph $G = (V, E)$. With high probability,*

$$\tau(V) \leq \tilde{\tau} \leq (\frac{11}{6} + \frac{1}{K}) \cdot \tau(V),$$

where K is the parameter which is defined in Algorithm 21.

Proof. Let $\rho(G)$ be the size of maximum path cover and B be the number of bridges in the graph. There are at least $\rho(G) - B$ edges of maximum path cover that are not bridge. These edges construct disjoint paths which implies there exists a matching of size $\frac{1}{2}(\rho(G) - B)$ that none of its edges is a bridge. Hence, by Lemma 5.2.45, the cost of graphic TSP is at most $2n - \frac{1}{3}(\rho(G) - B)$. Therefore, since $\tilde{\rho} \leq \rho(G)$ and $B \leq \tilde{B}$, we get $\tau(V) \leq \tilde{\tau}$.

By Lemma 5.2.36 and Lemma 5.2.53, we have $\left(\frac{1}{2} - \frac{1}{K}\right) \cdot \rho(G) - \frac{n}{K} \leq \tilde{\rho}$ and $\tilde{B} \leq B + \frac{n}{K}$ which implies

$$\begin{aligned}\tilde{\tau} &\leq 2n - \frac{1}{3}\left((\frac{1}{2} - \frac{1}{K}) \cdot \rho(G) - (B + \frac{n}{K})\right) \\ &\leq 2n - \frac{1}{3}(\frac{\rho(G)}{2} - B) + \frac{2n}{K} && \text{(Since } \rho(G) \leq n\text{).}\end{aligned}$$

Also, assume that the approximation ratio that the algorithm obtains is $\alpha + 2/K$ for some $\alpha \leq 2$. Thus,

$$\begin{aligned}(\alpha + \frac{2}{K}) \cdot \tau(V) &\geq \alpha \cdot \tau(V) + \frac{2n}{K} && \text{(Since } \tau(V) \geq n\text{)} \\ &\geq \alpha \cdot \max\{2n - \rho(G), n + B\} + \frac{2n}{K} && \text{(By Claim 5.2.42 and Claim 5.2.50).}\end{aligned}$$

Therefore, to show that $(\alpha + \frac{1}{K}) \cdot \tau(V) \geq \tilde{\tau}$, it is sufficient to show

$$2n - \frac{1}{3}(\frac{\rho(G)}{2} - B) \leq \alpha \cdot \max\{2n - \rho(G), n + B\}.$$

Now, let $\rho(G) = xn$ and $B = yn$ for $0 \leq x \leq 1$ and $0 \leq y \leq 1$. To obtain α, we write the following maximization problem,

$$\begin{aligned}\text{maximize} \quad & \alpha \\ \text{subject to} \quad & \tfrac{2 - \frac{1}{3}(\frac{x}{2} - y)}{\max\{2-x, 1+y\}} \leq \alpha, \\ & 0 \leq x \leq 1, \\ & 0 \leq y \leq 1.\end{aligned}$$

The solution to this problem is $11/6$.[2] This completes the proof. □

Theorem 5.2.55. *Let $\tau(V)$ be the cost of graphic TSP of graph $G = (V, E)$. For any $\varepsilon > 0$, there exists an algorithm that estimate the cost of graphic TSP, $\tilde{\tau}$, such that*

$$\tau(V) \leq \tilde{\tau} \leq (\frac{11}{6} + \varepsilon) \cdot \tau(V),$$

w.h.p in $\tilde{O}(n)$ running time.

[2] See e.g. this WolframAlpha link.

Proof. Let $\tilde{\tau}$ be the output of Algorithm 27. If we choose $K = \frac{1}{\varepsilon}$, then by Lemma 5.2.54, we have

$$\tau(V) \leq \tilde{\tau} \leq (\frac{11}{6} + \varepsilon) \cdot \tau(V).$$

Also, by Theorem 5.2.37 and Lemma 5.2.53, estimating $\tilde{\rho}$ and $\tilde{B}$ can be done in $\tilde{O}(n)$ time. □

5.2.8 A Slightly Subquadratic Algorithm for Graphic TSP

With the recent advances in designing sublinear algorithms for maximum matching [42, 43, 57, 59], we can now achieve a more precise estimation of the size of graphic TSP at the cost of increased running time. In this section, we present an algorithm that approximates the graphic TSP with an accuracy within a factor of $5/3 + \varepsilon$ in $O(n^{2-\Omega_\varepsilon(1)})$. We use the following result by Bhattacharya, Kiss, and Saranurak [57] to design our slightly subquadratic graphic TSP estimator.

Proposition 5.2.56. *Suppose that we have access to the adjacency matrix of graph G. Then, There exists an algorithm that estimates the size of the maximum matching of graph G, $\tilde{\mu}$, such that*

$$\mu(G) - \varepsilon n \leq \tilde{\mu} \leq \mu(G),$$

w.h.p in $n^{2-\Omega_\varepsilon(1)}$ time.

Combining this algorithm with the framework of Chen, Kannan, and Khanna [68] implies a $(13/7 + \varepsilon)$-approximation for graphic TSP in $n^{2-\Omega_\varepsilon(1)}$ time. Our algorithm in Section 5.2.7 makes an improvement on both the running time and approximation ratio for the graphic TSP over this recent result. Furthermore, using Proposition 5.2.56 and our estimator for counting the number of bridges, we are able to obtain a 5/3-approximation ratio. Now we are ready to propose our algorithm.

Algorithm 28: Slightly subquadratic algorithm for graphic TSP.

1 Let $\tilde{\mu}$ be the output of Proposition 5.2.56 on G.
2 Let $\tilde{B}$ be the estimate of the number of bridges.
3 $\tilde{\tau} = 2n - \frac{1}{3}(\tilde{\mu} - \tilde{B})$
4 **return** $\tilde{\tau}$

Lemma 5.2.57. *Let $\tilde{\tau}$ be the output of Algorithm 28 and $\tau(V)$ be the cost of graphic TSP of graph $G = (V, E)$. With high probability,*

$$\tau(V) \leq \tilde{\tau} \leq (\frac{5}{3} + \varepsilon) \cdot \tau(V).$$

Proof. Let B be the number of bridges in G. There exists a matching with a size of at least $\mu(G) - B$ that none of its edges is a bridge. Thus, by Lemma 5.2.45, it holds $\tau(V) \leq 2n - \frac{1}{3}(\mu(G) - B)$. Combining with $\tilde{\mu} \leq \mu(G)$ and $B \leq \tilde{B}$, we get $\tau(V) \leq \tilde{\tau}$.

Additionally, by Proposition 5.2.56 and Lemma 5.2.53, we have $\tilde{\mu} \geq \mu(G) - \varepsilon n$ and $\tilde{B} \leq B + \varepsilon n$. Thus,

$$\begin{aligned}
\tilde{\tau} &\leq 2n - \frac{1}{3}\left((\mu(G) - \varepsilon n) - (B + \varepsilon n)\right) \\
&\leq 2n - \frac{1}{3}(\mu(G) - B) + 2\varepsilon n && \text{(Since } \mu(G) \leq n\text{)}.
\end{aligned}$$

Assume the approximation ratio of the algorithm is $\alpha + 2\varepsilon$ for some $\alpha \leq 2$, we must have

$$\begin{aligned}(\alpha + 2\varepsilon) \cdot \tau(V) &\geq \alpha \cdot \tau(V) + 2\varepsilon n && \text{(Since } \tau(V) \geq n\text{)} \\ &\geq \alpha \cdot \max\{2n - \mu(G), n + B\} + 2\varepsilon n. && \text{(By Lemma 5.2.44 and Claim 5.2.50)}\end{aligned}$$

In order to show $(\alpha + 2\varepsilon) \cdot \tau(V) \geq \tilde{\tau}$, it is sufficient to show

$$2n - \frac{1}{3}(\mu(G) - B) \leq \alpha \cdot \max\{2n - \mu(G), n + B\}.$$

Let $\mu(G) = xn$ and $B = yn$ for $0 \leq x \leq 1$ and $0 \leq y \leq 1$. To obtain α, we write the following maximization problem,

$$\begin{aligned}\text{maximize} \quad & \alpha \\ \text{subject to} \quad & \tfrac{2 - \frac{1}{3}(x-y)}{\max\{2-x, 1+y\}} \leq \alpha, \\ & 0 \leq x \leq 1, \\ & 0 \leq y \leq 1.\end{aligned}$$

The solution to this problem is 5/3.[3] □

Theorem 5.2.58. *Let $\tau(V)$ be the cost of graphic TSP of graph $G = (V, E)$. For any $\varepsilon > 0$, there exists an algorithm that estimate the cost of graphic TSP, $\tilde{\tau}$, such that*

$$\tau(V) \leq \tilde{\tau} \leq (\frac{5}{3} + \varepsilon) \cdot \tau(V),$$

w.h.p in $n^{2-\Omega_\varepsilon(1)}$ running time.

Proof. Let $\tilde{\tau}$ be the output of Algorithm 28. By Lemma 5.2.54, we have

$$\tau(V) \leq \tilde{\tau} \leq (\frac{5}{3} + \varepsilon) \cdot \tau(V).$$

Moreover, by Proposition 5.2.56 and Lemma 5.2.53, estimating $\tilde{\mu}$ and $\tilde{B}$ can be done in $n^{2-\Omega_\varepsilon(1)}$ time. □

5.2.9 Lower Bound for Approximating Maximum Path Cover

"Conditional" Hardness for the Approximation Ratio

In this section, we prove that if there exists a constant $\alpha > 0$ and an algorithm that returns a $(\frac{1}{2} + \alpha)$-approximate estimate for the size of maximum path cover in $\tilde{O}(n)$ time in a bipartite graph, then there is a $(\frac{1}{2} + \alpha)$-approximate algorithm for estimating the maximum matching size in $\tilde{O}(n)$ time. This remains an important open problem in the study of sublinear time maximum matching algorithms. See in particular

[3] See e.g. this WolframAlpha link.

[43]. This implies that short of a major result in the study maximum matchings in the sublinear time model, which have received significant attention in the literature (see [176, 34, 43, 42, 59] and references therein), our path cover algorithm has an optimal approximation ratio.

Let $G = (V, U, E)$ be a bipartite graph. We construct a graph $G' = (V', U', E')$ such that a better than $\frac{1}{2}$-approximate estimate of maximum path cover on G' leads to a better than $\frac{1}{2}$-approximate estimate of maximum matching in G. Let r be a large constant. We create r copies of G, showing the i-th copy with $G_i = (V_i, U_i, E)$. Also, we create another $r-1$ copies $H_1, \ldots, H_{r-1}$ of G with $H_i = (V_i, U_{i+1}, E)$. Now we let the $G' = (\bigcup_{i=1}^{r} G_i) \cup (\bigcup_{i=1}^{r-1} H_i)$. Now we claim that the size of maximum path cover of the graph G' is roughly $2r \cdot \mu(G)$ which can be used as an estimator for the maximum matching of G.

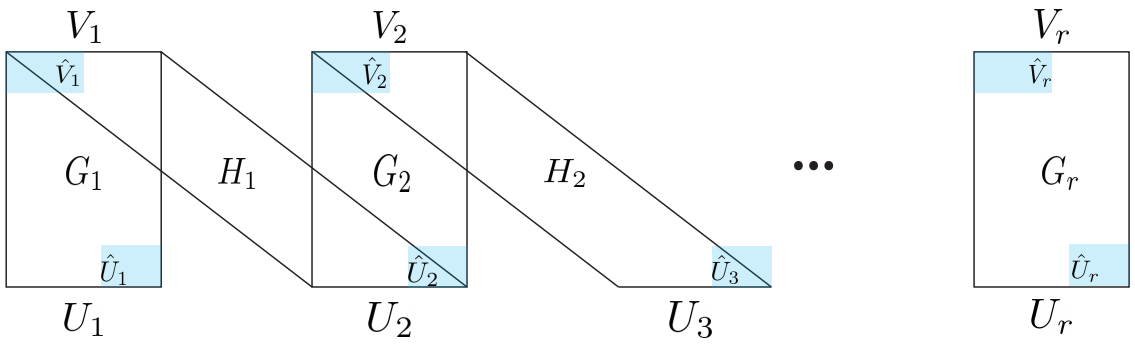


Figure 5.4: Illustration of graph $G' = (V', U', E')$. Each G_i is shown by a rectangle and each H_i is shown by a parallelogram. Top and bottom horizontal lines illustrate V_i and U_i. Blue highlighted parts represent the vertex cover of the graph.

Before proving the main result of this section, we characterize some properties of the constructed graph G'.

Claim 5.2.59. $\mu(G') = r \cdot \mu(G)$.

Proof. First, since all graphs $\{G_i\}_{i=1}^{r}$ are the same as G and are vertex-disjoint, if we consider the maximum matching of G in each of the r graphs, we will have a matching of size $r \cdot \mu(G)$. Thus, $\mu(G') \geq r \cdot \mu(G)$.

Let $\hat{V} \cup \hat{U}$ be the minimum vertex cover of G such that $\hat{V} \in V$ and $\hat{U} \in U$. By König's Theorem (Proposition 2.2.1), we have $|\hat{V} \cup \hat{U}| = \mu(G)$. Now we show there exists a vertex cover of size $r \cdot \mu(G)$ for graph G'. Let $\hat{V}_i \in V_i$ (resp. $\hat{U}_i \in U_i$) be the copy of vertices V (resp. U) in graph G_i. We claim $(\bigcup_{i=1}^{r} \hat{V}_i \cup \hat{U}_i)$ is a vertex cover for G'. If an edge is in G_i, then at least one of its endpoints is in $\hat{V}_i \cup \hat{U}_i$ since $\hat{V}_i \cup \hat{U}_i$ is a vertex cover of G_i. Moreover, by the construction, $\hat{V}_i \cup \hat{U}_{i+1}$ is a vertex cover of H_i. Hence, each edge of H_i is also covered by the vertex cover. Therefore, since there exists a vertex cover of size $|(\bigcup_{i=1}^{r} \hat{V}_i \cup \hat{U}_i)| = r \cdot |\hat{V} \cup \hat{U}| = r \cdot \mu(G)$, then we have $\mu(G') \leq r \cdot \mu(G)$ which completes the proof. □

Observation 5.2.60. *It holds* $(2r-1) \cdot \mu(G) \leq \rho(G') \leq 2r \cdot \mu(G)$.

Proof. Since the union of maximum matching of all graphs $\{G_i\}_{i=1}^{r}$ and $\{H_i\}_{i=1}^{r-1}$ creates a path cover, we get $(2r-1) \cdot \mu(G) \leq \rho(G')$. Futhermore, if there exists a path cover of size larger than $2r \cdot \mu(G)$, then the maximum matching of these paths will be larger than $r \cdot \mu(G)$ which contradicts Claim 5.2.59. Thus, $\rho(G') \leq 2r \cdot \mu(G)$. □

Now we are ready to show the reduction.

Lemma 5.2.61. *For any constant $\alpha > 0$, if there exists an algorithm that can estimate the maximum path cover within a $(\frac{1}{2}+\alpha)$-factor in $O(T(n))$ time, then the same algorithm can be used to estimate the maximum matching of bipartite graph G within a $(1-\varepsilon)\cdot(\frac{1}{2}+\alpha)$-factor in $O(T(n/\varepsilon))$ time.*
Proof. We construct graph G' as described at the beginning of the section with $r = \frac{1}{2\varepsilon}$. By Observation 5.2.60, $(\frac{1}{\varepsilon}-1)\cdot\mu(G) \leq \rho(G') \leq \frac{1}{\varepsilon}\cdot\mu(G)$. Let $\tilde{\rho}$ be the estimate of the algorithm for the maximum path cover of G'. Hence, we have

$$(\frac{1}{2}+\alpha)(\frac{1}{\varepsilon}-1)\cdot\mu(G) \leq \tilde{\rho} \leq \frac{1}{\varepsilon}\cdot\mu(G).$$

Now let $\tilde{\mu} = \varepsilon\cdot\tilde{\rho}$ be the estimate for the maximum matching of G. Hence,

$$(1-\varepsilon)(\frac{1}{2}+\alpha)\cdot\mu(G) \leq \tilde{\mu} \leq \mu(G).$$

Since the number of vertices and number of edges of G' is $r = \frac{1}{2\varepsilon}$ times more than G, then the running time will be $O(T(n/\varepsilon))$. □

A reduction to matchings can also be proved for $(1,2)$-TSP, albeit with an extra promise for the matching instance that the matching is either perfect or half-perfect. This problem, formalized below, also remains open in the study matchings. We show that a better than 1.5-approximation for $(1,2)$-TSP in $\widetilde{O}(n)$ time would resolve this question.

Problem 5.2.62. *Suppose that we are given a bipartite graph $G = (L, R, E)$ with $|L| = |R| = n$ and are promised that either $\mu(G) = n$ or $\mu(G) = (\frac{1}{2}+\varepsilon)n/2$ for any desirably small constant $\varepsilon > 0$. Provided adjacency matrix access to the graph, does there exist an $n^{1+o(1)}$ time algorithm that distinguishes the two?*

Theorem 5.2.63. *If there is an algorithm that estimates the size of $(1,2)$-TSP within a $(\frac{3}{2}-\varepsilon_0)$-factor for some fixed constant $\varepsilon_0 \in (0, \frac{1}{4}]$ in $n^{1+o(1)}$, then Problem 5.2.62 can indeed be solved in $n^{1+o(1)}$ time.*
Proof. Let G_1 and G_2 be two graphs with n vertices such that $\mu(G_1) = n$ and $\mu(G_2) = (\frac{1}{2}+\frac{\varepsilon_0}{16})$. We construct graph $G_1' = (V_1', E_1')$ and $G_2' = (V_2', E_2')$ as described at the beginning of the section with $r = \frac{1}{\varepsilon_0}$. By Observation 5.2.60, we have $\rho(G_1') \geq (\frac{2}{\varepsilon_0}-1)n$ and $\rho(G_2') \leq (\frac{1}{\varepsilon_0}+\frac{1}{8})n$. Thus, by Observation 5.2.38, we get

$$\tau(V_1') \leq \frac{4}{\varepsilon_0}n - (\frac{2}{\varepsilon_0}-1)n = (\frac{2}{\varepsilon_0}+1)n,$$
$$\tau(V_2') \geq \frac{4}{\varepsilon_0}n - (\frac{1}{\varepsilon_0}+\frac{1}{8})n - 1 \geq (\frac{3}{\varepsilon_0}-\frac{1}{4})n,$$

for sufficiently large n, which implies

$$\frac{\tau(V_2')}{\tau(V_1)} = \frac{3-\varepsilon_0/4}{2+\varepsilon_0} \geq \frac{3}{2}-\varepsilon_0.$$

Therefore, the algorithm for (1,2)-TSP can distinguish between G_1' and G_2' which implies Problem 5.2.62 can be solved in $n^{1+o(1)}$ time for $\varepsilon = \varepsilon_0/16$. □

Lower Bound for Graphic TSP: note that we can reduce an instance of (1,2)-TSP to graphic TSP by adding a new vertex and connecting the newly added vertex to all vertices of the graph. Therefore, the $n^{1+o(1)}$ time lower bound also holds for graphic TSP.

Information-Theoretic Lower Bounds on the Running Time

Since any constant approximation algorithm for estimating maximum path cover can be used to estimate the size of matching within a constant factor, then all of the lower bounds for $O(1)$-approximating maximum matching in sublinear time also hold for (1)-approximating maximum path cover in sublinear time. We restate some of these lower bounds along with a short proof (see [34] for a detailed discussion).

Lemma 5.2.64. *Any algorithm that estimates maximum path cover within a constant multiplicative factor requires $\Omega(n)$ queries in the adjacency list model.*

Proof. Consider two graphs that the first one does not have any edge and the second one has only a single edge. In order to give any multiplicative approximation for maximum path cover, the algorithm needs to find the edge which requires $\Omega(n)$ queries in the adjacency list model. □

Lemma 5.2.65. *Any algorithm that estimates maximum path cover within a constant multiplicative factor require $\Omega(n^2)$ queries in the adjacency matrix model.*

Proof. Consider the same construction as Lemma 5.2.64. To give any multiplicative approximation for maximum path cover, the algorithm needs to find the edge which requires $\Omega(n^2)$ queries in the adjacency matrix model. □

Lemma 5.2.66. *Any algorithm that estimates maximum path cover within a multiplicative-additive requires $\Omega(n)$ queries in the adjacency matrix model.*

Proof. Consider a graph with no edge and a graph with one Hamiltonian cycle and no other edges. In order for the algorithm to distinguish between these two graphs, it must find at least one edge of the second graph which requires $\Omega(n)$ queries in the adjacency matrix model. □

There is also a lower bound for multiplicative-additive estimation of matching in adjacency list model [159] that also holds for maximum path cover.

Lemma 5.2.67. *Any algorithm that estimates maximum path cover within a constant multiplicative-additive factor requires $\Omega(\bar{d})$ queries in the adjacency list model.*

5.2.10 Implementation Details

In this section, we discuss why Lemma 5.2.30, restated below, holds.

Lemma 5.2.30. *There exists a data structure that given a graph G in the adjacency list format, (implicitly) fixes a random permutation π over its edges. Then for any vertex v, the data structure returns the degree of vertex v in the subgraph P produced by Algorithm 21 according to a random permutation π. Each query v to the data structure is answered in $\tilde{O}(T(v,\pi))$ time w.h.p. where $T(v,\pi)$ is as defined in Section 5.2.3.*

The proof of Lemma 5.2.30 uses standard ideas from the literature [156, 34]. The only modification, essentially, is to show that these algorithms also work for multi-graphs. Let us focus on the specific algorithm proposed in [34, Appendix A]. Given the adjacency list of a graph $G = (V, E)$, it defines gives a procedure $\textsc{Lowest}(v, i)$ that first draws a random rank $E \to [0, 1]$ on each edge (implicitly), then for any input vertex v and an integer $i \leq \deg_G(v)$, returns a vertex u such that (v, u) is the i-th lowest rank edge incident to v. It is proved in [34] that if the procedure is called for a fix vertex v and all indices i with $1 \leq i \leq r$, then the total

running time is $\tilde{O}(r)$. The only difference between the implementation of our algorithm and the one in [34] is that we have multiple copies of a single edge in the original graph. First, we observe that the procedure $\textsc{Lowest}(v, i)$, in addition to returning the neighbor u, can also return the rank of the edge (v, u). (This is explicitly computed by $\textsc{Lowest}$(v, i) in [34].) Now let G' be the multigraph with K copies of each edge of G. Instead of a multigraph, we can assume that we have K copies of same graph G called $G_1, G_2, \ldots, G_K$. Also, for each i, let $\textsc{Lowest}_{G_i}$ be the $\textsc{Lowest}$ procedure corresponding to graph G_i. For each vertex v, we use a balanced binary search tree (BST) that stores the ranks of the lowest incident edge to v in each graph. So at any point during the course of the algorithm, there are at most K different values in the BST of vertex v. Now for the next $\textsc{Lowest}$ query to the multigraph graph G' for vertex v, we can return the minimum edge in the BST of vertex v. Since K is a constant and the any query to a BST is answered in $O(\log n)$ time, the total running time will be the same as [34, Appendix A] within a $O(\log n)$-factor.

5.3 Sublinear Algorithm for Steiner Tree

In this section, we present a sublinear-time algorithm for the Metric Steiner Tree problem. For the sake of completeness, we restate some of the relevant definitions below.

Definition 5.3.1 (Sublinear Metric Steiner Tree)**.** *In the metric Steiner tree problem, we are given a set of points V, a set of terminal points $T \subseteq V$, and query access to an oracle $\mathcal{O}$ to the $|V| \times |V|$ distance matrix of a metric space (V, w), where $\mathcal{O}(u, v)$ returns the weight $w(u, v)$ of the edge (u, v).*

Let $\mathsf{ST}(V, T, w)$ denote the weight of a minimum-weight Steiner tree on instance (V, T, w). Then, the goal is to design an algorithm that estimates $\mathsf{ST}(V, T, w)$ using the fewest possible queries to the distance matrix via the oracle $\mathcal{O}$.

Definition 5.3.2 (Threshold Set Cover)**.** *Given a universe of elements $\mathcal{U}$ and a collection $\mathcal{F}$ of subsets of $\mathcal{U}$, in the Threshold Set Cover problem the goal is to estimate $\mathsf{ThSC}(\mathcal{U}, \mathcal{F}) \coloneqq |\mathcal{U}| - \mathsf{SC}(\mathcal{U}, \mathcal{F})$, where $\mathsf{SC}(\mathcal{U}, \mathcal{F})$ denotes the size of an optimal set cover solution for $(\mathcal{U}, \mathcal{F})$, i.e., the minimal number of sets in $\mathcal{F}$ whose union equals $\mathcal{U}$.*

Following the notation of [69] and for simplicity, in our technical sections, we also refer to this problem as set cover. We prove the following theorems in this section:

Theorem 5.3.3 (Our Algorithm for Threshold Set Cover)**.** *There exists an algorithm that, given a set system $(\mathcal{U}, \mathcal{F})$ with oracle access to its adjacency matrix (also known as membership queries), outputs a multiplicative-additive $(1/2, \varepsilon \cdot |\mathcal{U}|)$-approximation to Threshold Set Cover, in $\widetilde{O}(|\mathcal{F}|^{5/3})$ time, with high probability.*

Theorem 5.3.4 (Sublinear Algorithm for Metric Steiner Tree)**.** *There exists an algorithm that, given an instance of metric Steiner tree denoted by (V, T, w) with oracle access $\mathcal{O}$ to the distance matrix of (V, w), outputs a $(2-\eta)$-estimate of $\mathsf{ST}(V, T, w)$ using $\widetilde{O}(n^{5/3})$ queries to $\mathcal{O}$, where $\eta > 0$ is a universal constant, with high probability.*

5.3.1 Technical Overview

In this section, we provide a brief overview of the technical challenges involved in designing our algorithms. To design a sublinear time algorithm for the Steiner tree problem, we use the framework developed by Chen, Khanna, and Tan [69]. They demonstrated that breaking the 2-approximation barrier for the Steiner tree problem can be reduced to solving an instance of a set cover problem. We refer the reader to Section 4.1 of [69] for details on this reduction.

We denote the variant of the set cover problem as *Threshold Set Cover*. Given a collection of sets $\mathcal{F}$ over a universe of elements $\mathcal{U}$, we aim to estimate the value of $\mathsf{ThSC}(\mathcal{U},\mathcal{F}) = |\mathcal{U}| - \mathsf{SC}(\mathcal{U},\mathcal{F})$, where $\mathsf{SC}(\mathcal{U},\mathcal{F})$ denotes the size of the optimal set cover of the given instance. To achieve our goal of breaking the 2-approximation barrier for the Steiner tree problem, we need to estimate $\mathsf{ThSC}(\mathcal{U},\mathcal{F})$ with a $(\gamma, \varepsilon\cdot|\mathcal{U}|)$ multiplicative-additive error, where γ must be a constant and ε is any (small enough) constant. For this problem, we only have access to a membership oracle of the instance, meaning we can query whether a particular element e is in a particular set S or not. Note that this type of access is generally considered more challenging compared to an adjacency list oracle, where the algorithm can access either the ith element of a set S, or the ith set containing an element e. The reason is that if an element is included in only a constant number of sets, the algorithm is required to spend $\Omega(|\mathcal{F}|)$ queries to find just one set that contains the element. Consequently, we cannot use the results from the literature on sublinear set cover [111, 127] because they all rely on an adjacency list access model.

We will now provide an informal, step-by-step description of our algorithm for Threshold Set Cover, highlighting its differences, innovations, and technical challenges in comparison to the algorithm of Chen, Khanna, and Tan [69]. For simplicity, in this technical overview we assume that $|\mathcal{F}| = \widetilde{\Theta}(|\mathcal{U}|)$, since this represents the worst-case scenario for the Steiner tree problem. However, our formal proof does not depend on this assumption. We let $n = |\mathcal{F}|$.

First step: sparsification of the Threshold Set Cover instance. The goal of this step is to produce a new instance where each element and each set has a low degree—specifically, where each element is in only a few sets, and each set contains only a few elements. This step is standard in designing sublinear algorithms for the set cover problem for different access models, and a slightly different version of it is also used in the algorithm by [69]. Our slight modification of the sparsification step allows us to relax some constraints in the reduction from the Steiner tree problem to Threshold Set Cover, enabling us to achieve the same query complexity for both problems.

Let $x > 0$ be some constant that we optimize later. Consider a set $S \in \mathcal{F}$ and suppose we randomly sample $\widetilde{O}(n^{1-x})$ elements from the universe and query the membership of all the sampled elements in S. If the size of S is at least $\widetilde{\Omega}(n^x)$, we expect to see a large intersection. Conversely, if the size of S is much smaller, we expect to see a small intersection. If a large intersection exists, we can remove the set S and all its elements from the instance. Since this event occurs at most $\widetilde{O}(n^{1-x})$ times, we can account for the removed elements and sets using the additive error in our estimation. Similarly, we can show that all elements belonging to more than $\widetilde{\Omega}(n^x)$ sets can be covered by a random subcollection of sets of size $\widetilde{O}(n^{1-x})$. Therefore, without loss of generality, by spending $\widetilde{O}(n^{2-x})$ time, we can assume that each set contains at most $\widetilde{O}(n^x)$ elements, and each element is included in at most $\widetilde{O}(n^x)$ sets.

Second step: constructing an auxiliary graph H and estimating the size of its maximum matching. Similar to [69], we construct a graph H with a vertex set where each vertex corresponds to an element of $\mathcal{U}$. We connect two vertices if their corresponding elements appear together in at least one set from $\mathcal{F}$. It is important to note that we do not construct H explicitly, as doing so would be computationally expensive and require $\Omega(n^2)$ time. As shown by [69], if the size of the maximum matching of H is large, it is evident that $\mathsf{ThSC}(\mathcal{U}, \mathcal{F})$ is significantly smaller than $|\mathcal{U}|$. Conversely, if the size of the maximum matching of H is close to zero, then $\mathsf{ThSC}(\mathcal{U}, \mathcal{F})$ is also close to zero. This is sufficient for our purposes, as our goal is to obtain a constant-factor approximation. Intuitively, each matching edge in H indicates that there are two elements that can be covered together, which increases the value of $\mathsf{ThSC}(\mathcal{U}, \mathcal{F})$.

There is extensive literature on estimating the size of maximum matching in sublinear time [26, 34, 42, 41, 40, 57, 59, 60, 132, 144, 156, 159, 176], with significant progress made in recent years. For our application, we use the algorithm of Behnezhad [34] to estimate the size of a random greedy maximal matching (RGMM) of the graph. In summary, this algorithm can estimate the size of the RGMM of a graph in $\widetilde{O}(\bar{d})$ time if given access to the adjacency list of the graph, where $\bar{d}$ denotes the average degree of the graph. We can now use this algorithm as a black box:

- The average degree of H is $\widetilde{O}(n^{2x})$, since each element is in $\widetilde{O}(n^x)$ sets and each set contains $\widetilde{O}(n^x)$ elements.
- Each time the algorithm visits a vertex in H (corresponding to an element), we can spend $\widetilde{O}(n^{1+x})$ time to find its adjacency list in H. This involves first querying all sets that include the element, and then making queries between those sets and all elements.

Therefore, we can simulate the algorithm from [34] in $\widetilde{O}(\bar{d} \cdot n^{1+x}) = \widetilde{O}(n^{1+3x})$ time. By balancing this with the sparsification step, which requires $\widetilde{O}(n^{2-x})$ time, we can set $x = 1/4$ to achieve an algorithm with a running time of $\widetilde{O}(n^{7/4})$. This is essentially the running time of the algorithm by Chen, Khanna, and Tan [69].

Third step: using the algorithm of Behnezhad [34] in a white-box manner. To improve the running time of our algorithm, we need to open up the RGMM algorithm from [34] and utilize its properties to apply it more effectively. The RGMM algorithm is a local algorithm that explores the neighborhood of a given vertex to determine whether it is matched. A key observation is that during each exploration, the algorithm requires a random neighbor of the vertex that has not been explored yet. However, in the previous approach, we constructed the entire adjacency list of the vertex, which is redundant and inefficient. Intuitively, we only need to randomly identify one of the vertex's neighbors in each step.

However, the first challenge we encounter is that we cannot select a neighbor uniformly at random. To illustrate this, consider the following example. Suppose that we have five elements $\mathcal{U} = \{e_1, \ldots, e_5\}$ and three sets: $S_1 = \{e_1, e_2, e_3\}$, $S_2 = \{e_1, e_2, e_4\}$, and $S_3 = \{e_1, e_2, e_5\}$. Suppose that we want to find a random neighbor of e_1 in H. If we first find all sets that include e_1 and then query between those sets and all elements uniformly at random until we find an edge in H, we are likely to see the edge (e_1, e_2) because it appears in all sets. Consequently, the algorithm has a bias towards finding neighbors that appear in more sets with the element.

To overcome this challenge, rather than defining an auxiliary simple graph H, we define an auxiliary multigraph H. In this multigraph, if two elements appear in the same set multiple times, we add an edge for

each of those occurrences. Note that the average degree of H remains at most $\widetilde{O}(n^{2x})$. However, the algorithm and analysis for RGMM from [34] are designed for simple graphs. We extend these results to multigraphs and show that we can estimate the size of RGMM for a multigraph, given access to its adjacency list. This extension may be of independent interest and could be useful for tackling other problems in sublinear time. To establish this, we build on the exquisite approach first introduced by Yoshida, Yamamoto, and Ito [176] and further explored in various settings [34, 43, 44]. We employ techniques such as the analysis of the round-complexity of maximal independent sets [90], double-counting arguments to bound the average complexity of RGMM on multigraphs, and others; we encourage the reader to refer to Section 5.3.3 for further details.

Now, suppose that for each vertex the RGMM algorithm explores in H, we first query all sets to identify those that include the corresponding element. Since the algorithm explores at most $\widetilde{O}(\bar{d})$ vertices in H, this step will cost at most $\widetilde{O}(\bar{d} \cdot n) = \widetilde{O}(n^{1+2x})$ in total. Let v be a vertex that the RGMM algorithm is exploring at the moment. Define $\mathcal{S}_v$ to be the collection of sets that include element v. Now, if we query uniformly at random between all elements and the collection $\mathcal{S}_v$, each incident edge of v in the multigraph H has an equal probability of being sampled, which resolves the first challenge. For now, assume that the degree of all vertices in H is $\bar{d}$. Since $|S_v| = \widetilde{O}(n^x)$ and there are $\widetilde{O}(n)$ elements in total, we expect to find an element in one of the sets of S_v every $\widetilde{O}(n^{1+x}/\bar{d})$ queries. Thus, to identify a random neighbor of a vertex in H, we need to spend $\widetilde{O}(n^{1+x}/\bar{d})$ time. The RGMM algorithm queries for a random neighbor of a vertex $\widetilde{O}(\bar{d})$ times, since the exploration size is $\widetilde{O}(\bar{d})$; therefore, the total cost is $\widetilde{O}(n^{1+x})$. Combining this with the cost of sparsification, the total cost of the algorithm is $\widetilde{O}(\max(n^{2-x}, n^{1+2x}))$, which is $\widetilde{O}(n^{5/3})$ if we set $x = 1/3$.

The second challenge arises because the RGMM algorithm may predominantly visit vertices with a very low degree in H. For such vertices, finding a random neighbor can be much more time-consuming. Generally, if a vertex v in H has degree $\deg_H(v)$, then each time the algorithm finds a random neighbor of v, it needs to spend $\widetilde{O}(n^{1+x}/\deg_H(v))$ time. Therefore, if the algorithm frequently encounters vertices with constant degree, each query to find a random neighbor can cost $\widetilde{O}(n^{1+x})$. With the exploration size being $\widetilde{O}(\bar{d})$, this can significantly increase the query complexity of the algorithm. As a property of the local RGMM algorithm, we demonstrate that each vertex is visited by the algorithm in proportion to its degree in H. More formally, we prove that RGMM requires $\widetilde{O}(\deg_H(v)/n)$ neighbors of v on average, for a uniformly random permutation of edges. Thus, the degree-dependent factors cancel each other out, and the average cost of this part can be upper-bounded by $\widetilde{O}(n^{1+x})$, which is enough for us to get the $\widetilde{O}(n^{5/3})$ running time for the Threshold Set Cover.

5.3.2 Sublinear Algorithm for Set Cover

In this section, we formalize and analyze our algorithm for the set cover variant described above. Throughout this section, we assume that $\mathcal{U}$ denotes the universe and $\mathcal{F}$ denotes the collection of sets. We will slightly abuse notation by letting $k = |\mathcal{U}|$ and $n = |\mathcal{F}|$. Without loss of generality, we can assume that $n \geq k$.[4] We use $\mathsf{SC}(\mathcal{U}, \mathcal{F})$ for the size of the minimum set cover of the input instance. Our goal is to design an algorithm that estimates the value of $\chi = k - \mathsf{SC}(\mathcal{U}, \mathcal{F})$ with at most εk additive error and a constant multiplicative factor. For $\gamma_1 \in (0, 1]$ and $\gamma_2 > 0$, we say that $\widetilde{\chi}$ is a multiplicative-additive (γ_1, γ_2)-approximation of the value χ if $\gamma_1 \chi - \gamma_2 \leq \widetilde{\chi} \leq \chi$. Similar to the reduction from the Steiner tree problem to set cover in [69],

[4]For the sake of this problem, for each element in the universe we can add a set that only contains the element. The same assumption is also made in [69].

we need to estimate $k - \mathsf{SC}(\mathcal{U}, \mathcal{F}_{\neq 2})$ where $\mathcal{F}_{\neq 2}$ denotes the collection of all sets in $\mathcal{F}$ except those of size exactly 2. For simplicity, we focus on estimating χ, and in the final step of this section, we will explain how to handle sets of size 2 in our algorithm. For our application, we need γ_1 to be constant and $\gamma_2 = \varepsilon k$ where ε is a small fixed constant.

A High-Level Description of the Algorithm: Our algorithm for estimating the value of χ is formalized in Algorithm 29. Apart from the collection of sets $\mathcal{F}$ and the universe of elements $\mathcal{U}$, the algorithm runs with two parameters, α and β, both of which can be determined based on the values of x and y, which we optimize in the final step of the analysis. The algorithm has three phases: 1) sparsification of the sets, 2) sparsification of the elements, and 3) estimating the size of a maximum matching of the auxiliary graph H (Definition 5.3.5). In the following, we first describe each component in words before moving on to the formal proofs.

(Step 1) Set Sparsification: This component of the algorithm is formalized in Algorithm 30. The algorithm maintains a collection of sets $\hat{\mathcal{F}}$ and a universe of elements $\hat{\mathcal{U}}$ that are initially equal to $\mathcal{F}$ and $\mathcal{U}$, respectively. We iterate over all sets in $\mathcal{F}$ one by one and for each set S, we sample $r_1 = |\hat{\mathcal{U}}|/\alpha$ random elements from $\hat{\mathcal{U}}$. Intuitively, if $|S \cap \hat{\mathcal{U}}| \geq \widetilde{\Omega}(\alpha)$, we expect to have an element of S in the r_1 random sampled elements. Having this in mind, if there is a large enough intersection ($\Omega(\log n)$) between the sampled elements and S, we remove set S and all its elements from $\hat{\mathcal{F}}$ and $\hat{\mathcal{U}}$, respectively (we add this set to our solution). Therefore, after the execution of the algorithm, each remaining set in $\hat{\mathcal{F}}$ has at most $\widetilde{O}(\alpha)$ elements. On the other hand, if $|S \cap \hat{\mathcal{U}}|$ is smaller than α, we expect to see a small intersection with the r_1 sampled elements. Consequently, the number of times the algorithm removes a set and its elements from $\hat{\mathcal{F}}$ and $\hat{\mathcal{U}}$ is at most $k/\alpha = o(k)$, which can be accounted for by the additive error in the estimation. Also, if at any point the size of the maintained universe becomes smaller than some threshold (in the algorithm the value of the threshold is $\widetilde{\Theta}(\alpha)$, the algorithm stops processing the rest of the sets (Algorithm 30) since $\hat{\mathcal{U}} = \widetilde{O}(\alpha)$. This step differs from the algorithm in [69] because we sequentially sparsify the sets, whereas their approach is non-adaptive.

(Step 2) Sparsification of Elements: This part of the algorithm is formalized in Algorithm 31. Similar to the previous step, we want to sparsify our instance such that each element in the remaining instance appears in at most $\widetilde{O}(\beta)$ sets. Let $\hat{\mathcal{U}}$ and $\hat{\mathcal{F}}$ be the output of Algorithm 30. We sample $r_2 = |\hat{\mathcal{U}}|/\beta$ random sets from the collection $\hat{\mathcal{F}}$. With the same intuition as the previous step, if some element is in at least $\widetilde{\Omega}(\beta)$ sets of $\hat{\mathcal{F}}$, we expect to see it in many sampled sets. We partition the elements of $\hat{\mathcal{U}}$ into $\mathcal{U}_{low}$ and $\mathcal{U}_{high}$, depending on whether their intersection with the randomly sampled elements is smaller than a given threshold or not. With high probability, each element in $\mathcal{U}_{low}$ appears in at most $\widetilde{O}(\beta)$ sets of $\hat{\mathcal{F}}$, and each element in $\mathcal{U}_{high}$ appears in at least $\widetilde{\Omega}(\beta)$ sets of $\hat{\mathcal{F}}$. This suffices to show that any random subset of $\hat{\mathcal{F}}$ of size $\varepsilon k/2$ can cover all elements of $\mathcal{U}_{high}$, which can be included in the additive error of the estimation. This step is similar to the approach used in [69].

After steps 1 and 2 of the algorithm, we have the property that each set in the remaining instance has at most $\widetilde{O}(\alpha)$ elements, and each element in the remaining instance is in at most $\widetilde{O}(\beta)$ sets.

(Step 3) Estimating the Maximum Matching of Auxiliary Graph H: We construct an auxiliary multigraph H with vertex set $\mathcal{U}_{low}$. For each set $S \in \hat{\mathcal{F}}$, we add an edge in H between every two elements

of S. Note that we do not explicitly construct the multigraph H because doing so would require $\Omega(nk)$ time, which is not feasible. We now estimate the size of the maximum matching in H to produce our final estimate. Intuitively, if χ is very small (close to zero), meaning that nearly k sets are required to cover the universe, then the maximum matching in H will also be small. To see this, note that a matching edge implies that its two endpoints can be covered by the same set. Conversely, if χ is large (almost equal to k), then H will have a large matching because each set in the set cover solution covers many new elements, which can be almost paired up in H. Thus, obtaining a constant approximation for the size of the maximum matching in H is sufficient to achieve our goal, and we do this by estimating the size of a random greedy maximal matching in H. To that end, we modify and analyze the algorithm from [34] for multigraphs and adapt it to our access model for the multigraph H, as discussed in detail in Section 5.3.3. The algorithm in [69] constructs a similar graph, but it is not a multigraph in the sense that, for any two elements that appear together in multiple sets, only a single edge is added between them in H.

Definition 5.3.5 (Auxiliary Multigraph H)**.** *Let $\hat{\mathcal{F}}$ and $\mathcal{U}_{low}$ be as defined in Algorithm 29, respectively, of Algorithm 29. We construct an auxiliary graph H with vertex set $\mathcal{U}_{low}$ such that for each set S in $\hat{\mathcal{F}}$ and each two different elements $e, e' \in \mathcal{U}_{low}$, we add an edge (e, e') to H. Note that H is a multigraph: multiple sets may contain both elements e and e', and we add an edge for each of these sets.*

Algorithm 29: Sublinear Time Algorithm for Set Cover

Input: Collection of sets $\mathcal{F}$ and universe of elements $\mathcal{U}$.
Parameter: $\alpha \leftarrow n^x, \quad \beta \leftarrow 10 \max(k/n^{1-y}, 1) \cdot n \log(n)/k$.
Let $\hat{\mathcal{F}}$ and $\hat{\mathcal{U}}$ be the output of Algorithm 30 with input $\mathcal{F}$, $\mathcal{U}$ and α.
Let $\mathcal{U}_{low}$ and $\mathcal{U}_{high}$ be the output of Algorithm 31 with input $\hat{\mathcal{F}}$, $\hat{\mathcal{U}}$, and β.
Let H be the auxiliary multigraph defined in Definition 5.3.5. ▷ We do not build H explicitly.
Let $\widetilde{\mu}$ be the estimate of $\mathbf{E}_\pi |\mathrm{GMM}(H, \pi)|$ using the algorithm in Section 5.3.3.
Let $\widetilde{\chi} \leftarrow \widetilde{\mu} + |\mathcal{U} \setminus \mathcal{U}_{low}| - \varepsilon k/2$.
return $\widetilde{\chi}$.

Algorithm 30: Sparsification of Sets

Input: Collection of sets $\mathcal{F}$, universe of elements $\mathcal{U}$, and parameter α that controls the sparsification ratio.
$\hat{\mathcal{F}} \leftarrow \mathcal{F}, \quad \hat{\mathcal{U}} \leftarrow \mathcal{U}, \quad c \leftarrow 0.$ ▷ We use c only for the analysis.
for $S \in \mathcal{F}$ **do**
  if $|\hat{\mathcal{U}}| < 10\alpha \log n$ **then**
    break
  $r_1 \leftarrow |\hat{\mathcal{U}}|/\alpha$.
  Let $e_1, e_2, \ldots, e_{r_1}$ be r_1 random elements of $\hat{\mathcal{U}}$.
  Make queries between S and elements $e_1, \ldots, e_{r_1}$.
  if $|\{e_1, \ldots, e_{r_1}\} \cap S| \geq 10 \log n$ **then**
    $\hat{\mathcal{F}} \leftarrow \hat{\mathcal{F}} \setminus \{S\}$.
    $c \leftarrow c + 1$.
    for $e \in \hat{\mathcal{U}}$ **do**
      if $e \in S$ **then**
        $\hat{\mathcal{U}} \leftarrow \hat{\mathcal{U}} \setminus \{e\}$.
return $\hat{\mathcal{F}}$,$\hat{\mathcal{U}}$

Algorithm 31: Sparsification of Elements

Input: Collection of sets $\hat{\mathcal{F}}$, universe of elements $\hat{\mathcal{U}}$, and parameter β that controls the sparsification ratio.
$r_2 \leftarrow |\hat{\mathcal{U}}|/\beta$.
if $r_2 < 20 \log n/\varepsilon$ **then**
  $\mathcal{U}_{low} \leftarrow \hat{\mathcal{U}}, \quad \mathcal{U}_{high} \leftarrow \emptyset$.
  return $\mathcal{U}_{low}, \mathcal{U}_{high}$.
Let $\{S_1, S_2, \ldots, S_{r_2}\}$ be r_2 random sets from $\hat{\mathcal{F}}$.
Make queries between all elements in $\hat{\mathcal{U}}$ and sets in $\{S_1, S_2, \ldots, S_{r_2}\}$.
Let $\mathcal{U}_{low}$ be the elements that appeared in at most $20 \log n/\varepsilon$ many sets.
$\mathcal{U}_{high} \leftarrow \hat{\mathcal{U}} \setminus \mathcal{U}_{low}$.
return $\mathcal{U}_{low}, \mathcal{U}_{high}$.

Proof of Correctness

In this section, we prove the correctness of Algorithm 29.

Claim 5.3.6. *For any set S such that the condition of Algorithm 30 holds during the execution of Algorithm 30, with high probability it holds that $|S \cap \hat{\mathcal{U}}| \geq \alpha$, where $\hat{\mathcal{U}}$ denotes the universe that Algorithm 30 maintains at the time it processes S.*

Proof. Suppose that S is a set such that $|S \cap \hat{\mathcal{U}}| < \alpha$ at the time that Algorithm 30 processes this set. Let X_i be the random variable that indicates $e_i \in S$. Thus, we have $\Pr[X_i] \leq |S \cap \hat{\mathcal{U}}|/|\hat{\mathcal{U}}|$. Let $X = \sum_{i=1}^{r_1} X_i$. By linearity of expectation, we have $\mathbf{E}[X] < r_1\alpha/|\hat{\mathcal{U}}| = 1$. Also, note that X_i's are negatively associated

random variables. Let $\lambda = (9 \log n)/\mathbf{E}[X]$. Therefore, using the Chernoff bound for negatively associated random variables (Proposition 2.3.5) we have

$$\begin{aligned}
\Pr[X \geq (1+\lambda)\,\mathbf{E}[X]] &\leq \left(\frac{e^{\lambda}}{(1+\lambda)^{1+\lambda}}\right)^{\mathbf{E}[X]} \\
&\leq \left(\frac{e^{\lambda}}{\lambda^{\lambda}}\right)^{\mathbf{E}[X]} && \text{(Since } \lambda > 1\text{)} \\
&= \left(\frac{e}{\lambda}\right)^{9 \log n} && \text{(Since } \lambda = (9 \log n)/\mathbf{E}[X]\text{)} \\
&\leq \frac{1}{n^9} && \text{(Since } \lambda > e^2\text{)}
\end{aligned}$$

which implies that with probability of at least $1 - n^{-9}$,

$$X < (1+\lambda)\,\mathbf{E}[X] = \mathbf{E}[X] + 9 \log n < 10 \log n.$$

Since we have n sets, using a union bound, with a probability at least $1 - n^{-8}$, for any set such that the condition of Algorithm 30 holds, we have $|S \cap \hat{\mathcal{U}}| \geq \alpha$. □

Claim 5.3.7. *Let c be the variable used in Algorithm 30. With high probability, we have $c \leq k/\alpha$.*

Proof. By Claim 5.3.6, for every set S such that the condition on Algorithm 30 of Algorithm 30 holds, with high probability we have that $|S \cap \hat{\mathcal{U}}| \geq \alpha$. Hence, each time the algorithm increases c, the size of $\hat{\mathcal{U}}$ decreases by α. Therefore, the total number of times the algorithm increases c is upper-bounded by k/α. □

Claim 5.3.8. *Let $\hat{\mathcal{F}}$ and $\hat{\mathcal{U}}$ be the output of Algorithm 30. Then, each set $S \in \hat{\mathcal{F}}$ has at most $20\alpha \log n$ elements in $\hat{\mathcal{U}}$ with high probability.*

Proof. First, note that if the algorithm stops because of the condition of Algorithm 30 and does not process S, it holds that $|\hat{\mathcal{U}}| < 10\alpha \log n$ and the claim trivially holds.

Let S be a set such that at the time that Algorithm 30 processes S, we have $|S \cap \hat{\mathcal{U}}| \geq 20\alpha \log n$. Similar to the proof of Claim 5.3.6, let X_i be the random variable that indicates $e_i \in S$ and $X = \sum_{i=1}^{r_1} X_i$. Hence, $\mathbf{E}[X] \geq 20 r_1 \alpha \log n/|\hat{\mathcal{U}}| = 20 \log n$. Since X_i's are negatively associated random variables, using Chernoff bound (Proposition 2.3.5) for $\lambda = (9 \log n)/\mathbf{E}[X]$, we have

$$\begin{aligned}
\Pr[X \leq (1-\lambda)\,\mathbf{E}[X]] &\leq \left(\frac{e^{\lambda}}{(1+\lambda)^{1+\lambda}}\right)^{\mathbf{E}[X]} \\
&\leq \left(\frac{e^{\lambda}}{\lambda^{\lambda}}\right)^{\mathbf{E}[X]} && \text{(Since } \lambda > 1\text{)} \\
&= \left(\frac{e}{\lambda}\right)^{9 \log n} && \text{(Since } \lambda = (9 \log n)/\mathbf{E}[X]\text{)} \\
&\leq \frac{1}{n^9} && \text{(Since } \lambda > e^2\text{)}
\end{aligned}$$

which implies that with a probability of at least $1 - n^{-9}$,

$$X > (1-\lambda)\,\mathbf{E}[X] = \mathbf{E}[X] - 9 \log n \geq 20 \log n - 9 \log n > 10 \log n.$$

Therefore, the condition on Algorithm 30 of Algorithm 30 must hold for all such sets with a probability of at least $1 - n^{-8}$ using union bound. □

Lemma 5.3.9 (Sets Sparsification Guarantee)**.** *Let $\hat{\mathcal{F}}$ and $\mathcal{U}_{low}$ be outputs of Algorithm 30 and Algorithm 31. Also, let $S \in \hat{\mathcal{F}}$. Then, with high probability, S contains at most $\widetilde{O}(\alpha)$ elements of $\mathcal{U}_{low}$.*

Proof. Note that $\mathcal{U}_{low} \subseteq \hat{\mathcal{U}}$ where $\hat{\mathcal{U}}$ is the output of Algorithm 30. Also, by Claim 5.3.8, we have $|S \cap \hat{\mathcal{U}}| \leq 20\alpha \log n$. Thus, with high probability we have $|S \cap \mathcal{U}_{low}| \leq 20\alpha \log n = \widetilde{O}(\alpha)$. □

Lemma 5.3.10 (Elements Sparsification Guarantee)**.** *Let $\mathcal{U}_{low}$ be as output by Algorithm 31 and let $e \in \mathcal{U}_{low}$. Then, with high probability, there are at most $\widetilde{O}(\beta)$ sets of $\hat{\mathcal{F}}$ that contain e.*

Proof. Let e be an element of $\hat{\mathcal{U}}$, where $\hat{\mathcal{U}}$ is the output of Algorithm 30. We show that if at least $40\beta \log n/\varepsilon$ sets in $\hat{\mathcal{F}}$ contain e, then $e \in \mathcal{U}_{high}$ in the output of Algorithm 31 with high probability.

Let X_i be an indicator variable for S_i containing e. Thus, we have $\mathbf{E}[X_i] \geq 40\beta \log n/(\varepsilon n)$. Define $X = \sum_{i=1}^{r_2} X_i$. Hence, $\mathbf{E}[X] \geq r_2 \cdot 40\beta \log n/(\varepsilon n) = 40 \log n/\varepsilon$. Since X_i's are negatively associated random variables, using Chernoff bound (Proposition 2.3.5) for $\lambda = (9 \log n)/\mathbf{E}[X]$, we have

$$\Pr[X \leq (1-\lambda)\,\mathbf{E}[X]] \leq \frac{1}{n^9} \qquad \text{(Similar to the proof of Claim 5.3.8)}$$

which implies that $X > 20 \log n$ with a probability of at most $1 - n^{-9}$, since

$$X > (1-\lambda)\,\mathbf{E}[X] = \mathbf{E}[X] - 9 \log n \geq 40 \log n/\varepsilon - 9 \log n > 20 \log n/\varepsilon.$$

Using a union bound for all elements in $\hat{\mathcal{U}}$ that are in at least $40\beta \log n/\varepsilon$ sets of $\hat{\mathcal{F}}$, with high probability all of them are going to be included in $\mathcal{U}_{high}$. As a result, for each $e \in \mathcal{U}_{low}$, e is in at most $\widetilde{O}(\beta)$ sets of $\hat{\mathcal{F}}$. □

Claim 5.3.11. *Let $\mathcal{F}'$ be a random collection of $\varepsilon k/5$ sets of $\hat{\mathcal{F}}$. Then, with high probability, every element in $\mathcal{U}_{high}$ is in one of the sets of $\mathcal{F}'$.*

Proof. Let $e \in \hat{\mathcal{U}}$ be such that at most $15\beta \log n/\varepsilon$ sets of $\hat{\mathcal{F}}$ contain e. Similar to the proof of Lemma 5.3.10, let X be a random variable that denotes the number of sets S_i (for $i = 1, 2, ..., r_2$) that contain e. Using a Chernoff bound, we can prove that with a probability of at least $1 - n^{-2}$, for all such elements e, we have $X < 20 \log n/\varepsilon$. Therefore, all elements in $\mathcal{U}_{high}$ are in at least $15\beta \log n/\varepsilon$ sets of $\hat{\mathcal{F}}$.

Consequently, when $|\mathcal{F}'| \geq \varepsilon k/5$, the expected number of sets in $\mathcal{F}'$ that cover element $e \in \mathcal{U}_{high}$ is at least

$$\frac{15\beta \log n}{\varepsilon} \cdot \frac{1}{n} \cdot \frac{\varepsilon k}{5} \geq \frac{15n \log n}{\varepsilon k} \cdot \frac{1}{n} \cdot \frac{\varepsilon k}{5} = 3 \log n$$

where we used that $\beta \geq n/k$ (see Algorithm 29 in Algorithm 29). Using Chernoff bounds again, we expect all these elements to be covered by at least one set with high probability, which concludes the proof. □

Claim 5.3.12. *Let $\mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}})$ be the optimal set cover size for the universe of elements $\mathcal{U}_{low}$ and the collection of sets $\hat{\mathcal{F}}$ which are the outputs of Algorithm 31 and Algorithm 30, respectively. Let $\widetilde{\mu}$ be the output of Lemma 5.3.17 (i.e., a size estimate of a random greedy maximal matching of H) with $\varepsilon k/2$ additive error. Then, it holds that*

$$\frac{1}{2}\left(|\mathcal{U}_{low}| - \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}})\right) - \frac{\varepsilon k}{2} \leq \widetilde{\mu} \leq |\mathcal{U}_{low}| - \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}).$$

Proof. Let M be any maximal matching of H. First, we show that

$$\frac{1}{2}\left(|\mathcal{U}_{low}| - \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}})\right) \leq |M| \leq |\mathcal{U}_{low}| - \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}).$$

If for each edge in M, we take the corresponding set, and for the rest of the elements we take a set that only covers that element, we will have covered all elements of $\mathcal{U}_{low}$ with at most $|\mathcal{U}_{low}| - |M|$ sets, which implies that $|M| \leq |\mathcal{U}_{low}| - \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}})$. On the other hand, no set can cover two elements that are unmatched by M at the same time. Thus, we have $|\mathcal{U}_{low}| - 2|M| \leq \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}})$.

Since the output of *Lemma* 5.3.17 can be smaller than the minimum-sized maximal matching by an additive factor of at most $\varepsilon k/2$, we get the claim. □

Claim 5.3.13. *Conditioning on the high-probability events of Claim 5.3.7 and Claim 5.3.11, it holds that* $\mathsf{SC}(\mathcal{U}, \mathcal{F}) \leq \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}) + \varepsilon k/2$.

Proof. All elements of $\mathcal{U} \setminus (\mathcal{U}_{low} \cup \mathcal{U}_{high})$ can be covered using the c sets that are deleted in Algorithm 30. Moreover, by Claim 5.3.7, we have $c < o(k)$ since $\alpha > \omega(1)$. On the other hand, the elements of $\mathcal{U}_{high}$ can be covered using $\varepsilon k/5$ sets, as shown in Claim 5.3.11. Finally, the elements of $\mathcal{U}_{low}$ can be covered using $\mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}})$ sets. Therefore, the union of these three collections covers all elements, and the size of this solution is at most $\mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}) + \varepsilon k/2$, which completes the proof. □

Lemma 5.3.14. *Conditioning on the high-probability events of Claim 5.3.7 and Claim 5.3.11, it holds that* $\chi/2 - \varepsilon k \leq \widetilde{\chi} \leq \chi$. *(Recall that* $\chi = k - \mathsf{SC}(\mathcal{U}, \mathcal{F})$ *and* $\widetilde{\chi}$ *is the output of Algorithm 29, defined on its Algorithm 29.)*

Proof. We have that

$$\begin{aligned}
\widetilde{\chi} &= \widetilde{\mu} + |\mathcal{U} \setminus \mathcal{U}_{low}| - \frac{\varepsilon k}{2} && \text{(Output of Algorithm 29)} \\
&\leq |\mathcal{U}_{low}| - \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}) + |\mathcal{U} \setminus \mathcal{U}_{low}| - \frac{\varepsilon k}{2} && \text{(By Claim 5.3.12)} \\
&= |\mathcal{U}| - \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}) - \frac{\varepsilon k}{2} \\
&\leq |\mathcal{U}| - \mathsf{SC}(\mathcal{U}, \mathcal{F}) && \text{(By Claim 5.3.13)} \\
&= \chi.
\end{aligned}$$

On the other hand,

$$\begin{aligned}
2(\widetilde{\chi} + \varepsilon k) &= 2\left(\widetilde{\mu} + |\mathcal{U} \setminus \mathcal{U}_{low}| + \frac{\varepsilon k}{2}\right) \\
&\geq 2\left(\frac{|\mathcal{U}_{low}|}{2} - \frac{\mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}})}{2} + |\mathcal{U} \setminus \mathcal{U}_{low}|\right) && \text{(By Claim 5.3.12)} \\
&\geq |\mathcal{U}| - \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}) \\
&= |\mathcal{U}| - \mathsf{SC}(\mathcal{U}_{low}, \mathcal{F}) \\
&\geq |\mathcal{U}| - \mathsf{SC}(\mathcal{U}, \mathcal{F}) \\
&= \chi,
\end{aligned}$$

which concludes the proof. □

Time Complexity

Claim 5.3.15. *Algorithm 30 runs in $O(nk/\alpha)$ time with high probability.*

Proof. First note that Algorithm 30 samples at most k/α elements for each set in $\mathcal{F}$ (Algorithm 30). So if we ignore the block of if-condition in Algorithm 30, the runtime is at most $O(nk/\alpha)$. Further, by Claim 5.3.7, the condition on Algorithm 30 holds at most k/α times with high probability. Since each time the algorithm enters the if-condition it spends $O(k)$ time to iterate over all elements, the total running time is at most $O(k^2/\alpha + nk/\alpha) = O(nk/\alpha)$. □

Claim 5.3.16. *Algorithm 31 runs in $O(k^2/\beta)$ time.*

Proof. The algorithm samples $r_2 = O(k/\beta)$ sets and for each of them makes a membership query between the set and all elements. Hence, the total running time is upper-bounded by $O(k^2/\beta)$. □

We will prove the following lemma about the complexity of RGMM in the next section. However, to complete the time analysis, we state the lemma here.

Lemma 5.3.17. *Let H be the multigraph defined in Definition 5.3.5. There exists an algorithm with an expected running time of $\widetilde{O}_\varepsilon(k\beta + \alpha\beta n)$ that estimates the value of $E_\pi|\mathrm{GMM}(H,\pi)|$ with εk additive error with high probability.*

Lemma 5.3.18. *The total running time of the algorithm is $\widetilde{O}(nk/\alpha + k^2/\beta + n\alpha\beta)$ with high probability.*

Proof. The algorithm runs in expected time $\widetilde{O}(nk/\alpha + n\alpha\beta)$ as shown by Claim 5.3.15, Claim 5.3.16, Lemma 5.3.17, and the fact that $k \leq n$. To ensure a high-probability bound on the running time, we execute $\Theta(\log n)$ instances of the algorithm in parallel and use the estimate from the first instance that finishes. Since the expected running time is $\widetilde{O}(kn/\alpha + k^2/\beta + n\alpha\beta)$, the first instance is likely to terminate within $\widetilde{O}(nk/\alpha + k^2/\beta + n\alpha\beta)$ time with probability $1 - 1/\operatorname{poly}(n)$. (We remark that since our approximation guarantees hold with high probability, they also hold with high probability for each of the instances.) □

Putting Everything Together

We combine our ideas to obtain our final theorem for estimating $\chi = k - \mathsf{SC}(\mathcal{U},\mathcal{F})$. Then, we extend our analysis and make slight modifications to the algorithm to estimate $k - \mathsf{SC}(\mathcal{U},\mathcal{F}_{\neq 2})$ which is crucial for designing a sublinear algorithm for the Steiner tree problem.

Theorem 5.3.3 (Our Algorithm for Threshold Set Cover)**.** *There exists an algorithm that, given a set system $(\mathcal{U},\mathcal{F})$ with oracle access to its adjacency matrix (also known as membership queries), outputs a multiplicative-additive $(1/2, \varepsilon\cdot|\mathcal{U}|)$-approximation to Threshold Set Cover, in $\widetilde{O}(|\mathcal{F}|^{5/3})$ time, with high probability.*

Proof. First, if $k \leq O(n^{2/3})$, then we can easily query between all elements and sets and compute the estimate in $\widetilde{O}(n^{5/3})$ time since all the steps of the algorithm such as finding the maximal matching can be run in linear time with respect to the size of the input. Now suppose that $k > n^{2/3}$. We use Algorithm 29 to produce

the estimate $\widetilde{\chi}$, setting $x = 1/3$ and $y = 1/3$. By Lemma 5.3.14, we have that $\widetilde{\chi}$ is a multiplicative-additive $(1/2, \varepsilon k)$-approximation to the value of χ.

Moreover, we have $\alpha = n^{1/3}$ and $\beta = 10\max(k/n^{1-y}, 1) \cdot n\log(n)/k = \widetilde{O}(n^{1/3})$. By Lemma 5.3.18, the total runtime of the algorithm is upper-bounded by $\widetilde{O}(kn/\alpha + k^2/\beta + n\alpha\beta) = \widetilde{O}(n^{5/3})$ because of the assumption that $k \leq n$. □

Now we show how we can estimate $k - \mathsf{SC}(\mathcal{U}, \mathcal{F}_{\neq 2})$ with a slight modification. The only change that is needed in our algorithm is to remove edges of H that are produced by sets of size 2 in $\hat{\mathcal{F}}$. Hence, when the algorithm estimates $\mathbf{E}_\pi[\mathrm{GMM}(H, \pi)]$ using the vertex oracle, the oracles must not use those edges in the exploration as they do not exist in graph H. Thus, in the implementation of Lemma 5.3.17, when the algorithm queries pairs (u, S) where $u \in \mathcal{U}_{low}$ and $S \in \mathcal{F}_v$ to find neighbors of v in H, if $u \in S$, the algorithm starts querying between all elements of $\mathcal{U} \setminus v$ and S until it finds another element in S. If the algorithm finds another element of S, then it accepts the edge (v, u) as a valid edge, and otherwise it continues the random search. So each time the algorithm finds such pair (u, S), it invokes the above procedure to validate the edge. In the next theorem, we demonstrate how to bound the running time of the modified algorithm and discuss its approximation ratio.

Theorem 5.3.19. *There exists an algorithm that outputs a multiplicative-additive $(1/2, \varepsilon k)$-approximation of value of $\chi = k - \mathsf{SC}(\mathcal{U}, \mathcal{F}_{\neq 2})$ in $\widetilde{O}(|\mathcal{F}|^{5/3})$ time with high probability.*

Proof. We need to show that the new estimation is a multiplicative-additive $(1/2, \varepsilon k)$-approximation of $k - \mathsf{SC}(\mathcal{U}, \mathcal{F}_{\neq 2})$. We follow the same approach as proof of Lemma 5.3.14. To follow the same steps, we need analogous claims similar to Claim 5.3.12 and Claim 5.3.13. The exact same proof of Claim 5.3.12 also works to provide a bound on $\mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}_{\neq 2})$. More specifically, we have

$$\frac{1}{2}\left(|\mathcal{U}_{low}| - \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}_{\neq 2})\right) - \frac{\varepsilon k}{2} \leq \widetilde{\mu} \leq |\mathcal{U}_{low}| - \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}_{\neq 2}).$$

Further, to show that $\mathsf{SC}(\mathcal{U}, \mathcal{F}_{\neq 2}) \leq \mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}_{\neq 2}) + \varepsilon k/2$, note that we have $c = o(k)$ (see Algorithm 30 for definition of c) which implies that $\mathcal{U} \setminus (\mathcal{U}_{low} \cup \mathcal{U}_{high})$ can be covered using $o(k)$ sets of size larger than 2 (all sets that are removed in Algorithm 30 have size larger than 2). Additionally, elements of $\mathcal{U}_{high}$ can be covered using $\varepsilon k/5$ sets of $\hat{\mathcal{F}}$ according to Claim 5.3.11. Thus, we can cover elements of $\mathcal{U}_{high}$ using $2\varepsilon k/5$ sets of $\hat{\mathcal{F}}_{\neq 2}$ since we can replace each set of size 2 with two sets of size 1 that cover a single element. Finally, elements of $\mathcal{U}_{low}$ can be covered using $\mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}_{\neq 2})$ sets. The union of all these sets is a valid solution for $\mathsf{SC}(\mathcal{U}, \hat{\mathcal{F}}_{\neq 2})$ which has a size of at most $\mathsf{SC}(\mathcal{U}_{low}, \hat{\mathcal{F}}_{\neq 2}) + \varepsilon k/2$. Therefore, the error of our estimation is $(1/2, \varepsilon k)$.

Now it remains to bound the running time of the modified algorithm. We prove that the same bound on the running time as in Lemma 5.3.17 holds. Now consider a vertex v that is corresponding to an element in $\mathcal{U}_{low}$. Let $\hat{\mathcal{F}}_v$ be the collection of sets that include v. Also, let $\hat{\mathcal{F}}'_v = \{S | S \in \hat{\mathcal{F}}_v \text{ and } |S| = 2\}$. Let $r = |\hat{\mathcal{F}}_v|$ and $\hat{\mathcal{F}}_v = \{S_1, \ldots, S_r\}$. Also, let $\tau_i = |S_i \cap (\mathcal{U}_{low} \setminus v)|$. Hence, in Lemma 5.3.17, when the algorithm queries a pair (u, S) and it returns $u \in S$, the probability that $S = S_i$ is $\tau_i / \sum_{i=1}^{r} \tau_i$. For this set, using a Chernoff bound, the algorithm needs to spend at most $\widetilde{O}(k/\tau_i)$ time to see another element of S_i or explore all elements

of $\mathcal{U}$. Therefore, the expected additional time the algorithm needs to spend compared to Lemma 5.3.17 is

$$\sum_{i}^{r}\left(\frac{\tau_i}{\sum_{i=1}^{r}\tau_i}\right)\cdot\widetilde{O}\left(\frac{k}{\tau_i}\right)=\widetilde{O}\left(\sum_{i}^{r}\frac{k}{\sum_{i=1}^{r}\tau_i}\right)\leq\widetilde{O}\left(\frac{rk}{\deg_H(v)}\right),$$

where the last inequality follows by the fact that $\deg_H(v)\leq\sum_{i=1}^{r}\tau_i$ since $\deg_H(v)=(\sum_{i=1}^{r}\tau_i)-|\{S|S\in\hat{\mathcal{F}}_v \text{ and } |S|=2\}|$ after the modification of the graph H. On the other hand, by Lemma 5.3.33, the expected number of times that the oracle calls an adjacent edge of vertex v, is at most $\widetilde{O}(\deg_H(v)/k)$ (note that we use k here and n in Section 5.3.3 are equivalent since in the next section we used n as the number of vertices of a given graph). Also, these two variables have a negative correlation. Therefore, the total cost for all vertices is at most $\widetilde{O}(rk)$. Combining with the fact that $r=\widetilde{O}(\beta)$ (Lemma 5.3.10), the total additional cost is $\widetilde{O}(k\beta)$, which is dominated by other terms in Lemma 5.3.18 which implies that the algorithm has the same running time as Theorem 5.3.3 if we choose α and β similarly. □

5.3.3 Random Greedy Maximal Matching on Multigraphs

In this section, we show that the algorithm of [34] can be extended to work efficiently on multigraphs. First, we prove that if we have access to the adjacency list of a multigraph H, then it is possible to estimate the size of a random greedy maximal matching of H in $\widetilde{O}(\bar{d})$ time, where $\bar{d}$ is the average degree of H. We denote $H=(V_H,E_H)$. We slightly abuse notation by using n to denote $|V_H|$ (note that n here is equivalent to k in the set cover section). We also let π represent the permutation over the edges of H that we use to select edges of the RGMM.

Similar to the work of [34], we define a vertex oracle VO, which, given a vertex $v\in V_H$ and a permutation π over the edges of H, determines whether v is matched in $\text{GMM}(H,\pi)$. This oracle explores the neighborhood around v locally to answer the query and does not need to examine the entire graph. Additionally, to implement $\text{VO}(v,\pi)$, we define an edge oracle EO, which, given an edge $e\in E_H$ and a permutation π over the edges of E_H, determines whether e is matched in $\text{GMM}(H,\pi)$. Similar local oracles for random greedy maximal matching, random greedy maximal independent sets, and their modified versions have been extensively used in the literature [34, 43, 44, 154, 156, 176].

Both oracles are formally defined in Algorithm 32 and Algorithm 33.

Algorithm 32: Vertex oracle $\text{VO}(v,\pi)$ to determine if vertex v is matched in $\text{GMM}(H,\pi)$.

```
Let e_1 = (v, u_1), ..., e_d = (v, u_d) be the edges incident to v (including all copies of multiedges) with
  π(e_1) < ... < π(e_d).
for i in 1...d do
    if EO(e_i, u_i, π) = True then
        return True
return False
```

Algorithm 33: Edge oracle $\mathrm{EO}(e, v, \pi)$ to determine an edge e is in $\mathrm{GMM}(H, \pi)$ where v is an endpoint of e.

```
if EO(e, v, π) computed before then return the computed result;
Let e_1 = (v, u_1), ..., e_d = (v, u_d) be the edges incident to e (including all copies of multiedges) such
  that π(e_1) < ... < π(e_d) < π(e).
for i in 1...d do
    if EO(e_i, u_i, π) = True then
        return False
return True
```

It is not hard to see that the outputs of both the vertex oracle and the edge oracle are consistent with the RGMM, as the status of an edge (whether it is in the RGMM or not) depends only on the status of edges with a smaller rank in π. Both oracles utilize this property by querying the neighboring edges with smaller ranks in the permutation to determine if they are matched. Based on the results for these lower-ranked edges, the oracle can then decide if the vertex or edge will be matched.

Observation 5.3.20. $\mathrm{VO}(v, \pi) = \textsc{True}$ *if and only if* v *is matched in* $\mathrm{GMM}(H, \pi)$.

Observation 5.3.21. $\mathrm{EO}(e, v, \pi) = \textsc{True}$ *if and only if* $e \in \mathrm{GMM}(H, \pi)$.

We define $T(v, \pi)$ as the number of recursive calls made by $\mathrm{VO}(v, \pi)$ for a vertex $v \in V_H$ and a permutation π over E_H. The main result of this section is the following bound on $T(v, \pi)$.

Theorem 5.3.22. *Let* $\bar{d}$ *be the average degree of* H*. It holds that* $\mathbf{E}_{v,\pi}[T(v, \pi)] = O(\bar{d} \log n)$.

Let $Q(e, \pi)$ be the number of oracle calls to $\mathrm{EO}(e, \cdot, \pi)$ during the execution of VO for all vertices and permutation π. The following lemma is the main building block of the proof of Theorem 5.3.22.

Lemma 5.3.23. *Let* $e \in E_H$*. It holds that* $\mathbf{E}_{\pi}[Q(e, \pi)] = O(\log n)$.

Before proving Lemma 5.3.23 we complete the proof of Theorem 5.3.22. Recall that $\deg_H(v)$ denotes the degree of vertex v in H, where multiedges are counted according to their number of occurrences.

Proof of Theorem 5.3.22. We have

$$\begin{aligned}
\mathbf{E}_{v,\pi}[T(v, \pi)] = \frac{1}{n} \mathbf{E}_{\pi}\Big[\sum_{v \in V_H} T(v, \pi)\Big] &= \frac{1}{n} \mathbf{E}_{\pi}\Big[\sum_{e \in E_H} Q(e, \pi)\Big] \\
&= \frac{1}{2n} \sum_{v \in V_H} \sum_{e \ni v} \mathbf{E}_{\pi}[Q(e, \pi)] \\
&= \frac{1}{2n} \sum_{v \in V_H} \deg_H(v) \cdot O(\log n) \\
&= O(\bar{d} \cdot \log n).
\end{aligned}$$

□

In what follows, we focus on proving Lemma 5.3.23. The general framework and approach we use to demonstrate this claim are similar to those in [34], with some modifications to accommodate multigraphs.

We repeat all the steps to ensure completeness. For edge $e = (u, v)$, we use $\vec{e}$ to denote the directed edge from u to v and $\overleftarrow{e}$ for the directed edge from v to u. Similarly, we define $Q(\vec{e}, \pi)$ (resp. $Q(\overleftarrow{e}, \pi)$) as the total number of queries to the edge e directed from u to v (resp. from v to u).

Observation 5.3.24. *For any edge e, we have $\mathbf{E}_\pi[Q(e, \pi)] = \mathbf{E}_\pi[Q(\vec{e}, \pi)] + \mathbf{E}_\pi[Q(\overleftarrow{e}, \pi)]$.*

Proof. The proof follows from the fact that $Q(e, \pi) = Q(\vec{e}, \pi) \cup Q(\overleftarrow{e}, \pi)$ and $Q(\vec{e}, \pi) \cap Q(\overleftarrow{e}, \pi) = \emptyset$. □

Let $\mathcal{R}$ be the set of edges stored in memory for recursive calls during the execution of the vertex oracle defined in Algorithm 32. Additionally, if the edge $e = (u, v)$ is visited with direction from u to v (i.e., $\mathrm{EO}(e, v, \pi)$ is called), we assume that $\vec{e}$ is stored in $\mathcal{R}$; otherwise, $\overleftarrow{e}$ is stored in $\mathcal{R}$. It is not hard to observe that at any point during the oracle calls, the edges stored in $\mathcal{R}$ have decreasing ranks according to the permutation π.

Observation 5.3.25. *Let $(e_1, e_2, \ldots, e_d)$ be edges that are stored in $\mathcal{R}$ during the execution of vertex oracle (e_1 stored first). Then, it holds that $\pi(e_1) > \pi(e_2) > \ldots > \pi(e_d)$.*

Proof. The proof follows from the fact that $\mathrm{EO}(e, \cdot, \pi)$ only queries edges e' for which $\pi(e') < \pi(e)$. Consequently, the edges stored in memory at any point are ranked in decreasing order. □

Observation 5.3.26. *Let $(e_1, e_2, \ldots, e_d)$ represent the edges stored in $\mathcal{R}$ at some point during the execution of the vertex oracle, in the order they are visited by the oracle. Then, $(e_1, e_2, \ldots, e_d)$ forms a path in H.*

Proof. From the definition of the edge oracle, $P = (e_1, e_2, \ldots, e_d)$ is clearly a walk in the graph since each time the edge oracle queries an incident edge. Now we show that P is a path. For the sake of contradiction, let e_i be the smallest index where $(e_1, \ldots, e_{i-1})$ is a path but $(e_1, \ldots, e_i)$ is not. Let $e_i = (w, z)$ where w is shared with e_{i-1}. Hence, vertex z is one of the endpoints of $e_1, \ldots, e_{i-1}$ (it might be e_{i-1} since we have multiedges). Let e_j $(j < i)$ be the edge that the edge oracle queries as an incident edge of z at the time it visits z. By Observation 5.3.25 we have that $\pi(e_i) < \pi(e_j)$. Thus, the edge oracle already queried e_i before e_j and the reason it did not add e_i to the matching is because some other neighbor of w with smaller ranking is already in the matching. Therefore, that neighbor must be queried before e_i and the edge oracle must stop querying neighbors of w at that point, which is a contradiction. Therefore, P is a path. □

We define a query path as a path that is stored in $\mathcal{R}$ at some point during the execution of vertex oracle. Let $\vec{P} = (\vec{e_1}, \vec{e_2}, \ldots, \vec{e_d})$ be a query path for some permutation $\pi \in \Pi$, where Π denotes the set of all permutations over the edges of multigraph H. Next, we define a function that maps a permutation π to some permutation based on $\vec{P}$.

Definition 5.3.27 (Function f)**.** *Let $\vec{P} = (\vec{e_1}, \vec{e_2}, \ldots, \vec{e_d})$ be a query path and $\pi \in \Pi$ be some permutation over the edges of H. We define a function $f(\pi, \vec{P}) \in \Pi$ that maps permutation π to another permutation $f(\pi, \vec{P})$ such that $f(\pi, \vec{P})(e_i) = \pi(e_{i+1})$ for $i < k$, $f(\pi, \vec{P})(e_k) = \pi(e_1)$, and $f(\pi, \vec{P})(e) = \pi(e)$ for $e \notin \vec{P}$.*

Now based on Definition 5.3.27, we are able to construct a bipartite graph $H^{\vec{e}}_\Pi = (X^{\vec{e}}_\Pi, Y^{\vec{e}}_\Pi, E^{\vec{e}}_\Pi)$ for a given edge $\vec{e}$ that is crucial for proving Lemma 5.3.23.

Definition 5.3.28 (Bipartite Graph $H^{\vec{e}}_\Pi = (X^{\vec{e}}_\Pi, Y^{\vec{e}}_\Pi, E^{\vec{e}}_\Pi)$)**.** *Let $\vec{e}$ be an edge in H with some direction. We define $H^{\vec{e}}_\Pi = (X^{\vec{e}}_\Pi, Y^{\vec{e}}_\Pi, E^{\vec{e}}_\Pi)$ to be a bipartite graph such that $|X^{\vec{e}}_\Pi| = |Y^{\vec{e}}_\Pi| = |E_H|!$, where each vertex in*

each part corresponds to a permutation over the edges of H. Let $x_\pi \in X_\Pi$ be a vertex that corresponds to permutation π and $\vec{P}$ be a query path obtained on permutation π for some vertex oracle call that ends to $\vec{e}$. Then, $E^{\vec{e}}_\Pi$ contains the edge $(x_\pi, y_{f(\pi,\vec{P})})$, where $y_{f(\pi,\vec{P})} \in Y^{\vec{e}}_\Pi$ corresponds to permutation $f(\pi, \vec{P})$.

Observation 5.3.29. $\deg_{H^{\vec{e}}_\Pi}(x_\pi) = Q(\vec{e}, \pi)$.

Proof. For each query path that ends with $\vec{e}$ that can be produced by some vertex oracle call on permutation π, we add an incident edge to x_π. Hence, $\deg_{H^{\vec{e}}_\Pi}(x_\pi) = Q(\vec{e}, \pi)$. □

As a result of Observation 5.3.29, in order to prove Lemma 5.3.23, it is sufficient to prove that $\mathbf{E}_{x\sim X^{\vec{e}}_\Pi}[\deg_{H^{\vec{e}}_\Pi}(x)] = O(\log n)$. We call a permutation $\pi \in \Pi$ a *bad permutation* if and only if there exists a query path for this permutation that ends at $\vec{e}$ and it has a length larger than $c \log n$ for some large constant c. We use $\overline{X}^{\vec{e}}_\Pi \subset X^{\vec{e}}_\Pi$ to denote the set of vertices that correspond to bad permutations in $X^{\vec{e}}_\Pi$. We prove the following two lemmas, which are sufficient to achieve our goal.

Lemma 5.3.30. *If c is a large enough constant, then $|\overline{X}^{\vec{e}}_\Pi| \le |E_H|!/n^2$.*

Lemma 5.3.31. *Let $y \in Y^{\vec{e}}_\Pi$. Then the number of neighbors of y in $X^{\vec{e}}_\Pi \setminus \overline{X}^{\vec{e}}_\Pi$ is at most $c \log n$.*

Their proofs are deferred to later sections.

Proof of Lemma 5.3.23. First, note that $\deg_{H^{\vec{e}}_\Pi}(x_\pi) \le O(n^2)$ for $x_\pi \in X^{\vec{e}}_\Pi$. To see this, suppose that we run $\mathrm{VO}(v, \pi)$ for some vertex v. The edge oracle for e is either directly called by $\mathrm{VO}(v, \pi)$ or from some neighboring edge oracle calls (the first time that the oracle visits the edge because of caching in Algorithm 33 of Algorithm 33). Hence, $\mathrm{VO}(v, \pi)$ produces at most n edge oracle calls to e. Since we have n different options for v, the total number of query paths to $\vec{e}$ is at most $O(n^2)$. Further, each edge of x_π in graph $H^{\vec{e}}_\Pi$ corresponds to a query path that ends with $\vec{e}$. Therefore, we have $\deg_{H^{\vec{e}}_\Pi}(x_\pi) \le O(n^2)$.

Now, we prove that $\mathbf{E}_{x\sim X^{\vec{e}}_\Pi}[\deg_{H^{\vec{e}}_\Pi}(x)] = O(\log n)$. The total number of edges between $Y^{\vec{e}}_\Pi$ and $X^{\vec{e}}_\Pi \setminus \overline{X}^{\vec{e}}_\Pi$ is at most $|Y^{\vec{e}}_\Pi| \cdot c \log n$ by Lemma 5.3.31. Moreover, the total number of edges between $Y^{\vec{e}}_\Pi$ and $\overline{X}^{\vec{e}}_\Pi$ is at most $|\overline{X}^{\vec{e}}_\Pi| \cdot O(n^2)$ because $\deg_{H^{\vec{e}}_\Pi}(x_\pi) \le O(n^2)$. Therefore, we have

$$|E^{\vec{e}}_\Pi| \le |Y^{\vec{e}}_\Pi| \cdot c \log n + |\overline{X}^{\vec{e}}_\Pi| \cdot O(n^2) \le O(|E_H|! \cdot \log n),$$

where the last inequality follows by Lemma 5.3.30. Thus

$$\mathbf{E}_{x\sim X^{\vec{e}}_\Pi}[\deg_{H^{\vec{e}}_\Pi}(x)] = \frac{|E^{\vec{e}}_\Pi|}{|E_H|!} \le \frac{O(|E_H|! \cdot c \log n)}{|E_H|!} \le O(\log n).$$

Plugging in Observation 5.3.29, we obtain $\mathbf{E}_\pi[Q(\vec{e}, \pi)] = O(\log n)$. Finally, by Observation 5.3.24, we have $\mathbf{E}_\pi[Q(e, \pi)] = O(\log n)$. □

Implementation Details of RGMM for Multigraphs

Until now, we demonstrate that the query complexity of our vertex oracle is $O(\bar{d} \log n)$. Now we show how to utilize this oracle to estimate the size of RGMM. The following result is due to [34].

Proposition 5.3.32 (See Appendix A in [34])**.** *Let $T(v)$ be the time needed to return a random neighbor of vertex v that is not exposed to v yet. Also, let $Q(v)$ be the expected number of times that the algorithm*

needs a random neighbor of v if we start the oracle calls from a random vertex for a random permutation. Then, the expected time to run the vertex oracle for a random vertex and a random permutation is at most $\sum_v T(v) \cdot Q(v)$.

Remark 11. *Note that [34] proved the above lemma for $T(v) = O(1)$, but essentially the same proof applies to any $T(v)$.*

Up to this point, we are able to estimate $E_\pi|\mathrm{GMM}(H, \pi)|$ (using $\widetilde{\Theta}(1)$ uniformly random vertex oracle calls) efficiently if we have access to the adjacency list of the multigraph H because of Proposition 5.3.32. However, this is not feasible in our application because we do not have access to the adjacency list of H, and building it before running the vertex oracle is too costly. We use the following property of RGMM to refine Proposition 5.3.32 into a tighter bound that can be applied to our set cover problem.

Lemma 5.3.33. *Let $Q(v)$ be the expected number of times that the oracle queries an adjacent edge of v if we start the oracle calls from a random vertex, for a random permutation over the edges of the multigraph H. It holds that $Q(v) = \widetilde{O}(\deg_H(v)/n)$.*

Proof. Let $e_1, e_2, \ldots, e_{\deg_H(v)}$ be all edges incident to v in H. Thus, we have

$$\begin{aligned} Q(v) = \frac{1}{n} \cdot \frac{1}{|E_H|!} \sum_{\pi \in \Pi} \sum_{i=1}^{\deg_H(v)} Q(e_i, \pi) &= \frac{1}{n} \sum_{i=1}^{\deg_H(v)} \frac{1}{|E_H|!} \sum_{\pi \in \Pi} Q(e_i, \pi) \\ &= \frac{1}{n} \sum_{i=1}^{\deg_H(v)} \mathbf{E}_\pi[Q(e_i, \pi)] \\ &= \frac{1}{n} \sum_{i=1}^{\deg_H(v)} O(\log n) \qquad \text{(By Lemma 5.3.23)} \\ &= \widetilde{O}(\deg_H(v)/n) \end{aligned}$$

□

Corollary 5.3.34. *Let $T(v)$ be the time needed to return a random neighbor of vertex v that is not exposed to v yet. Then, the expected time to run the vertex oracle for a random vertex and a random permutation is $\sum_v \widetilde{O}(T(v) \cdot \deg_H(v)/n)$.*

Proof. The proof can be obtained by plugging Lemma 5.3.33 into Proposition 5.3.32. □

Proof of Lemma 5.3.30

We use the following result for the round-complexity of the parallel randomized greedy maximal independent set by [90].

Proposition 5.3.35 ([90])**.** *Let π be a permutation of the vertices of a graph G with n vertices, drawn uniformly at random. For any constant c, with probability $1 - n^{-c}$, we have $\rho(G, \pi) = O(\log n)$.*

Given a graph G, we can construct its line graph $L(G)$ by adding a vertex in $L(G)$ for each edge of G and adding an edge between two vertices of $L(G)$ if their corresponding edges share an endpoint in G. It is easy to see that a random greedy MIS on $L(G)$ is equivalent to a random greedy maximal matching of G, which implies the following corollary as a result of Proposition 5.3.35.

Corollary 5.3.36. *Let π be a permutation of the edges of a graph G with n vertices, drawn uniformly at random. For any constant c, with probability $1 - n^{-c}$, we have $\rho(L(G), \pi) = O(\log n)$.*

Note that for any edge that appears in the solution of parallel random greedy MIS of $L(H)$, if the edge oracle queries that edge, the answer to this query is going to be consistent since they simulate the same greedy algorithm with respect to the given permutation. Now we prove that the round-complexity of random greedy MIS on the line graph of H is large for bad permutations which is enough to show Lemma 5.3.30. Let π be a bad permutation. Hence, there exists a query path $\vec{P} = (\vec{e_1}, \vec{e_2}, \ldots, \vec{e_r})$ for this permutation such that $r > c \log n$. Let $\rho(e_i)$ be the round edge e_i is removed from $L(H)$ when we run parallel randomized greedy MIS on $L(H)$ with respect to π. We claim that $\rho(e_i) > \rho(e_{i+2})$ for $1 < i < r - 1$. For the sake of contradiction, assume that $\rho(e_i) \leq \rho(e_{i+2})$. Thus, $\rho(e_i) \leq \rho(e_{i+1})$ since if e_{i+1} removed from the graph before e_i, it means that it is removed because an edge adjacent to e_{i+1} but not adjacent to e_i is in MIS of $L(H)$ which implies that e_{i+2} must be removed at the same or before, so $\rho(e_{i+2}) \leq \rho(e_{i+1}) < \rho(e_i)$ which is a contradiction. So it must hold that $\rho(e_i) \leq \rho(e_{i+1})$ which implies that either e_i and e_{i+1} are getting deleted from $L(H)$ in the same round or e_i is removed before e_{i+1}. Hence, when we are removing e_i, it is not a local minimum with respect to π. Now consider the round that e_i is removed from $L(H)$. Let $e_i = (u, v)$ and v be the shared endpoint with e_{i+1}. There are three possible scenarios:

- e_i **is removed because another multiedge** (u, v) **is in MIS:** Let $e' = (u, v)$ be the multiedge that is a local minimum in the round we remove e_i. Note that we have $\pi(e') < \pi(e_i)$. Thus, the edge oracle of e_{i-1} must first queries e' before e_i. Then, since e' is in random greedy MIS of $L(H)$ it must stop the process at this point and does not query e_i which is a contradiction.

- e_i **is removed because edge** (u, w) **is in MIS where** w **some vertex of the graph:** Let $e' = (u, w)$. Similar to the previous case, the edge oracle for e_{i-1} must first query e' and since e' is in the solution it must stop the process which is a contradiction.

- e_i **is removed because edge** (v, w) **is in MIS where** w **some vertex of the graph:** Let $e' = (v, w)$. Similar to the previous case, the edge oracle for e_i must first query e' and since e' is in the solution it must stop the process which is a contradiction.

Therefore, we have $\rho(e_i) > \rho(e_{i+2})$ which implies that $\rho(e_2) \geq \rho(e_4) + 1 \geq \rho(e_6) + 2 \geq \ldots \geq r/2 - 1$. Let c be sufficiently large enough. Hence, we have $\rho(L(H), \pi) \geq r/2 - 1 \geq c \log n/2 - 1$. By Corollary 5.3.36, this situation occurs for at most a $1/n^2$ fraction of permutations, which completes the proof of Lemma 5.3.30.

Proof of Lemma 5.3.31

Let $y_\pi \in Y_\Pi^{\vec{e}}$. Also, let $x_{\pi_1}, x_{\pi_2} \in X_\Pi^{\vec{e}} \setminus \overline{X}_\Pi^{\vec{e}}$ be two vertices corresponding to permutations π_1 and π_2 such that they are connected to y_π. Let $\vec{P_1}$ and $\vec{P_2}$ be the two query paths such that $f(\pi_1, \vec{P_1}) = f(\pi_2, \vec{P_2}) = \pi$. We show that either $P_1 \subseteq P_2$ or $P_2 \subseteq P_1$. In other words, one of P_1 or P_2 must be a subpath of the other path. This is enough to complete the proof of Lemma 5.3.31 the longest query path of all permutations in $X_\Pi^{\vec{e}}$ is $O(\log n)$ by the definition. So for the rest of this subsection, we focus on proving this claim.

Let $\vec{P_1} = (\vec{e_{k_1}}, \ldots, \vec{e_2}, \vec{e_1})$ and $\vec{P_2} = (\vec{e_{k_2}}', \ldots, \vec{e_2}', \vec{e_1}')$ where we have $e_1 = e_1' = e$. For the sake of contradiction, suppose that none of the two paths is a subpath of the other one. Because of this assumption, there exists some i such that $\vec{e_j} = \vec{e_j}'$ for $j \leq i$ and $\vec{e_{i+1}} \neq \vec{e_{i+1}}'$. Without loss of generality, assume that

$\pi_1(e_i) \leq \pi_2(e_i)$ (because of the symmetry up to this point in the proof). By the definition of function f (Definition 5.3.27) and the fact that $\pi = f(\pi_1, \vec{P}_1) = f(\pi_2, \vec{P}_2)$, we have

$$\pi_2(e_{i+1}) = f(\pi_2, \vec{P}_2)(e_{i+1}) = f(\pi_1, \vec{P}_1)(e_{i+1}) = \pi_1(e_i). \tag{5.13}$$

Moreover, combining with equality (5.13) and the assumption that $\pi_1(e_i) \leq \pi_2(e_i)$, we get

$$\pi_2(e_{i+1}) = \pi_1(e_i) \leq \pi_2(e_i).$$

On the other hand, we know π_2 is a permutation over edges of H which implies that

$$\pi_2(e_{i+1}) < \pi_2(e_i). \tag{5.14}$$

Let $\hat{e}$ be an arbitrary edge. We claim that if $\min(\pi_1(\hat{e}), \pi_2(\hat{e})) < \pi_1(e_i)$, then $\pi_1(\hat{e}) = \pi_2(\hat{e})$. If $\hat{e} \notin P_1 \cup P_2$, then the claim clearly holds. If $\hat{e} \in \{e_1, \ldots, e_{i-1}\}$, i.e. $\hat{e} = e_j$ for $j < i$, we have

$$\pi_1(\hat{e}) = \pi_1(e_j) = f(\pi_1, \vec{P}_1)(e_{j+1}) = f(\pi_2, \vec{P}_2)(e_{j+1}) = \pi_2(e_j) = \pi_2(\hat{e}).$$

In all other cases, $\min(\pi_1(\hat{e}), \pi_2(\hat{e})) \geq \pi_1(e_i)$.

Claim 5.3.37. *Assuming that neither of P_1 or P_2 is a subpath of the other, the edge e_{i+1} is in the random greedy maximal matching of H with respect to permutation π_2.*

Proof. If e_{i+1} is not in the RGMM of permutation π_2, there must exists some edge $\hat{e}$ that blocks e_{i+1} from being in RGMM such that $\pi_2(\hat{e}) < \pi_2(e_{i+1})$. Combining with equality (5.13), we obtain $\pi_2(\hat{e}) < \pi_1(e_i)$. Since $\min(\pi_1(\hat{e}), \pi_2(\hat{e})) < \pi_1(e_i)$, we have $\pi_1(\hat{e}) = \pi_2(\hat{e})$. Thus, $\pi_1(\hat{e}) < \pi_1(e_i)$. Because both π_1 and π_2 are similar up to ranking $\pi(e_i)$, then edge $\hat{e}$ must also be in RGMM of permutation π_1.

Let $\vec{e_{i+1}} = (u, v)$. There are three possible scenarios for $\hat{e}$:

- $\hat{e} = (u, v)$**, i.e. is one of the multiedges between** u **and** v **similar to** e_{i+1}**:** In this case, either $\text{VO}(u, \pi_1)$ or $\text{EO}(e_{i+2}, u, \pi_1)$ query $\text{EO}(\hat{e}, v, \pi_1)$ before $\text{EO}(e_{i+1}, v, \pi_1)$ since $\pi_1(\hat{e}) < \pi_1(e_{i+1})$. Since $\hat{e}$ is in RGMM of permutation π_1, the process terminates and $\vec{P}_1$ is not a valid query path.
- $\hat{e} = (u, w)$ **for some vertex** w**:** In this case, either $\text{VO}(u, \pi_1)$ or $\text{EO}(e_{i+2}, u, \pi_1)$ query $\text{EO}(\hat{e}, w, \pi_1)$ before $\text{EO}(e_{i+1}, v, \pi_1)$ since $\pi_1(\hat{e}) < \pi_1(e_{i+1})$. Since $\hat{e}$ is in RGMM of permutation π_1, the process terminates and $\vec{P}_1$ is not a valid query path.
- $\hat{e} = (v, w)$ **for some vertex** w**:** In this case, either $\text{EO}(e_{i+1}, v, \pi_1)$ query $\text{EO}(\hat{e}, w, \pi_1)$ before $\text{EO}(e_i, \cdot, \pi_1)$ since $\pi_1(\hat{e}) < \pi_1(e_i)$. Since $\hat{e}$ is in RGMM of permutation π_1, the process terminates and $\vec{P}_1$ is not a valid query path.

Therefore, e_{i+1} is in the random greedy maximal matching of H with respect to permutation π_2. □

Now we are ready to prove the contradiction which completes the proof of Lemma 5.3.31. By inequality (5.14), edge oracle $\text{EO}(e'_{i+1}, \cdot, \pi_2)$ must query e_{i+1} before e_i. Also, by Claim 5.3.37, edge e_{i+1} is in the random greedy maximal matching of H with respect to permutation π_2, which means that P_2 is not a valid query path.

Implementation of RGMM for Set Cover with Our Access Model

In this section, we demonstrate how we can change the RGMM algorithm to work with our access model in the set cover problem instead of having access to the adjacency list. The challenge arises when the algorithm needs to find a random neighbor of a vertex v (which corresponds to an element in the set cover problem) in graph H. Naively, the algorithm queries all sets in $\hat{\mathcal{S}}$ and finds all sets that cover the element corresponding to v. Then, for each of the sets, it queries all elements in $\mathcal{U}_{low}$ to find all neighbors of v in H.

Lemma 5.3.17. *Let H be the multigraph defined in Definition 5.3.5. There exists an algorithm with an expected running time of $\widetilde{O}_\varepsilon(k\beta + \alpha\beta n)$ that estimates the value of $E_\pi|\mathrm{GMM}(H,\pi)|$ with εk additive error with high probability.*

Proof. First, it is important to mention that the number of vertices in H is k. In order to estimate $E_\pi|\mathrm{GMM}(H,\pi)|$ with εk additive error, it is sufficient to run the vertex oracle for $\Theta(1/\varepsilon \cdot \mathrm{poly}\log k))$ random vertices and permutations. Then, using a Chernoff bound, it is easy to show that we can estimate $E_\pi|\mathrm{GMM}(H,\pi)|$ with εk additive error.

In order to simulate the oracles in our access model and use Corollary 5.3.34, we demonstrate how we can find a random neighbor of a vertex v in H. Consider the first time that the algorithm needs to find a random neighbor of v. We first query all sets in $\hat{\mathcal{S}}$ to identify those that contain the element corresponding to v. Let $\hat{\mathcal{S}}_v$ be the collection of sets that include v. This step takes $O(n)$ time since $|\hat{\mathcal{S}}| \le n$. By Lemma 5.3.10, we have $|\hat{\mathcal{S}}_v| \le \widetilde{O}(\beta)$ with high probability. Now consider all pairs of (u,S) where $v \in \mathcal{U}_{low}$ and $S \in \hat{\mathcal{S}}_v$. There are at most $\widetilde{O}(k\beta)$ such pairs. We start to query these pairs randomly until finding an element that exists in one of the sets of $\hat{\mathcal{S}}_v$. Note that in expectation, the algorithm needs to make $T(v) = \widetilde{O}(k\beta/\deg_H(v))$ queries to find such an element which is a neighbor of v in H. Therefore, using a concentration inequality such as a Chernoff bound, with high probability the algorithm finds such an element in $T(v)\cdot \mathrm{poly}\log n = \widetilde{O}(k\beta/\deg_H(v))$ attempts. Also, since the algorithm makes all the queries in random order, it has the same probability of seeing any edges adjacent to v.

Therefore, each time that the algorithm needs to find a neighbor of vertex v in H, it spends at most $\widetilde{O}(n + k\beta/\deg_H(v))$ time. Now by Corollary 5.3.34, the expected time to run the vertex oracle for a random vertex and a random permutation is

$$\begin{aligned}\sum_v \widetilde{O}\left(\frac{T(v)\cdot\deg_H(v)}{k}\right) = \sum_v \widetilde{O}\left(\frac{(n+k\beta/\deg_H(v))\cdot\deg_H(v)}{k}\right) &= \sum_v \widetilde{O}\left(\frac{n\cdot\deg_H(v)}{k} + \beta\right)\\ &= \widetilde{O}\left(k\beta + n\cdot\sum_v \frac{\deg_H(v)}{k}\right)\\ &= \widetilde{O}(k\beta + n\bar{d}),\end{aligned}$$

where $\bar{d}$ is the average degree of multigraph H.

On the other hand, each set in $\hat{\mathcal{S}}_v$ has at most $\widetilde{O}(\alpha)$ neighbors in $\mathcal{U}_{low}$ by Lemma 5.3.9. Therefore, vertex v has at most $\widetilde{O}(\alpha\cdot\beta)$ neighbors in H since $|\hat{\mathcal{S}}_v| \le \widetilde{O}(\beta)$. Thus, we have $\bar{d} \le \widetilde{O}(\alpha\cdot\beta)$ which completes the proof. □

5.3.4 Connection to Steiner Tree

In this section, we show how our improved algorithm for set cover (from Section 5.3.2) implies an improved sublinear algorithm for metric Steiner tree. Formally, we show the following theorem.

Theorem 5.3.4 (Sublinear Algorithm for Metric Steiner Tree)**.** *There exists an algorithm that, given an instance of metric Steiner tree denoted by (V, T, w) with oracle access $\mathcal{O}$ to the distance matrix of (V, w), outputs a $(2-\eta)$-estimate of $\mathsf{ST}(V, T, w)$ using $\widetilde{O}(n^{5/3})$ queries to $\mathcal{O}$, where $\eta > 0$ is a universal constant, with high probability.*

The overall structure of our algorithm is similar to the algorithm of Chen, Khanna and Tan [69]. The main difference in our algorithm compared to [69] is in the set cover component. In the following, we first provide an overview of their algorithm and then we provide the query-efficient implementation of their algorithm and our modification to it. We will finally provide the query complexity analysis and the proof of Theorem 5.3.4. Note that the approximation analysis of our algorithm follows directly from the proof of Theorem 3 in [69].

Algorithm at a High Level

Step 1: Minimum spanning tree over terminals. The algorithm of [69] as a first step starts with an MST $\mathcal{T}^*$ over the terminals T (whose cost can be estimated in nearly linear time using the sublinear MST algorithm of Czumaj and Sohler [75]). It is known that $w(\mathcal{T}^*)/2 \leq \mathsf{ST}(V, T, w) \leq w(\mathcal{T}^*)$. To get a strictly better-than-2 approximation of $\mathsf{ST}(V, T, w)$, it suffices to detect whether $\mathsf{ST}(V, T, w)$ is closer to $w(\mathcal{T}^*)/2$ or $w(\mathcal{T}^*)$. Hence, the rest of the algorithm is either to provide "significant" local improvements over $w(\mathcal{T}^*)$ using "set cover" like structure (i.e., step 2 in Section 4 of [69]) or "local structure" (i.e., step 3 in Section 4 of [69]), and thus output $(1 - O(\eta))w(\mathcal{T}^*)$ as the estimate of $\mathsf{ST}(V, T, w)$; or conclude that $\mathsf{ST}(V, T, w)$ is closer to $w(\mathcal{T}^*)$ and output it as the estimate of $\mathsf{ST}(V, T, w)$. Then, they show how to implement these steps using sublinear queries to $\mathcal{O}$.

Step 2: Improvement using set cover. First, they partition the edges into $L = O((\log k)/\varepsilon)$ buckets such that the edges of the ith bucket have weights in $[(1+\varepsilon)^{i-1}, (1+\varepsilon)^i)$. Let H_i be the graph built on all the terminals and all the edges upto the ith bucket. They define an instance of set cover corresponding to each level i where *ideally*,

- The elements correspond to the connected components of H_{i-1}.
- The sets correspond to the Steiner vertices.
- A set W_v (corresponding to a Steiner vertex v) contains an element u_S (corresponding to a component S) if the distance between v and some terminal in S is less than a threshold τ (think of it as $\frac{3}{5} \cdot (1+\varepsilon)^i$).

How to use set cover in making a decision about the Steiner tree cost. They show that if one can solve each of these set cover instances approximately, then one can check whether the total contribution of these set cover improvements is more than $O(\eta) \cdot w(\mathcal{T}^*)$. In particular, in that case $\mathsf{ST}(V, T, w)$ is strictly less than $(1 - O(\eta)) \cdot w(\mathcal{T}^*)$. Intuitively, this is because one can include the Steiner vertices corresponding to the set cover solution and remove a subset of the edges in $\mathcal{T}^*$, while still maintaining a feasible solution

to the Steiner tree instance. Hence, this implies that the cost of the constructed solution of the Steiner tree instance is less than $(1 - O(\eta)) \cdot w(\mathcal{T}^*)$.

A challenge and the notion of representatives. The main challenge with the above algorithm is computing the set cover instance. More precisely, in the third bullet point above, in order to check whether a set (corresponding to a Steiner vertex v) contains an element (corresponding to a connected component S), they need to compute the distance of v to all terminals $t \in T$ which could be very costly. Instead, they define a *net* on S which is a maximal subset $\tilde{S} \subseteq S$ such that any pair of terminals in $\tilde{S}$ has distance at least $\varepsilon \cdot (1 + \varepsilon)^i$. They call the terminals in $\tilde{S}$ the *representatives*.

Modified set cover instance and the notion of light/heavy levels. Now, to detect if a set contains an element, we only need to check the distance of v to all terminals in $\tilde{S}$. When $|\tilde{S}|$ is small, this can be done efficiently. So, in their algorithm they *only* assign an element to a connected component S if the size of its representatives is *small*. To show that this does not introduce a large error, they define a level i to be *light* if the total sum of the edges of $\mathcal{T}^*$ in bucket i is "small", and define it to be *heavy* otherwise. They show that one can ignore all the levels i that are light, and moreover if a level is heavy, then most of its components have small sets of representatives and thus the error introduced by ignoring the components S with large net size is negligible.

Finally, we note that the notion of representatives will be used in other parts of the overall algorithm such as computing $\mathcal{T}^*$ which we will go over when describing the implementation of this step.

Step 3: Improvement using "local structure". In this step, they consider the hierarchical structure of the connected components. Specifically, they focus on components S that have exactly two child components, S_1 and S_2, where each of these child components also has exactly two child components: S_{11}, S_{12} for S_1 and S_{21}, S_{22} for S_2. Then, they check whether there exists a single Steiner vertex v that can be used to connect components $S_{11}, S_{12}, S_{21}, S_{22}$ and instead remove the corresponding edges in $\mathcal{T}^*$ connecting these components. Similarly to step 2, if the overall advantage of all these 2-level local improvements is more than $O(\eta) \cdot w(\mathcal{T}^*)$, then the algorithm outputs $(1 - O(\eta)) \cdot w(\mathcal{T}^*)$ as its estimate of $\mathsf{ST}(V, T, w)$.

Given that our algorithm does not change this step at all, we refer the reader to [69] for further details. Moreover, the query complexity is exactly the same as in [69].

Implementation of the Algorithm

Here, we focus on a query-efficient implementation of the algorithm, and particularly highlight where the set cover component was used and how we modify it.

First, we note that one cannot compute H_i exactly, so [69] shows that it suffices to work with an approximate graph H_i' such that $H_i \subseteq H_i' \subseteq H_{i+1}$.

Subroutines. Next, [69] define some useful subroutines for simulating the set cover instance on H_i':

- $\mathsf{Find}(u, i)$: This receives a terminal u and a level i, and finds some representative terminal u' in the same connected component of H_i' that contains u. Moreover, they show that this subroutine can be implemented using $\widetilde{O}(k)$ queries.

- $\mathsf{BFS}(u, i)$: This subroutine reports all representative terminals that are in the same connected component in H'_i as u, if the total number of such representatives is $\widetilde{O}(L/\varepsilon) = \widetilde{O}(1/\varepsilon)$. Otherwise, the procedure is terminated. This procedure employs Find subroutines and has total query complexity of $\widetilde{O}(k/\varepsilon)$ which is $\widetilde{O}(k)$ given that ε is a constant.

Parameters. The algorithm uses four parameters, whose values will be later set to optimize the query complexity of the algorithm.

- M is a threshold parameter used on the number of components that contain a "small" number of representatives. Note that these are the components to which we assign an element in the universe $\mathcal{U}_i$ of the corresponding set cover instance $(\mathcal{U}_i, \mathcal{F}_i)$.
- R is a threshold parameter defining "low-degree" and "high-degree" elements.
- P is a threshold parameter denoting "low-degree" or "high-degree" sets.
- κ is a threshold parameter on the value of k. At a high level, when $k < \kappa$, we can afford to query all distances between terminals and the Steiner vertices.

Simulation of Step 2 and its query complexity. We now outline the implementation of Step 2 and specify the query complexity of each step, along with potential conditions they impose on the parameters we need to set.

- **The case of small number of terminals: if** $k \leq \kappa$, then we query all distances between terminals and Steiner vertices, which takes $O(n\kappa)$ queries. Then, we estimate $|\mathcal{U}| - \mathsf{SC}(\mathcal{U}, \mathcal{F})$ using our algorithm, but without any further queries.
- **Otherwise,** for each level i, they show that one of the following cases hold:

  Case 1: The total number of representative terminals in all connected components is $\widetilde{O}(M/\varepsilon)$. To detect this case, they use greedy MIS which can be implemented by the BFS and Find subroutines and will take $\widetilde{O}(Mk/\varepsilon)$ queries (for further details, see [176]). If this is the case, then again the set cover instance can easily be computed by querying the distance of all Steiner vertices to all representative terminals which requires $\widetilde{O}(nM/\varepsilon)$ queries. Then, similarly to the case of $k \leq \kappa$, $|\mathcal{U}_i| - \mathsf{SC}(\mathcal{U}_i, \mathcal{F}_i)$ can be estimated using our algorithm, without any further queries.

  Case 2: $|\mathcal{U}_i| \leq M$. In this case they show that level i is in fact light and thus can be ignored. To detect this case, they estimate the size of $|\mathcal{U}_i|$ using calls to BFS starting from $\widetilde{O}(k/M)$ random terminals, which overall takes $\tilde{O}(k^2/M)$ queries. Note that if we are in this case, we take no further action. Also, this step requires $k > M$, which we will ensure in our parameter setup.

  Case 3: The last case is when $|\mathcal{U}_i| \geq M$.

– **Partitioning of the terminals based on their degree.** First, they partition the terminals into T_{low} and T_{high} based on whether the number of "close-by" Steiner vertices (roughly within distance $(3/5)(1+\varepsilon)^i$) to them is smaller than or larger than R. This partitioning can be computed using $\widetilde{O}(kn/R)$ queries by randomly sampling $\widetilde{O}(n/R)$ Steiner vertices and checking their distance to all the terminals.

– **Handling high-degree terminals.** Then, by picking $\widetilde{O}(n/R)$ sets uniformly at random, with high probability, all elements corresponding to the components containing at least one terminal in T_{high} are covered. As they can only afford an $\varepsilon|\mathcal{U}_i|$ additive error in their estimate of $|\mathcal{U}_i| - \mathsf{SC}(\mathcal{U}_i, \mathcal{F}_i)$, they require that $n/R < \widetilde{O}(\varepsilon M) = \widetilde{O}(\varepsilon|\mathcal{U}_i|)$.

– **Handling low-degree terminals.** Next, they solve the set cover instance on $\mathcal{U}_{low}$, i.e., the connected components that have no terminal in T_{high}.

- **Partitioning of the Steiner vertices based on their degree.** They partition the sets of $\mathcal{F}_i$ into $\mathcal{W}_1$ and $\mathcal{W}_2$ based on whether their degree to T_{low} is less than $\Theta(P)$ or higher. This partitioning can be computed using $\widetilde{O}(nk/P)$ queries by randomly sampling k/P terminals from T_{low}. This requires $k > \widetilde{\Omega}(P)$ which we will ensure in our parameter setup. Then, they consider the set cover instances $(\mathcal{U}_{low}, \mathcal{W}_1)$ and $(\mathcal{U}_{low}, \mathcal{W}_2)$ separately, and return the better of the two solutions.

  Our modification to this step: In our set cover algorithm, however, we define $\mathcal{W}_2$ slightly differently, as described in Set Sparsification, see Algorithm 30. More precisely, we iterate over Steiner vertices (i.e., sets in $\mathcal{F}_i$) one by one in an arbitrary order, and at every round $j \le |\mathcal{F}_i|$, we check whether the degree of the Steiner vertex v_j to T_{low} is more than $\Theta(P)$. If so, we add its corresponding set, W_j, to $\mathcal{W}_2$. Similarly to their test, our test can also be implemented using $\widetilde{O}(nk/P)$ queries. However, each time we add a set W_j to $\mathcal{W}_2$, we find all its "nearby" terminals and remove them from T_{low}, more precisely, $T_{low} \leftarrow T_{low} \setminus W_j$. This step can be simply done by querying the distance of the Steiner vertex v_j and all terminals in T_{low}. Hence, each time a set is added to $\mathcal{W}_2$, we perform an extra $\widetilde{O}(k)$ queries compared to the algorithm of [69]. However, as we can simply bound $|\mathcal{W}_2|$ by k/P, the overall query complexity remains as $\widetilde{O}(nk/P + k^2/P) = \widetilde{O}(nk/P)$.

- **Handling high-degree Steiner vertices.** To solve $(\mathcal{U}_{low}, \mathcal{W}_2)$, note that by a simple double-counting argument, $|\mathcal{W}_2| \le kR/P$. So if $kR/P \le \varepsilon M \le \varepsilon|\mathcal{U}_i|$, which will be ensured in the parameter setting, we can afford to pick all sets in $\mathcal{W}_2$ and thus, similarly to their argument, we only need to estimate $|\bigcup_{W\in\mathcal{W}_2} W|$ in the set cover instance. This is done by randomly sampling the terminals and using BFS and will take an overall $\widetilde{O}(k^2/M)$ queries.

  Our modification to this step: With the adjusted partitioning of $\mathcal{F}_i$ into $\mathcal{W}_1$ and $\mathcal{W}_2$ in our algorithm, the size of $\mathcal{W}_2$ is at most k/P. Therefore, by setting $k/P \le \varepsilon M \le \varepsilon|\mathcal{U}_i|$, we can afford to select all sets in $\mathcal{W}_2$. Notably, this modification relaxes the required condition of [69] from "$kR/P \le \varepsilon M$" to "$k/P \le \varepsilon M$".

- **Handling low-degree Steiner vertices.** Finally, we need to solve $(\mathcal{U}_{low}, \mathcal{W}_1)$. In their approach, this part takes $O(RP \cdot RPk)$ queries, and this is the step where our main improvement comes from.

  Our modification to this step. By our improved bound for set cover (from Section 5.3.2), the query complexity of this part reduces to $\widetilde{O}(k^2/M + RP(n+k))$. More precisely, to simulate the algorithm in Lemma 5.3.17, we do the following.

1. To run the RGMM oracle, we need to sample $\widetilde{O}(1)$ elements from $\mathcal{U}_i$ uniformly at random. Note that each element of $\mathcal{U}_i$ corresponds to a small component (a component with a small number of representatives). To find a small component uniformly at random, we first pick a terminal uniformly at random and run a BFS to determine whether it lies in a small component, and if so, whether it is a representative terminal. If the terminal is a representative and lies in a small component, we choose the corresponding connected component with probability $1/z$ where z is the number of representative terminals in that connected component (note that BFS returns this number as well). This approach ensures that each small connected component has an equal probability of being sampled. Moreover, due to the bound on the number of small connected components, i.e., $|\mathcal{U}_i| \geq M$, we expect to encounter a terminal in a small connected component every $\widetilde{O}(k/M)$ samples. Therefore, the total cost of running all these BFS subroutines is $\widetilde{O}(k^2/M)$, since each BFS takes $\widetilde{O}(k)$ time.
2. For a small component (a vertex in graph H of Lemma 5.3.17), we need to identify all the Steiner nodes within a distance of at most τ (which is set roughly as $(3/5)(1+\varepsilon)^i$). This step can be completed in $\widetilde{O}(n)$ time. A similar step with the same time complexity also appears in the proof of Lemma 5.3.17.
3. Note that the number of Steiner nodes within this close distance is at most R. Let $\hat{\mathcal{S}}$ be the set of these Steiner nodes. Next, the RGMM algorithm requires a random neighbor of a small component. To get such a neighbor, we keep picking pairs (v,t) in a random order, where $t \in T_{low}$ and $v \in \hat{\mathcal{S}}$, and querying their distance. The first time that we find a pair (v,t) in close distance, we run a BFS from t to check if it is a representative terminal and if it lies in a small component, which takes $\widetilde{O}(k)$ time. If it is not a representative terminal or does not lie in a small connected component, we skip this terminal. Otherwise, we return its small connected component with probability $1/z$, where z is the number of representative terminals in that connected component, ensuring that all neighbors have an equal probability of being selected. We run the above procedure until we find a random neighbor. Therefore, using the running time from Lemma 5.3.17 (substituting α and β for R and P), and considering that we run the BFS at most $\widetilde{O}(RP)$ times (which corresponds to the maximum degree of H), we obtain an $\widetilde{O}(RP(n+k))$ time algorithm to estimate the size of the matching.

Simulation of Step 3 and its query complexity. As we are following the exact implementation of [69] for this step, the additional query complexity of this step (compared to the Step 2) is equal to $\widetilde{O}(nk/M)$ for both their algorithm and our algorithm.

Query Complexity Analysis and Proof of Theorem 5.3.4

For completeness, we start with the query complexity analysis of [69].

Analysis of the query complexity of the algorithm of [69]. As computed in the implementation of the algorithm subsection, the overall query complexity of their algorithm is bounded by

$$\widetilde{O}(n\kappa + \frac{Mk}{\varepsilon} + \frac{nM}{\varepsilon} + \frac{k^2}{M} + \frac{nk}{R} + \frac{nk}{P} + \frac{k^2}{M} + (RP)^2 k + \frac{nk}{M})$$

$$= \widetilde{O}(n\kappa + nM + \frac{nk}{R} + \frac{nk}{P} + (RP)^2 k + \frac{nk}{M}). \qquad \triangleright\, \varepsilon = O(1), k \le n$$

Furthermore, the conditions that need to be satisfied are

- $k \le \kappa$, or
- $k > M$ and $n/R < \widetilde{O}(\varepsilon M)$ and $k > \widetilde{\Omega}(P)$ and $kR/P \le \varepsilon M$.

The query complexity of their algorithm under the above conditions can be optimized by setting $\kappa = M = n^{6/7}$, $R = n^{1/7}$, and $P = n^{2/7}$, which gives the total query complexity of $\widetilde{O}(n^{13/7})$.

Now, we prove the main theorem of this section.

Proof of Theorem 5.3.4. As we are implementing the same algorithm as [69], except replacing their set cover subroutine with a more efficient algorithm, the approximation analysis follows exactly from their proof. It only remains to bound the query complexity of our proposed algorithm for metric Steiner tree using the improved sublinear algorithm for set cover. In the implementation of the algorithm subsection, we analyzed the query complexity of the component that is implemented differently in our algorithm. Now, we put the query complexity of all parts together and compute the overall complexity.

Analysis of the query complexity of our algorithm. The overall query complexity of our algorithm is bounded by

$$\begin{aligned} &\widetilde{O}(n\kappa + \frac{Mk}{\varepsilon} + \frac{nM}{\varepsilon} + \frac{k^2}{M} + \frac{nk}{R} + \frac{nk}{P} + \frac{k^2}{M} + RP(k+n) + \frac{nk}{M}) \\ &= \widetilde{O}(n\kappa + nM + \frac{nk}{R} + \frac{nk}{P} + RPn + \frac{nk}{M}) \qquad \triangleright\, \varepsilon = O(1), k \le n \end{aligned}$$

Note again that the main difference is that the term P^2R^2k is replaced by RPn. Furthermore, the conditions that need to be satisfied are also slightly more relaxed, as follows:

- $k \le \kappa$, or
- $k > M$ and $n/R < \widetilde{O}(\varepsilon M)$ and $k > \widetilde{\Omega}(P)$ and $k/P \le \varepsilon M$.

Specifically, "$kR/P \le \varepsilon M$" is replaced by "$k/P \le \varepsilon M$". Then, our algorithm can be optimized by setting $\kappa = M = n^{2/3}, R = \widetilde{\Theta}(P) = \widetilde{\Theta}(n^{1/3})$ which gives the total query complexity of $\widetilde{O}(n^{5/3})$. □

5.4 Sublinear Algorithm for Steiner Forest

In this section, we present a sublinear-time algorithm for the Metric Steiner Forest problem. For completeness, we restate some of the key definitions below.

Definition 5.4.1 (Sublinear Metric Steiner Forest)**.** *In the metric Steiner Forest problem, we are given a set of points V, a set of k terminal pairs $T = \{(s_1, t_1), \ldots, (s_k, t_k)\} \subseteq V \times V$, and query access to the $|V| \times |V|$ distance matrix of a metric space (V, w), where an oracle query for (u, v) returns the weight $w(u, v)$ of the edge (u, v).*

Let $\mathsf{SF}(V, T, w)$ denote the minimum weight of a Steiner Forest on instance (V, T, w). Then, the goal is to design an algorithm that estimates $\mathsf{SF}(V, T, w)$ using $o(n^2)$ queries to the distance matrix via the oracle.

More specifically, we prove the following results in this section.

Theorem 5.4.2. *There exists an algorithm for estimating the cost of Steiner Forest within a multiplicative factor of $O(\log k)$ using $O(T_{\text{MIS}} \cdot \log k) = \widetilde{O}(n^{3/2})$ queries to the distance matrix oracle. Here, T_{MIS} denotes the best runtime of a sublinear algorithm for finding a multiplicative $O(1)$-approximation for the size of any MIS under the adjacency matrix model.*

Theorem 5.4.3. *For any $\varepsilon \in (0,1)$ there is an algorithm (Algorithm 37) that, given a graph (V,E) with oracle access to its adjacency matrix, with high probability reports a multiplicative $(1+\varepsilon)$-approximation to the value $|\text{RGMIS}(\pi)|$ for some permutation π of V and runs in $\widetilde{O}(n^{3/2}/\varepsilon^2)$ time.*

5.4.1 Technical Overview

A natural framework one would first try for solving this problem is the approach used in [65, 75] for estimating the weight of an MST; as we describe below, it fundamentally does not suffice for the Steiner Forest problem. Consider the subgraph consisting of all edges of weight up to some threshold τ, and let c be the number of connected components in this graph. Then any MST needs to use *exactly* $c-1$ edges of weight larger than τ. This intuition was formalized in these works, giving the equality $\text{MST} \approx n - W + \varepsilon \sum_{i=0}^{n} (1+\varepsilon)^i c_i$, where c_i is the number of connected components of the graph where we set the threshold to $\tau_i = (1+\varepsilon)^i$. Then these prior works focus on estimating c_i and providing a rigorous proof that the approximation they can get for c_i is sufficient for the ultimate task of estimating MST.

There is an equivalent intuition for the Steiner Forest problem when we consider the threshold graph parameterized by τ, with two differences.

- First, c should now be the number of *active* connected components; a component is active if for some pair (s_j, t_j) one of the two vertices is inside the component and the other one is outside.
- Second, the number of edges of weight more than τ used in the optimal solution is now at least $c/2$ and at most $c-1$, because not all active components will need to be connected in the end – we might only need a matching between these active components.

Thus, a potential algorithm would be to instead estimate the number of active connected components c_i for each threshold τ_i and follow the approach of [75]. Note that, roughly speaking, the second item above does not cause much trouble, as it only results in a factor-2 blow-up in the approximation.

Challenge in estimating the number of active components. A common approach for counting the number of connected components in a graph is to sample a random vertex and start growing a ball from that vertex (e.g. using BFS) until either the entire component is visited, in which case we increment our estimate of the number of components, or the total number of visited vertices reaches a threshold, in which case we can ignore that component and show that the error introduced by ignoring such large components is small. While this is the core idea, the details required to obtain a multiplicative approximation in [75] are more involved. In particular, they run a modified BFS procedure where very close vertices are treated as a single vertex, allowing for further improvements.

To mimic this approach, first of all, one has to sample a random terminal as opposed to a random vertex, to ensure that unrelated parts of the graph are not selected most of the time. However, one challenge that does not allow us to follow this approach is that when growing a ball from a terminal (or a vertex in general), the total time spent depends on the *volume* of the component that we explore, i.e., the number of vertices we visit. For the MST problem, this volume can be related to the cost that an optimal solution must pay within that component, since ultimately all vertices must be connected.

However, this does not hold for the Steiner Forest problem. When we grow a ball around a terminal, the optimal solution might only select a single path from the center to the boundary of the ball to connect the terminal to its pair. At a high level, the cost of such search algorithms depends on the *volume* of the ball, whereas the optimal solution for Steiner Forest might only pay the cost of the ball's *radius*.

Our new approach. In this work, we adopt a different approach than counting the number of active components, which follows a similar intuition but can be implemented in sublinear time. We consider balls of radius τ around all terminals, and call a ball *active* if the matching vertex of the center of the ball falls outside of it. The advantage is that checking whether a ball is active or not under this new definition requires only a single query. Now, if we take a set of M disjoint active balls, then any optimal solution needs to pay a cost of at least $M \cdot \tau$ (to buy a path from the center to the boundary of each ball).

Therefore, instead of counting the number of active components, our algorithm seeks to find a *maximal independent set* on the set of active balls for a certain threshold τ. This is where we use the sublinear algorithm for MIS on the graph defined on the set of active balls, and output $\sum_i M_i \cdot \tau_i$ for exponentially increasing thresholds τ_i. We show that this achieves an $O(\log k)$-approximation for Steiner Forest.

To that end, we note that each terminal in the MIS M_i induces a certain cluster of low diameter $O(\tau_i)$. We show how to inductively construct a solution that internally connects every such cluster, for increasing i, while paying at most $M_i \cdot \tau_i$ for each threshold τ_i. This hierarchically built partial solution then serves as a scaffolding to connect the demands. We assign each demand pair carefully to a specific level i. Then, for each i, we show that one can connect all assigned demands within the same budget of $M_i \cdot \tau_i$ by selecting a spanning forest in a suitably defined auxiliary graph.

Finally, we also show that this MIS-based approach cannot lead to a better than $O(\log k)$-approximation for Steiner Forest.

Challenges for MIS. Let us now describe the main challenges that arise when developing a sublinear-time algorithm for MIS, and outline our key ideas for addressing them.

Recall from the introduction above that a direct implementation of the RGMIS oracle [176] in the adjacency matrix model would result in an expected runtime of $O(\Delta n)$ for a single vertex query. The two major issues are that this is not sublinear-time for dense graphs, and that it only returns a single bit of information about a single vertex, whereas the MIS size can range from 1 to n. We can thus obtain a $(1, \varepsilon n)$ multiplicative-additive approximation by making $\widetilde{O}(1/\varepsilon^2)$ queries, but a purely multiplicative approximation seems to require $\Omega(n)$ queries (to e.g. differentiate between different constant MIS sizes).

Even in the adjacency list model, the query complexity analysis of [176] does not explicitly translate into an algorithm. The main challenge lies in generating the neighbors of a vertex in increasing order of their ranks in the random permutation π. Achieving this efficiently seems difficult if the goal is to spend only $o(n)$ time per step to obtain the next neighbor in the ordered list. Moreover, approaches that rely on this

query complexity analysis to design algorithms for maximum matching [34] cannot be applied here. In the case of matching, each edge has only two endpoints, and revealing its rank requires "notifying" just these two endpoints. However, for MIS, we would need to notify up to $\Omega(n)$ neighbors in the worst case.

Our MIS approach. As a warm-up let us first describe a simpler approach that leads to an algorithm with $\widetilde{O}(n^{5/3})$ runtime. Our first insight is that, rather than using an oracle as above, an RGMIS can be also built directly bottom-up by iterating over vertices in the permutation π; when a vertex is visited, we add it to the RGMIS and disqualify all its neighbors from further consideration. We can perform s steps of this process in time $O(ns)$, and it will result in explicitly building the entire RGMIS if its size is at most s. Let us run it for $s = \widetilde{O}(n^{2/3})$ steps. One can then prove that with high probability, the maximum degree in the remaining graph is at most $n^{1/3}$; thus the vertex oracle [176] runs in time $O(n^{4/3})$. Then, $\widetilde{O}(n^{1/3}/\varepsilon^2)$ queries to this oracle will result in an $(1+\varepsilon, O(\varepsilon n^{2/3}))$ multiplicative-additive error, which gives a purely multiplicative approximation for the entire graph because the RGMIS size is now known to be at least $n^{2/3}$.

In order to improve upon the above and obtain our final running time of $\widetilde{O}(n^{3/2})$, we show a different implementation of the RGMIS vertex oracle in the adjacency matrix model (Algorithm 35), and we prove that it has an expected running time of $O(n)$ instead of $O(\Delta n)$, even in dense graphs. We do this via a probabilistic argument in which we couple the execution of our Algorithm 35 on the original input graph to the execution of the Yoshida, Yamamoto and Ito oracle (Algorithm 34) on a certain larger graph, and relate the number of adjacency matrix oracle calls of the former to the number of recursive calls of the latter. This is possible as long as the random permutation on this larger graph has a favorable structure; we show that this happens with at least a constant probability via a connection to a combinatorial ballot voting problem whose study dates back to the 19th century [173, 52].

5.4.2 Sublinear Metric Steiner Forest

In this section, we will prove Theorem 5.4.2. We assume that we have access to an algorithm for MIS that gives a multiplicative $O(1)$-approximation in the adjacency matrix model. Theorem 5.4.3, shown in Section 5.4.4, provides such an algorithm.

Notation. We will use the term *terminal* to refer to any of the vertices in $\bigcup_{i\le k}\{s_i, t_i\}$, and will abuse notation and use T to also refer to all terminals when clear from the context. We use *terminal pairs* to refer specifically to the pairs (s_i, t_i). For any terminal in T, we define its *match* as the other vertex in the pair, i.e., $m(s_i) = t_i$ and $m(t_i) = s_i$. Moreover, for ease of notation, we assume that terminal pairs are disjoint. Note that this is without loss of generality by creating copies of original vertices, where each pair (s_i, t_i) uses distinct copies of the corresponding two vertices.

Finally, we use OPT to denote the value of the optimal Steiner Forest connecting all given terminal pairs.

Preprocessing. First, we apply a standard preprocessing step to bound the approximation factor in terms of $\log k$ rather than $\log W$. Let $X = \max_{i\le k} w(s_i, t_i)$, which can be computed using $O(k)$ queries. It is straightforward to show that $X \le \mathrm{OPT} \le k \cdot X$. Now, we will ignore all terminal pairs (s_i, t_i) such that $w(s_i, t_i) \le X/k$. At the end of the algorithm we can connect these pairs directly; since there are at most k such pairs, their additional cost will be at most X. Thus, they contribute only an additive term of 1 to the approximation factor. Also note that we do not need to consider using edges whose weights are larger than

kX. By scaling, from now on we will assume that the minimum terminal pair has distance 2, and thus the maximum weight of an edge that we would want to ever pick in our solution is at most $2k^2$.

Threshold graph. Our algorithm will consider several thresholds $\tau_i = 2^i$ for i from 0 to $L = \log(2k^2) = O(\log k)$. We will use the *threshold graph* $G_i = (V, E_i)$ to refer to an unweighted subgraph of G where $(u, v) \in E_i$ if $w(u, v) \leq \tau_i$.

Active terminals and ball graphs. We will say that a terminal $s \in T$ is *active* in level $i \leq L$ if $w(s, m(s)) \geq \tau_i$. Now we define an unweighted *ball graph* $H_i = (V_i^H, E_i^H)$, where $V_i^H \subseteq T$ is the set of active terminals in level i. More precisely, we are implicitly thinking of V_i^H as the set of balls of radius τ_i around each active terminal. Now two vertices $u, v \in V_i^H$ have an edge between them if their corresponding balls collide, i.e., $(u, v) \in E_i^H$ iff $w(u, v) < 2\tau_i$.

Algorithm. Now our algorithm is as follows. For each $0 \leq i \leq L$, let M_i be the size of *any* MIS in H_i, and let $\hat{M}_i$ be a multiplicative estimate of it given by the algorithm of Theorem 5.4.3, i.e., $M_i \leq \hat{M}_i \leq c_{\text{MIS}} \cdot M_i$ (we can set c_{MIS} to be e.g. 1.01). Our algorithm will output $\text{SOL} = \sum_{i=0}^{L} \hat{M}_i \cdot \tau_i$.

Lemma 5.4.4. *For any $i \leq L$, we have* $\text{OPT} \geq M_i \cdot \tau_i$.

Proof. Consider an MIS in H_i whose size is M_i; let us denote it by $U_i \subseteq V_i^H$. Now take any $u \in U_i$ and consider the ball B_u of radius τ_i around u. Given that u is active in level i, it means that in order to reach from u to its match $m(u)$, intuitively, any Steiner Forest solution has to pick a path of length at least τ_i inside B_u.

More concretely, for each u we consider any path P_u connecting u and $m(u)$ in the solution. We would like to say that P_u contains a subpath of length at least τ_i that lies inside B_u. Since these balls are disjoint (as U_i is an MIS in H_i, the distance between any two ball centers is at least $2\tau_i$), so are the subpaths, which implies that we get $\text{OPT} \geq M_i \cdot \tau_i$.

However, strictly speaking P_u might not contain such a subpath, as we are dealing with a discrete metric space. For such u, for the sake of this analysis only, we consider the first edge (v_1, v_2) of P_u that exits B_u (i.e., $w(u, v_1) < \tau_i$ but $w(u, v_2) \geq \tau_i$). We subdivide the edge (v_1, v_2) into two edges (v_1, v') and (v', v_2), with the length of (v_1, v') chosen such that the length of the subpath of P_u from u to v' becomes exactly τ_i. We can now select this subpath for u, as it lies inside B_u (and thus will be disjoint from any other subpath). □

Lemma 5.4.5. *We have* $\text{OPT} \leq 6 \cdot \sum_{i=0}^{L} M_i \cdot \tau_i$.

Before we prove Lemma 5.4.5, let us first use it to finish the proof of Theorem 5.4.2.

Proof of Theorem 5.4.2. As for the approximation factor, note that by Lemma 5.4.5 we have

$$\text{OPT} \leq 6 \cdot \sum_i M_i \cdot \tau_i \leq 6 \cdot \sum_i \hat{M}_i \cdot \tau_i = 6 \cdot \text{SOL}.$$

Also, by Lemma 5.4.4 we have

$$\text{SOL} = \sum_i \hat{M}_i \cdot \tau_i \leq c_{\text{MIS}} \cdot \sum_i M_i \cdot \tau_i \leq c_{\text{MIS}} \cdot (L+1) \cdot (\max_i M_i \cdot \tau_i) \leq c_{\text{MIS}} \cdot (L+1) \cdot \text{OPT} \leq O(\log k) \cdot \text{OPT}.$$

Thus, our solution is an $O(\log k)$-approximation of OPT.

As for the runtime, our algorithm estimates the MIS size for each of the $L+1 = O(\log k)$ threshold values using the algorithm of Theorem 5.4.3, and thus has a total runtime of $\widetilde{O}(n^{3/2})$. Note that the set of vertices V_i^H can be computed using $O(k)$ queries. Further, checking whether two vertices have an edge in E_i^H can be done by one query to w, and thus we have an adjacency matrix oracle access for H_i. □

The rest of this section is devoted to the proof of Lemma 5.4.5. We start by introducing further notation.

Clusters and centers. Let $U_i \subseteq V_i^H$ denote the MIS whose size is M_i. For each active vertex $v \in V_i^H$, we define its *center* $c_i(v) = \arg\min_{u \in U_i} w(v,u)$ to be the closest point in the MIS U_i to v, breaking ties arbitrarily. Note that by maximality of U_i we have $w(v, c_i(v)) < 2\tau_i$. Furthermore, for each $u \in U_i$ we define the *cluster* of u, i.e., $C_i(u) = \{v \in V_i^H : c_i(v) = u\}$ as the set of active terminals whose center is u at level i. Finally, we use $\mathcal{C}_i = \{C_i(u) : u \in U_i\}$ to denote the set of all clusters of level i.

First we show that we can connect all clusters internally by paying at most $O(\sum_i M_i \cdot \tau_i)$.

Lemma 5.4.6 (Connectivity within clusters)**.** *There exists a subset of edges $F \subseteq V \times V$ of total cost at most $4 \cdot \sum_i M_i \cdot \tau_i$ such that any pair of vertices v_1, v_2 inside each cluster in $\mathcal{C}_0 \cup \cdots \cup \mathcal{C}_L$ is connected in the graph $G(V, F)$.*

Proof. We let $F_0 = \emptyset$. We will construct sets of edges $F_1, \ldots, F_L$ such that

- (Cost Constraint) for each $i \in \{1, \ldots, L\}$, the cost of F_i, defined as $\sum_{(x,y)\in F_i} w(x,y)$, is at most $4M_i\tau_i$,

while inductively maintaining the following property:

- (Connectivity Property) for $0 \le i \le L$, any pair of vertices v_1, v_2 both belonging to any cluster in $\mathcal{C}_0 \cup \cdots \cup \mathcal{C}_i$ is connected in the graph $G(V, F_0 \cup \cdots \cup F_i)$.

Setting $F = \bigcup_{i=0}^{L} F_i$, the above cost constraint and connectivity property imply the statement of the lemma.

As the base case, let us ensure the connectivity property for $i = 0$. Note that, by the preprocessing stage, we know that the minimum pairwise distance between any two active terminals is at least 2. Given that $\tau_0 = 1$, the graph H_0 has no edges, and thus the only MIS in this graph contains all vertices in V_0^H. Therefore, all clusters in $\mathcal{C}_0$ have size 1 and thus the property holds trivially for $i = 0$.

Next, suppose that the connectivity property holds up to level $i = \ell$. This in particular means that all vertices inside each cluster of $\mathcal{C}_\ell$ are already connected in $G(V, F_0 \cup \cdots \cup F_\ell)$. We will now show how to pick $F_{\ell+1}$ with cost at most $4M_\ell \cdot \tau_\ell$, satisfying the cost constraint, such that the vertices inside each cluster of $\mathcal{C}_{\ell+1}$ will become connected in $G(V, F_0 \cup \cdots \cup F_{\ell+1})$.

Consider an undirected simple graph G' where we merge each cluster of $\mathcal{C}_\ell$ into a supernode. More precisely, for each vertex in U_ℓ, there is a vertex in G'. Now, we connect two supernodes corresponding to $u_1, u_2 \in U_\ell$ with an edge in G' if there exists a vertex $v \in C_\ell(u_1)$ and a vertex $u \in C_\ell(u_2)$ such that $c_{\ell+1}(v) = u$; i.e., on the next level, the center of some point v in one cluster $C_\ell(u_1)$ will be a point u in the other cluster $C_\ell(u_2)$.

For the connectivity property to hold for level $\ell+1$, we need to make sure that v and u get connected after adding $F_{\ell+1}$. Given that both clusters $\mathcal{C}_\ell(u_1)$ and $\mathcal{C}_\ell(u_2)$ are already internally connected in $G(V, F_0 \cup \cdots \cup F_\ell)$,

to connect u and v it will be enough to connect $\mathcal{C}_\ell(u_1)$ to $\mathcal{C}_\ell(u_2)$ in any way. Intuitively, an edge in G' means that we need to connect the supernodes corresponding to its endpoints (possibly indirectly).

Note that because u belongs to the MIS $U_{\ell+1}$ and serves as the center for v, i.e., $c_{\ell+1}(v) = u$, we have that u and v are connected in $H_{\ell+1}$, and thus their distance is at most $2\tau_{\ell+1}$. This shows that if (u_1, u_2) is an edge in G', then the distance between $\mathcal{C}_\ell(u_1)$ and $\mathcal{C}_\ell(u_2)$ is at most $2\tau_{\ell+1}$.

Now we state how to construct $F_{\ell+1}$: We pick a spanning forest in G' and for each edge (u_1, u_2) in the spanning forest, we add the closest pair of vertices in the clusters of the corresponding supernodes, i.e., $C_\ell(u_1)$ and $C_\ell(u_2)$, to $F_{\ell+1}$.

First, note that if there is an edge between two supernodes in G', then they are in the same connected component of G', and because we pick a spanning forest, the two supernodes will become connected by $F_{\ell+1}$. Thus all connectivity requirements for the clusters in $\mathcal{C}_{\ell+1}$ will be satisfied and we will have the connectivity property for level $\ell + 1$. Second, the total number of edges we pick in the spanning forest is at most $|U_\ell| - 1 < M_\ell$, and the cost of each edge added to $F_{\ell+1}$ is at most $2\tau_{\ell+1} \leq 4\tau_\ell$. Thus, the total cost of $F_{\ell+1}$ is at most $4M_\ell\tau_\ell$, satisfying the cost constraint. □

Next, we need to connect each terminal $s \in T$ to its match, $m(s)$, using the available budget $O(\sum_i M_i \cdot \tau_i)$. Again, we define further notation.

Target terminals. Consider a terminal $s \in T$ such that $s \in V_i^H$ but $s \notin V_{i+1}^H$. The goal is to connect all such terminals s to their matches $m(s)$ using the budget of level i, i.e., $O(M_i \cdot \tau_i)$. Thus we define the set of *Target* terminals as $\text{Target}_i = V_i^H \setminus V_{i+1}^H$. Further, for $u \in U_i$ in the MIS, let $\text{Target}_i(u) = \text{Target}_i \cap C_i(u)$ be the target set inside the cluster of u.

Lemma 5.4.7 (Connecting target pairs)**.** *For every $i = 0, ..., L$, there exists a subset of edges $J_i \subseteq V \times V$ of total cost at most $2M_i \cdot \tau_i$ such that all terminals in Target_i are connected to their match in the graph $G(V, F \cup J_i)$. Here $F \subseteq V \times V$ is the subset of edges given by Lemma 5.4.6.*

Proof. We will again define an undirected simple graph G' where we merge each cluster of $\mathcal{C}_i$ into a supernode. Note that because we have included the set F in $G(V, F \cup J_i)$, all the nodes within a supernode are already connected.

The vertex set of G' is defined similarly as in the proof of Lemma 5.4.6, where for each vertex in U_i, there is a vertex in G'. However the edges are now defined differently. Two supernodes corresponding to $u_1, u_2 \in U_i$ are connected in G' if there exists a vertex $s \in \text{Target}_i(u_1)$ such that its match is in the other cluster, i.e., $m(s) \in C_i(u_2)$.

Our goal is to pick a set of edges J_i such that the pair $(s, m(s))$ gets connected. Given that the clusters are connected internally by Lemma 5.4.6 using F, it is enough to ensure that we connect the supernodes corresponding to u_1 and u_2 (possibly indirectly) whenever there is an edge between them in G'.

Further, note that picking an edge in G' can be implemented by adding an edge to J_i with a cost of at most $2\tau_i$. This is because s and $m(s)$ are in Target_i and thus are not in V_{i+1}^H. This means that their distance is at most $\tau_{i+1} \leq 2\tau_i$.

The rest of the proof follows that of Lemma 5.4.6. We state how to construct J_i: Again we pick a spanning forest in G' and for each edge (u_1, u_2) in the spanning forest, we add the closest pair of vertices in the clusters of the corresponding supernodes, i.e., $C_i(u_1)$ and $C_i(u_2)$, to J_i.

First, note that if there is an edge between two supernodes in G', then they are in the same connected component of G', and because we pick a spanning forest, the two supernodes will get connected by adding J_i. Thus all connectivity requirements for the pairs in Target_i are satisfied. Second, the total number of edges we pick in the spanning forest is at most $|U_i| - 1 < M_i$, and the cost of each edge is at most $2\tau_i$. Thus the total cost of J_i is bounded by $2M_i\tau_i$. □

Proof of Lemma 5.4.5. Note that for each terminal pair (s_j, t_j) there exists an index $i \leq L$ for which $s_j \in \text{Target}_i$. Therefore, if we use Lemma 5.4.7 for all values of $0 \leq i \leq L$, then all terminals in T will be connected to their match in the graph $G(V, F \cup J_0 \cup \cdots \cup J_L)$. Thus, OPT is at most the cost of $F \cup J_0 \cup \cdots \cup J_L$, which by Lemma 5.4.6 and Lemma 5.4.7 is at most $6 \cdot \sum_i M_i \cdot \tau_i$, as desired. □

5.4.3 Tightness of the MIS Approach for Steiner Forest

Here, we show that the $O(\log k)$-approximation analysis of our MIS-based estimate for Steiner Forest is tight.

Lemma 5.4.8. *The estimate $\sum_i M_i\tau_i$ cannot get better than $O(\log k)$-approximation for the Steiner Forest estimation problem.*

Proof. We provide two instances I_1, I_2, such that on I_1, the value of the estimate is of $O(\text{OPT}(I_1))$ and on I_2, the value of estimate is of $O(\text{OPT}(I_2) \cdot \log k)$. Since we require the estimate to always be greater than the optimal Steiner Forest cost, our MIS-based estimate, possibly after a fixed scaling, cannot guarantee better than $O(\log k)$-approximation. In both instances, there are $n = 2^L - 1$ vertices $v_1, \ldots, v_n$ and the number of terminal pairs k is $O(n)$.

Construction of I_1. In this instance, these n vertices are located on a line (i.e., one-dimensional Euclidean space) where the distance of any consecutive pair of vertices is exactly 2. Formally, for every $i, j \leq n$, $d(v_i, v_j) = 2(i - j)$. Finally, the terminal pairs are $\{(v_i, v_{i+n/2})\}_{i \leq n/2}$.

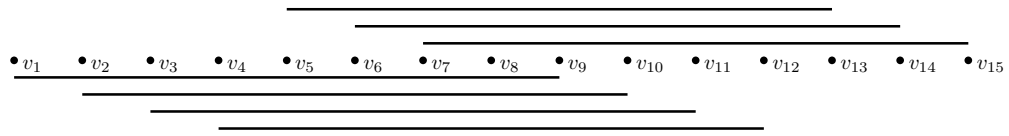


Figure 5.5: An example of I_1 with 15 vertices. The endpoints of line segments denote terminal pairs.

For any level $j < L$ in the MIS based algorithm, all pairs are active and it is straightforward to check that the size of every MIS is $O(n/2^j)$. Then, the estimate will have cost at least $\sum_{j<L} 2^j \cdot O(n/2^j) = O(nL) = O(n \cdot \log k)$. However, the optimal Steiner Forest on this instance has cost $O(n)$.

Construction of I_2. In this instance, the vertices are partitioned into L disjoint clusters $\mathcal{V}_1, \ldots, \mathcal{V}_L$ such that for every $i \leq L$, the cluster $\mathcal{V}_i$ contains $n/2^i$ vertices $v^i_1, \ldots, v^i_{n/2^i}$. In each cluster $\mathcal{V}_i$, vertices are located on a line where the distance of every consecutive pair of vertices is 2^{i+1}. Moreover, for every $i, j \leq L$, the minimum distance between any two clusters $\mathcal{V}_i$ and $\mathcal{V}_j$ is $M \gg n$. Finally, the terminals are of the form $\{(v^i_1, v^i_2), \ldots (v^i_{n/2^i-1}, v^i_{n/2^i})\}_{i \leq L}$.

For any level $j < L$ in the MIS based algorithm, all pairs in clusters $\mathcal{V}_j, \ldots, \mathcal{V}_L$ are active and it is straightforward to check that the size of every MIS is $O(n/2^j)$. Then, the estimate will have cost $\sum_{j<L} 2^j \cdot$

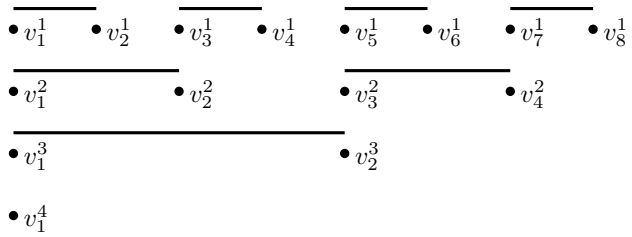


Figure 5.6: An example of I_2 with 15 vertices. The endpoints of line segments denote terminal pairs.

$O(n/2^j) = O(nL) = O(n \cdot \log k)$. Moreover, the optimal Steiner Forest cost is $\sum_{j \le L} 2^j \cdot n/2^j = O(nL) = O(n \cdot \log k)$. □

5.4.4 Sublinear Time MIS under the Adjacency Matrix Model

In this section, we design a sublinear time algorithm for estimating the MIS size to a multiplicative $1 + \varepsilon$ approximation factor.

Definition 5.4.9. (Permutations)

- *For a set V we denote by $S(V)$ the set of permutations of V, i.e., bijections from $\{1, ..., |V|\}$ to V.*
- *For a subset $V' \subseteq V$ and a permutation $\pi \in S(V)$ we define the* restriction of π to V'*, denoted by $\pi[V']$, as the permutation of V' obtained from V by keeping only those elements that belong to V' and renumbering appropriately (i.e., $\pi[V'](i)$ is the i-th element of V' among $\pi(1), \pi(2), ...$).*

Definition 5.4.10. *A maximal independent set (MIS) is a subset $V' \subseteq V$ of vertices such that for any edge $(u, v) \in M$, at most one of u, v is in V', and for any $v \in V \setminus V'$, some neighbor of v is in V'.*

Definition 5.4.11. *Given a permutation $\pi \in S(V)$, start with an empty set $I := \emptyset$, and for $i = 1, ..., n$, add vertex $\pi(i)$ to I if none of its neighbors are in I. The set* RGMIS(π) *is defined as the resulting set I.*

Fact 5.4.12. *For any $\pi \in S(V)$, the set* RGMIS(π) *is an MIS. Moreover, for any $v \in V$, we have that $v \in$* RGMIS(π) *if any only if for every neighbor u of v of lower rank in π (i.e., $\pi^{-1}(u) < \pi^{-1}(v)$) we have $u \notin$* RGMIS(π).

The following oracle $\mathcal{A}$, proposed by Nguyen and Onak [154] and analyzed by Yoshida, Yamamoto and Ito [176], answers the query of whether a vertex v is in RGMIS(π). We call it an "abstract" oracle since it does not specify how to implement Line 1, and we will only use the bound on its query complexity (i.e., number of recursive calls to $\mathcal{A}$) given by Theorem 5.4.13.

Algorithm 34: $\mathcal{A}(\pi, v)$ (abstract RGMIS oracle [154, 176])

```
1 let u_1, u_2, ..., u_k be all neighbors of v, sorted in increasing order of their ranks in π
2 for i = 1, ..., k do
3     if u_i has lower rank than v in π then
4         if A(π, u_i) then
5             return false
6 return true
```

Theorem 5.4.13 ([176]). *Let $T_{\mathcal{A}}(\pi, v)$ be the total number of recursive calls to $\mathcal{A}$ for a query $\mathcal{A}(\pi, v)$. Then we have*

$$\mathbf{E}_{v \in V, \pi \in S(V)} [T_{\mathcal{A}}(\pi, v)] \leq 1 + \frac{|E|}{|V|}.$$

We remark that the right-hand side is at most $O(\Delta)$, and that the bound is in expectation over both π and v.

Single-sample algorithm

As discussed in the introduction, a straightforward implementation of Line 1 in Algorithm 34 would result in a running time of $O(\Delta n)$. In this subsection we describe our more efficient implementation of the vertex oracle in the adjacency matrix model, Algorithm 35. We note that if we are satisfied with a running time of $O(n)$, we can materialize the entire random permutation π. Then, when the oracle is called for a vertex v and needs to iterate over its neighbors in increasing order of rank, we loop over vertices $\pi(1)$, $\pi(2)$, $\pi(3)$, ..., and check whether each of them is a neighbor of v.

Algorithm 35: $\mathcal{O}(\pi, v)$ (our MIS oracle for the adjacency matrix model)

```
1 let k = π^{-1}(v) be the rank of v in π
2 for i = 1, ..., k − 1 do
3     if (π(i), v) ∈ E then
4         if O(π, π(i)) then
5             return false
6 return true
```

In terms of the recursive calls being made and the results returned, the oracles $\mathcal{O}$ and $\mathcal{A}$ (Algorithms 34 and 35) are clearly equivalent. However, one might worry that $\mathcal{O}$ (Algorithm 35) potentially spends a lot of time querying non-edges on Line 3.

Intuitively, for a vertex u of degree $\deg(u)$, it takes expected $n/\deg(u)$ time to find a neighbor of u if we queried the adjacency matrix at random. Vertex degrees in a graph can vary widely, but this is not an issue: the proof of Theorem 5.4.13 in [176] in fact shows that for any vertex u, the expected number of recursive calls out of $\mathcal{A}(\pi, u)$ is at most $\deg(u)/n$ for a random "top-level" query vertex v. Thus, one could hope that if finding one neighbor of u to call recursively took $n/\deg(u)$ time (i.e., queries to the adjacency matrix), then the total runtime would be given by a calculation such as $\sum_{u \in V} \frac{\deg(u)}{n} \cdot \frac{n}{\deg(u)} = O(n)$.

However, such an argument is invalidated by the fact that both the trajectory of the recursive oracle calls (to $\mathcal{O}$) and the sequence of adjacency matrix calls are functions of the random permutation π, and thus seem highly correlated.[5] In fact, every vertex u queries the same vertices $\pi(1)$, $\pi(2)$, ... to find its neighbors. Nevertheless, we are still able to prove a running time bound of $O(n)$, albeit by employing a different argument in the proof.

Lemma 5.4.14. *Let $T_{\mathcal{O}}(\pi, v)$ be the total runtime of Algorithm 35 (including all recursive calls) for query $\mathcal{O}(\pi, v)$. Then we have*

$$\mathbf{E}_{v \in V, \pi \in S(V)} [T_{\mathcal{O}}(\pi, v)] = O(n).$$

[5] An argument as outlined above was recently used by [148] in the context of maximal matchings. However, their algorithm uses fresh, independent randomness to query random entries of the adjacency matrix to find neighbors of the current vertex, which enables the argument.

We remark that if one is not worried about logarithmic factors, then the strategy above can still be executed to obtain a $\widetilde{O}(n)$ bound, which one could argue is simpler than our proof below, with the caveat that this would also require modifying Algorithm 35 to introduce caching of the returned answers. See remark 12 for a further discussion.

The rest of this section is devoted to the proof of Lemma 5.4.14. Our plan is to couple the execution of our Algorithm 35 ($\mathcal{O}$) on the original input graph (V, E) to the execution of Algorithm 34 ($\mathcal{A}$) on a certain larger graph H.

We start by defining the latter. Informally, we start with a copy of (V, E) as a first layer. As the second layer V_2, we adjoin $2n$ new vertices, with no edges inside. As the third layer V_3, we adjoin n new vertices, with no edges inside. Finally, we connect each vertex in the second layer to each vertex in the first and the third layers. Formally:

Definition 5.4.15. *Let H be a graph with vertex set $V(H) = V \cup V_2 \cup V_3$, where V is the vertex set of (V, E), $|V_2| = 2n$, and $|V_3| = n$. We define the edge set $E(H) = E \cup (V \times V_2) \cup (V_2 \times V_3)$.*

Next, we identify requirements on the permutation π_H of vertices of H that will enable our coupling argument.

Definition 5.4.16. *We say that a permutation $\pi_H \in S(V(H))$ is* good *if:*

- *the first vertex comes from V_3, i.e., $\pi_H(1) \in V_3$,*
- *in every prefix of π_H there are at least as many vertices from V_2 as from V.*

We show that a random permutation π_H is indeed likely to be good:

Lemma 5.4.17. *Consider a uniformly random permutation $\pi_H \in S(V(H))$. Then:*

1. $\Pr[\pi_H$ *is good*$] \geq \frac{1}{12}$.
2. *If π_H is a uniformly random good permutation, $\pi := \pi_H[V]$ (π_H restricted to V) is uniformly random in $S(V)$.*

Proof. Note that $\pi_H(1) \in V_3$ with probability $\frac{|V_3|}{|V(H)|} = \frac{1}{4}$, and that conditioning on this event reveals no information about the relative order of vertices in $V \cup V_2$, so we can focus on the restricted permutation $\pi_H[V \cup V_2]$. We now employ a statement known as Bertrand's ballot theorem [173, 52]: in an election where candidate A receives p votes and candidate B receives q votes with $p > q$, if the votes are counted in random order, the probability that A will be strictly ahead of B throughout the count is $\frac{p-q}{p+q}$. This maps to our scenario by setting $p = |V_2| = 2n$ and $q = |V| = n$, resulting in a probability of $\frac{2n-n}{2n+n} = \frac{1}{3}$.

For the second part, note that conditioning on both events reveals no information about the relative order of vertices in V. □

Now assume that $\pi_H \in S(V(H))$ is good, and consider what happens in the RGMIS(π_H) process in H. First, $\pi_H(1) \in V_3$ joins the MIS, and thus removes all V_2 vertices from consideration (as it is connected to all of them). Other V_3 vertices will join the MIS when they arrive. All the remaining action happens on (V, E) according to the permutation $\pi = \pi_H[V]$ (π restricted to V). That is, we have RGMIS(π_H) = $V_3 \cup$RGMIS(π).

Moreover, consider the tree of recursive calls made when executing $\mathcal{O}(\pi, v)$ for a query vertex $v \in V$ (in the original graph (V, E)), and compare it to the tree of recursive calls made when executing $\mathcal{A}(\pi_H, v)$ (in H).

The difference is that in the latter, there will be some *additional* calls from a vertex in V to $\mathcal{A}(\pi_H, v_2)$ for a vertex $v_2 \in V_2$; that will then call $\mathcal{A}(\pi_H, \pi_H(1))$, which returns true, so $\mathcal{A}(\pi_H, v_2)$ returns false. Otherwise, the two trees are the same. In particular, the query complexity of $\mathcal{O}(\pi, v)$ (the number of recursive calls made) is at most that of $\mathcal{A}(\pi_H, v)$.

However, $\mathcal{O}$ also makes queries to M (particularly for non-edges), and we aim to upper-bound their number. We will charge this to the aforementioned *additional* recursive calls of $\mathcal{A}$; to that end, we will use the second condition of Definition 5.4.16.

Consider an execution of $\mathcal{O}(\pi, u)$ for any $u \in V$ (without counting its recursive calls). In the no-case (if $\mathcal{O}(\pi, u)$ returns false) it queries M for pairs $(\pi(i), u)$ for $i = 1, 2, ..., \ell$ for some ℓ, and $\mathcal{O}(\pi, \pi(\ell))$ is the first direct recursive call that returns true. In the yes-case (if $\mathcal{O}(\pi, u)$ returns true) it queries M for pairs $(\pi(i), u)$ for $i = 1, 2, ..., k-1$, where $k = \pi^{-1}(u)$ is the rank of u in π; denote $\ell := k - 1$ in the yes-case. Denote by L the rank of $\pi(\ell)$ in π_H (i.e., $\pi_H(L) = \pi(\ell)$). Then, we can say that $\mathcal{A}(\pi_H, u)$ must have made recursive calls to all neighbors of u in H with rank at most L in π_H. (In the yes-case, $\mathcal{A}$ queries all neighbors of lower rank before returning true; in the no-case, $\mathcal{A}(\pi_H, \pi_H(L)) = \mathcal{A}(\pi_H, \pi(\ell))$ is the first recursive call that returns true.) These neighbors in particular include all vertices in V_2 of at most that rank, because u is connected to all V_2 vertices in H. Thus, the number of calls to M made directly (without counting recursive calls) by an execution of $\mathcal{O}(\pi, u)$ for any $u \in V$ is

$$\begin{aligned}
\ell &= |\{\pi_H(1), ..., \pi_H(L)\} \cap V| \\
&\leq |\{\pi_H(1), ..., \pi_H(L)\} \cap V_2| \\
&\leq \text{number of } \textit{additional} \text{ recursive calls to } V_2 \text{ made directly by } \mathcal{A}(\pi_H, u),
\end{aligned}$$

where the first inequality follows since π_H is good. Now, summing this up over the entire execution tree of $\mathcal{O}(\pi, v)$, we get that the total number of calls to the adjacency matrix is at most the number the *additional* recursive calls to V_2 made by $\mathcal{A}$. Together with the above observation that the trees of $\mathcal{O}(\pi, v)$ and $\mathcal{A}(\pi_H, v)$ are the same when restricted to vertices in V, we have shown:

Lemma 5.4.18. *For any good $\pi_H \in S(V(H))$ and any $v \in V$, the running time of $\mathcal{O}(\pi, v)$ is upper-bounded by the oracle complexity of $\mathcal{A}(\pi_H, v)$ (up to a constant factor), i.e.,*

$$T_{\mathcal{O}}(\pi, v) \leq O(T_{\mathcal{A}}(\pi_H, v)),$$

where $\pi = \pi_H[V]$. □

We note that Theorem 5.4.13 allows us to bound the expectation of $T_{\mathcal{A}}$ over a random permutation $\pi_H \in S(V(H))$ (which may not be good) and a random query vertex $v \in V(H)$ (which may not be in V). However, V constitutes a constant fraction of $V(H)$, and a constant fraction of permutations are good; therefore, if the expectation over good permutations and over vertices in V was large, the entire expectation would also be large (up to a constant factor). Thus we have:

$$\begin{aligned}
\mathbf{E}_{v \in V,\, \pi \in S(V)}[T_{\mathcal{O}}(\pi, v)] &= \mathbf{E}_{v \in V,\, \text{good } \pi_H \in S(V(H))}[T_{\mathcal{O}}(\pi_H[V], v)] && \text{(by Lemma 5.4.17.2)} \\
&\leq O(1) \cdot \mathbf{E}_{v \in V,\, \text{good } \pi_H \in S(V(H))}[T_{\mathcal{A}}(\pi_H, v)] && \text{(by Lemma 5.4.18)} \\
&\leq O(1) \cdot \mathbf{E}_{v \in V(H),\, \pi_H \in S(V(H))}[T_{\mathcal{A}}(\pi_H, v)] && \text{(by Lemma 5.4.17.1)}
\end{aligned}$$

$$\leq O\left(\frac{|E(H)|}{|V(H)|}\right) \qquad \text{(by Theorem 5.4.13)}$$
$$= O(n).$$

This concludes the proof of Lemma 5.4.14. □

Remark 12. *We note that for a version of Algorithm 35 where we introduce caching (storing and reusing the results of each recursive call), there is a different and slightly simpler proof of Lemma 5.4.14 (ignoring* $\log n$ *factors in the running time), which was kindly pointed out to us by an anonymous reviewer. Specifically, in* $\mathcal{A}(\pi, u)$*, one can ensure that it takes time* $O(n/\deg(u) \cdot \log n)$ *to find each neighbor of* u*, with high probability for most permutations, via an application of the Chernoff bound. This is because if we consider a random permutation of vertex* u*'s row in the adjacency matrix, then, with high probability, any* $\Theta(n/\deg(u) \cdot \log n)$ *consecutive entries will contain at least one entry equal to 1. For permutations that do not satisfy this, we use a simple running time upper bound of* $O(n^2)$ *that holds when caching is used.*

We prefer to give a tight (up to $O(1)$ *factors) analysis for the* $\mathcal{O}$*-oracle (tight as it clearly runs in* $\Omega(n)$ *time). This is in the spirit of the analysis of [176], whose bound of* $1 + |E|/|V|$ *(Theorem 5.4.13) is exactly tight and does not rely on caching.*

Algorithm to estimate the MIS size

Algorithm 35 by itself only gives an answer for a single vertex. To obtain an additive error of $\widetilde{O}(n/s)$, we iterate it by sampling $\widetilde{O}(s)$ vertices. This routine, Algorithm 36, is one of the two major building blocks of our final algorithm for MIS.

Algorithm 36: AlgAddMul(V, E, s, π)

```
1 let r := (27s/ε²) log(n)
2 for i = 1, ..., r do
3     sample a random vertex v from V
4     let X_i := O(π, v)                          ▷ invoke Algorithm 35
5 return (1 + ε/2) · ( (Σ_i X_i)/r · n + εn/(3s) )
```

We remark that we add the $\Theta(\varepsilon)$ terms to obtain single-sided additive error.

Lemma 5.4.19. *For any* $\varepsilon \in (0, 1)$ *and* $s \geq 1$*, Algorithm 36 (*AlgAddMul*), given a graph* (V, E) *with oracle access to its adjacency matrix, as well as a uniformly random permutation* π *of its vertices, with probability* $1 - 1/\operatorname{poly}(n)$ *reports a* $(1+\varepsilon, \varepsilon n/s)$*-multiplicative-additive approximation to the value* $|\mathrm{RGMIS}(\pi)|$ *and runs in expected* $\widetilde{O}(ns/\varepsilon^2)$ *time.*

Proof. First, we prove that the output of AlgAddMul is a $(1+\varepsilon, \varepsilon n/s)$-multiplicative-additive approximation to the value of $|\mathrm{RGMIS}(\pi)|$. Since v is chosen uniformly at random in Line 3 of AlgAddMul, we have

$$\mathbf{E}[X_i] = \Pr[X_i = 1] = \frac{|\mathrm{RGMIS}(\pi)|}{n}.$$

Let $X = \sum_{i=1}^{r} X_i$. Then,

$$\mathbf{E}[X] = \frac{r \cdot |\mathrm{RGMIS}(\pi)|}{n}.$$

Since X is the sum of r independent Bernoulli random variables, we can apply the Chernoff bound (Proposition 2.3.1), which implies

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

Thus, with probability of $1 - 1/\operatorname{poly}(n)$, it holds that

$$\begin{aligned}
\frac{nX}{r} &\in \frac{n(\mathbf{E}[X] \pm \sqrt{3\,\mathbf{E}[X]\log n})}{r} \\
&= \frac{n\,\mathbf{E}[X]}{r} \pm \frac{\sqrt{3n^2\,\mathbf{E}[X]\log n}}{r} \\
&= |\mathrm{RGMIS}(\pi)| \pm \sqrt{\frac{3n|\mathrm{RGMIS}(\pi)|\log n}{r}} && (\text{since } \mathbf{E}[X] = \frac{r\cdot|\mathrm{RGMIS}(\pi)|}{n}) \\
&= |\mathrm{RGMIS}(\pi)| \pm \sqrt{\frac{\varepsilon^2 n|\mathrm{RGMIS}(\pi)|}{9s}} && (\text{since } r = \frac{27s}{\varepsilon^2}\log(n)).
\end{aligned}$$

Let ϑ be the output of the algorithm. We consider two possible scenarios:

- $|\mathrm{RGMIS}(\pi)| \leq n/s$: In this case, we have

$$\begin{aligned}
\vartheta = \left(1+\frac{\varepsilon}{2}\right)\cdot\left(\frac{nX}{r} + \frac{\varepsilon n}{3s}\right) &\in \left(1+\frac{\varepsilon}{2}\right)\cdot\left(|\mathrm{RGMIS}(\pi)| + \frac{\varepsilon n}{3s} \pm \sqrt{\frac{\varepsilon^2 n|\mathrm{RGMIS}(\pi)|}{9s}}\right) \\
&= \left(1+\frac{\varepsilon}{2}\right)\cdot\left(|\mathrm{RGMIS}(\pi)| + \frac{\varepsilon n}{3s} \pm \frac{\varepsilon n}{3s}\right),
\end{aligned}$$

  where the last inequality follows by the assumption $|\mathrm{RGMIS}(\pi)| \leq n/s$. Thus,

$$|\mathrm{RGMIS}(\pi)| \leq \vartheta \leq (1+\varepsilon)\cdot|\mathrm{RGMIS}(\pi)| + \frac{\varepsilon n}{s}.$$

- $|\mathrm{RGMIS}(\pi)| > n/s$: In this case, we have

$$\begin{aligned}
\vartheta = \left(1+\frac{\varepsilon}{2}\right)\cdot\left(\frac{nX}{r} + \frac{\varepsilon n}{3s}\right) &\in \left(1+\frac{\varepsilon}{2}\right)\cdot\left(|\mathrm{RGMIS}(\pi)| + \frac{\varepsilon n}{3s} \pm \sqrt{\frac{\varepsilon^2 n|\mathrm{RGMIS}(\pi)|}{9s}}\right) \\
&= \left(1+\frac{\varepsilon}{2}\right)\cdot\left(|\mathrm{RGMIS}(\pi)| + \frac{\varepsilon n}{3s} \pm \frac{\varepsilon}{3}|\mathrm{RGMIS}(\pi)|\right)
\end{aligned}$$

  where we have the last inequality because $|\mathrm{RGMIS}(\pi)| > n/s$. Hence,

$$|\mathrm{RGMIS}(\pi)| \leq \vartheta \leq (1+\varepsilon)\cdot|\mathrm{RGMIS}(\pi)| + \frac{\varepsilon n}{s}.$$

In both cases, the output of AlgAddMul is $(1+\varepsilon, \varepsilon n/s)$-multiplicative-additive approximation to the value of $|\mathrm{RGMIS}(\pi)|$. Let $T(\pi)$ be the expected running time of AlgAddMul on permutation π. Since we are sampling r vertices with replacement, we have

$$T(\pi) = r \cdot \mathbf{E}_{v\in V}[T_{\mathcal{O}}(\pi, v)].$$

Therefore, the expected running time of the algorithm is

$$\begin{aligned}\mathbf{E}_\pi[T(\pi)] = \frac{\sum_\pi r \cdot \mathbf{E}_{v\in V}[T_{\mathcal{O}}(\pi, v)]}{n!} &= r \cdot \mathbf{E}_{\pi\in S(V), v\in V}[T_{\mathcal{O}}(\pi, v)] \\ &= O(r \cdot n) \\ &= \widetilde{O}(ns/\varepsilon^2),\end{aligned}$$

which completes the proof. □

Now we can state our main Algorithm 37 and prove Theorem 5.4.3, which we restate for convenience. Roughly, our main idea here is a meet-in-the-middle approach between estimating the MIS size using Algorithm 36, which works well if the MIS is large, and building an MIS explicitly, which can be done if the MIS is small.

Algorithm 37: AlgMul(V, E)

```
π := uniformly random permutation of V
s := √n
CurrentMISSize := 0
Active[v] := 1 for all v ∈ V
j := 0
while j < n and CurrentMISSize < s do
    j := j + 1
    if Active[π(j)] = 1 then
        CurrentMISSize := CurrentMISSize + 1                ▷ add π(j) to the MIS
        for v ∈ V do
            if v = π(j) or (π(j), v) ∈ M then
                Active[v] := 0
if CurrentMISSize < s then
    return CurrentMISSize                                   ▷ we have explicitly built the entire MIS
else
    V' := {v ∈ V : Active[v] = 1}
    π' := π[V'] (π restricted to V')
    return CurrentMISSize + AlgAddMul(V', E, s, π')
```

Theorem 5.4.3. *For any $\varepsilon \in (0, 1)$ there is an algorithm (Algorithm 37) that, given a graph (V, E) with oracle access to its adjacency matrix, with high probability reports a multiplicative $(1 + \varepsilon)$-approximation to the value $|\mathrm{RGMIS}(\pi)|$ for some permutation π of V and runs in $\widetilde{O}(n^{3/2}/\varepsilon^2)$ time.*

We remark that our algorithm generates a random permutation $\pi \in S(V)$ (or $O(\log n)$ such permutations to obtain the runtime bound with high probability) and estimates $|\mathrm{RGMIS}(\pi)|$ for that permutation π; it does not attempt to estimate $\mathbf{E}_\pi[|\mathrm{RGMIS}(\pi)|]$.

Proof. First, we prove that the output of the algorithm is a $(1+\varepsilon)$-approximation of the value of $|\mathrm{RGMIS}(\pi)|$. If the condition on Algorithm 37 of the algorithm holds, then the algorithm explicitly built the entire MIS,

and the output is exactly equal to the size of RGMIS(π). Now, suppose that the condition on Algorithm 37 of the algorithm does not hold. Thus, CurrentMISSize $= s = \sqrt{n}$. Note that

$$|\text{RGMIS}(\pi)| = \sqrt{n} + |\text{RGMIS}(\pi[V'])|. \tag{5.15}$$

Let ϑ be the output of the algorithm. Hence, we have

$$\begin{aligned}
\vartheta &= \sqrt{n} + \text{AlgAddMul}(V', E, s, \pi') \\
&\geq \sqrt{n} + |\text{RGMIS}(\pi[V'])| && \text{(by Lemma 5.4.19)} \\
&= |\text{RGMIS}(\pi)| && \text{(by (5.15)).}
\end{aligned}$$

On the other hand,

$$\begin{aligned}
\vartheta &= \sqrt{n} + \text{AlgAddMul}(V', E, s, \pi') \\
&\leq \sqrt{n} + (1+\varepsilon) \cdot |\text{RGMIS}(\pi[V'])| + \frac{\varepsilon n}{s} && \text{(by Lemma 5.4.19)} \\
&= \sqrt{n} + (1+\varepsilon) \cdot |\text{RGMIS}(\pi[V'])| + \varepsilon\sqrt{n} && \text{(since } s = \sqrt{n}\text{)} \\
&= (1+\varepsilon) \cdot \left(\sqrt{n} + |\text{RGMIS}(\pi[V'])|\right) \\
&= (1+\varepsilon) \cdot |\text{RGMIS}(\pi)| && \text{(by (5.15)),}
\end{aligned}$$

which completes the proof for the approximation ratio.

Regarding the running time, the while loop in the algorithm runs in $O(n^{3/2})$ time, as in each step it either skips over an inactive vertex, which happens at most n times, or processes all vertices in $O(n)$ time; the latter happens at most $s = \sqrt{n}$ times. Furthermore, the relative order of the vertices in V' has no influence on the execution of the algorithm before it calls AlgAddMul on the last line; thus π' is a uniformly random permutation of V'. Therefore, by Lemma 5.4.19, the expected running time of AlgAddMul called on the last line is $\widetilde{O}(n^{3/2}/\varepsilon^2)$.

To obtain a high-probability bound on the running time, we execute $O(\log n)$ instances of the algorithm in parallel and stop as soon as the first instance completes. By applying Markov's inequality we conclude that each individual instance terminates within $\widetilde{O}(n^{3/2}/\varepsilon^2)$ time with constant probability. Consequently, with high probability, at least one of these instances finishes within $\widetilde{O}(n^{3/2}/\varepsilon^2)$ time. This completes the proof. □

Chapter 6

Conclusion and Open Questions

This thesis studies sublinear-time algorithms for graph problems, with a particular focus on the maximum matching problem. We approached this problem from both an algorithmic and a complexity-theoretic perspective. On the algorithmic side, we designed several new algorithms that improve the best-known results in terms of approximation ratio and running time, providing a better understanding of how well the size of a maximum matching can be estimated with only limited access to the input graph. On the complexity side, we established lower bounds that highlight the inherent trade-offs between approximation ratio and running time, thereby clarifying the fundamental limitations of sublinear time algorithms. We further applied sublinear matching techniques to several classical problems, including the Traveling Salesman Problem, Steiner Tree, Steiner Forest, and Earth Mover's Distance, obtaining new results and improving previous bounds. These applications demonstrate the broader reach of the techniques developed in this work beyond the matching problem itself.

We conclude this thesis with several important open problems related to the model and questions studied here, which we hope will guide future research on both the power and limitations of sublinear time algorithms.

Open Problem 1. *For graphs with maximum degree Δ, is it possible to estimate the size of maximum matching within a $(1-\varepsilon)$ approximation factor in time complexity of $\Delta^{O(1/\varepsilon)}$? Alternatively, is it possible to prove a lower bound of $\Delta^{\Omega(1/\varepsilon^2)}$ for any such approximation algorithm?*

The current fastest algorithm for this problem is due to [176], achieving a $(1-\varepsilon)$-approximation in time $\Delta^{O(1/\varepsilon^2)}$. The best known lower bound is discussed in Section 4.3, where we prove that any algorithm estimating the maximum matching size within a factor of $(1-\varepsilon)$ requires $\Delta^{\Omega(1/\varepsilon)}$ time.

Open Problem 2. *Is it possible to estimate the size of a maximum matching with an approximation ratio strictly larger than $1/2$ (by some constant) in $\widetilde{O}(n)$ time?*

Currently, two algorithms slightly improve over the $1/2$ approximation (Section 3.1, Section 3.2). The algorithm in Section 3.1 achieves a $(1/2+\varepsilon)$-approximation for a small constant ε in time $O(n^{1+f(\varepsilon)})$, where $f(\varepsilon)$ is a function that depends only on ε. The algorithm in Section 3.2 achieves a 0.51-approximation in $\widetilde{O}(n\sqrt{n})$ time.

Open Problem 3. *Can one design an algorithm that estimates the size of a maximum matching with an approximation ratio significantly larger than $1/2$ in time substantially faster than $O(n^2)$? i.e., can one achieve a $2/3$-approximation in $O(n\sqrt{n})$ time?*

Currently, all known algorithms either achieve an approximation ratio of $1/2$ or only slightly better than $1/2$ (Section 3.1, Section 3.2), or their running time is close to $O(n^2)$ [57].

Open Problem 4. *Can one estimate the cost of the metric TSP in $n^{2-\Omega(1)}$ time with an approximation ratio of $2-\varepsilon$ for some constant $\varepsilon > 0$?*

In Section 5.2, we show how one can obtain a better-than-2 approximation in sublinear time for special cases of the $(1,2)$-TSP and graphic TSP. Moreover, achieving a 2-approximation for the metric TSP is possible in sublinear time, due to the work of [75] on the metric MST.

Open Problem 5. *Is it possible to obtain a constant-factor estimate of the cost of the metric Steiner Forest in $n^{2-\Omega(1)}$ time?*

In Section 5.4, we show that there exists an algorithm that achieves an $O(\log k)$-approximation in sublinear time for the metric Steiner Forest problem, where k denotes the number of terminals.

Bibliography

[1] Amir Abboud and Virginia Vassilevska Williams. "Popular Conjectures Imply Strong Lower Bounds for Dynamic Problems". In: *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*. 2014, pp. 434–443.

[2] Anna Adamaszek, Matthias Mnich, and Katarzyna Paluch. "New Approximation Algorithms for (1, 2)-TSP". In: *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*. Vol. 107. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 9:1–9:14.

[3] Ajit Agrawal, Philip Klein, and R Ravi. "When Trees Collide: An Approximation Algorithm for the Generalized Steiner Problem on Networks". In: *SIAM Journal on Computing* 24.3 (1995), pp. 440–456.

[4] Ali Ahmadi, Iman Gholami, MohammadTaghi Hajiaghayi, Peyman Jabbarzade, and Mohammad Mahdavi. "Breaking a Long-Standing Barrier: 2-ε Approximation for Steiner Forest". In: *2025 IEEE 66th Annual Symposium on Foundations of Computer Science (FOCS)*. 2025.

[5] Nir Ailon, Moses Charikar, and Alantha Newman. "Aggregating inconsistent information: ranking and clustering". In: *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*. Ed. by Harold N. Gabow and Ronald Fagin. ACM, 2005, pp. 684–693. DOI: 10.1145/1060590.1060692. URL: https://doi.org/10.1145/1060590.1060692.

[6] Yeganeh Alimohammadi, Persi Diaconis, Mohammad Roghani, and Amin Saberi. "Sequential importance sampling for estimating expectations over the space of perfect matchings". In: *The Annals of Applied Probability* 33.2 (2023), pp. 999–1033.

[7] Noga Alon, László Babai, and Alon Itai. "A fast and simple randomized parallel algorithm for the maximal independent set problem". In: *Journal of algorithms* 7.4 (1986), pp. 567–583.

[8] Noga Alon, Eldar Fischer, Michael Krivelevich, and Mario Szegedy. "Efficient Testing of Large Graphs". In: *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*. 1999, pp. 656–666.

[9] 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*. 2012, pp. 1132–1139.

[10] Alexandr Andoni and Hengjie Zhang. "Sub-quadratic $(1+\varepsilon)$-approximate Euclidean Spanners, with Applications". In: *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2023, pp. 98–112.

[11] 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* ().

[12] Rubi Arviv, Lily Chung, Reut Levi, and Edward Pyne. "Improved Local Computation Algorithms for Constructing Spanners". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques* (2023).

[13] Itai Ashlagi, Jiale Chen, Mohammad Roghani, and Amin Saberi. "Stable Matching with Interviews". In: *16th Innovations in Theoretical Computer Science Conference, ITCS 2025, January 7-10, 2025, Columbia University, New York, NY, USA*. Ed. by Raghu Meka. Vol. 325. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025, 12:1–12:19. DOI: 10.4230/LIPICS.ITCS.2025.12. URL: https://doi.org/10.4230/LIPIcs.ITCS.2025.12.

[14] Sepehr Assadi. "Tight space-approximation tradeoff for the multi-pass streaming set cover problem". In: *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on principles of database systems*. 2017, pp. 321–335.

[15] 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)*. Ed. by Nikhil Bansal, Emanuela Merelli, and James Worrell. Vol. 198. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 19:1–19:13.

[16] Sepehr Assadi and Aaron Bernstein. "Towards a Unified Theory of Sparsification for Matching Problems". In: *2nd Symposium on Simplicity in Algorithms, SOSA 2019, January 8-9, 2019, San Diego, CA, USA*. Ed. by Jeremy T. Fineman and Michael Mitzenmacher. Vol. 69. OASIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 11:1–11:20.

[17] 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*. SIAM, 2019, pp. 767–786.

[18] Sepehr Assadi, Sanjeev Khanna, and Peter Kiss. "Improved bounds for fully dynamic matching via ordered ruzsa-szemeredi graphs". In: *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2025, pp. 2971–2990.

[19] Sepehr Assadi, Sanjeev Khanna, and Yang Li. "The Stochastic Matching Problem with (Very) Few Queries". In: *ACM Trans. Econ. Comput.* 7.3 (Sept. 2019).

[20] Sepehr Assadi, Sanjeev Khanna, and Yang Li. "The Stochastic Matching Problem: Beating Half with a Non-Adaptive Algorithm". In: *Proceedings of the 2017 ACM Conference on Economics and Computation, EC '17, Cambridge, MA, USA, June 26-30, 2017*. Ed. by Constantinos Daskalakis, Moshe Babaioff, and Hervé Moulin. ACM, 2017, pp. 99–116.

[21] Sepehr Assadi, Sanjeev Khanna, and Yang Li. "Tight bounds for single-pass streaming complexity of the set cover problem". In: *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*. 2016, pp. 698–711.

[22] 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*. Ed. by Timothy M. Chan. SIAM, 2019, pp. 1919–1936. DOI: 10.1137/1.9781611975482.116. URL: https://doi.org/10.1137/1.9781611975482.116.

[23] Baruch Awerbuch, Yossi Azar, and Yair Bartal. "On-line generalized Steiner problem". In: *Theoretical Computer Science* 324.2-3 (2004), pp. 313–324.

[24] Amir Azarmehr and Soheil Behnezhad. "Robust Communication Complexity of Matching: EDCS Achieves 5/6 Approximation". In: *arXiv preprint arXiv:2305.01070* (2023).

[25] Amir Azarmehr, Soheil Behnezhad, and Mohammad Roghani. "Bipartite Matching in Massive Graphs: A Tight Analysis of EDCS". In: *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL: https://openreview.net/forum?id=EDEISRmi6X.

[26] Amir Azarmehr, Soheil Behnezhad, and Mohammad Roghani. "Fully Dynamic Matching: -Approximation in Polylog Update Time". In: *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024*. Ed. by David P. Woodruff. SIAM, 2024, pp. 3040–3061. DOI: 10.1137/1.9781611977912.109. URL: https://doi.org/10.1137/1.9781611977912.109.

[27] Amir Azarmehr, Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. "Tight Pair Query Lower Bounds for Matching and Earth Mover's Distance". In: *Proceedings of 66th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2025, Sydney, Australia, December 14-17, 2025, To Appear* (2025).

[28] Étienne Bamas, Marina Drygala, and Andreas Maggiori. "An Improved Analysis of Greedy for Online Steiner Forest". In: *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2022, pp. 3202–3229.

[29] Surender Baswana, Manoj Gupta, and Sandeep Sen. "Fully Dynamic Maximal Matching in $O(logn)$ Update Time (Corrected Version)". In: *SIAM J. Comput.* 47.3 (2018), pp. 617–650.

[30] MohammadHossein Bateni, Hossein Esfandiari, and Vahab Mirrokni. "Almost optimal streaming algorithms for coverage problems". In: *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*. 2017, pp. 13–23.

[31] S. Behnezhad, M. Derakhshan, M. Hajiaghayi, C. Stein, and M. Sudan. "Fully Dynamic Maximal Independent Set with Polylogarithmic Update Time". In: *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*. Los Alamitos, CA, USA: IEEE Computer Society, Nov. 2019, pp. 382–405.

[32] Soheil Behnezhad. "Dynamic Algorithms for Maximum Matching Size". In: *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*. 2023, pp. 129–162.

[33] Soheil Behnezhad. "Improved Analysis of EDCS via Gallai-Edmonds Decomposition". In: *CoRR* abs/2110.05746 (2021). arXiv: 2110.05746.

[34] Soheil Behnezhad. "Time-Optimal Sublinear Algorithms for Matching and Vertex Cover". In: *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*. IEEE, 2021, pp. 873–884.

[35] Soheil Behnezhad, Moses Charikar, Weiyun Ma, and Li-Yang Tan. "Almost 3-Approximate Correlation Clustering in Constant Rounds". In: *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*. IEEE, 2022, pp. 720–731. DOI: 10.1109/FOCS54457.2022.00074. URL: https://doi.org/10.1109/FOCS54457.2022.00074.

[36] Soheil Behnezhad, Mahsa Derakhshan, and MohammadTaghi Hajiaghayi. "Stochastic matching with few queries: (1-ε) approximation". In: *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*. Ed. by Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy. ACM, 2020, pp. 1111–1124.

[37] Soheil Behnezhad and Alma Ghafari. "Fully dynamic matching and ordered ruzsa-szemerédi graphs". In: *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2024, pp. 314–327.

[38] Soheil Behnezhad and Sanjeev Khanna. "New Trade-Offs for Fully Dynamic Matching via Hierarchical EDCS". In: *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 3529–3566. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611977073.140.

[39] 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*. Ed. by Shuchi Chawla. SIAM, 2020, pp. 2492–2508.

[40] Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. "Approximating Maximum Matching Requires Almost Quadratic Time". In: *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, Canada, To Appear* (2024).

[41] Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. "Local Computation Algorithms for Maximum Matching: New Lower Bounds". In: *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2023, pp. 2322–2335.

[42] Soheil Behnezhad, Mohammad Roghani, and Aviad Rubinstein. "Sublinear Time Algorithms and Complexity of Approximate Maximum Matching". In: *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*. STOC 2023. Orlando, FL, USA: Association for Computing Machinery, 2023, pp. 267–280. DOI: 10.1145/3564246.3585231.

[43] Soheil Behnezhad, Mohammad Roghani, Aviad Rubinstein, and Amin Saberi. "Beating Greedy Matching in Sublinear Time". In: *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*. Ed. by Nikhil Bansal and Viswanath Nagarajan. SIAM, 2023, pp. 3900–3945.

[44] Soheil Behnezhad, Mohammad Roghani, Aviad Rubinstein, and Amin Saberi. "Sublinear Algorithms for TSP via Path Covers". In: *CoRR* abs/2301.05350 (2023). DOI: 10.48550/ARXIV.2301.05350. arXiv: 2301.05350. URL: https://doi.org/10.48550/arXiv.2301.05350.

[45] Lorenzo Beretta and Aviad Rubinstein. "Approximate Earth Mover's Distance in Truly-Subquadratic Time". In: *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*. Ed. by Bojan Mohar, Igor Shinkar, and Ryan O'Donnell. ACM, 2024, pp. 47–58.

[46] Piotr Berman and Marek Karpinski. "8/7-approximation algorithm for (1, 2)-TSP". In: *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*. ACM Press, 2006, pp. 641–648.

[47] Aaron Bernstein. "Improved Bounds for Matching in Random-Order Streams". In: *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. Ed. by Artur Czumaj, Anuj Dawar, and Emanuela Merelli. Vol. 168. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 12:1–12:13. ISBN: 978-3-95977-138-2.

[48] Aaron Bernstein, Aditi Dudeja, and Zachary Langley. "A framework for dynamic matching in weighted graphs". In: *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. 2021, pp. 668–681.

[49] Aaron Bernstein, Sebastian Forster, and Monika Henzinger. "A Deamortization Approach for Dynamic Spanner and Dynamic Maximal Matching". In: *ACM Trans. Algorithms* 17.4 (Oct. 2021).

[50] 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 '16. Arlington, Virginia: Society for Industrial and Applied Mathematics, 2016, pp. 692–711. ISBN: 9781611974331.

[51] Aaron Bernstein and Cliff Stein. "Fully Dynamic Matching in Bipartite Graphs". In: *Automata, Languages, and Programming*. Ed. by Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 167–179. ISBN: 978-3-662-47672-7.

[52] Joseph Bertrand. "Solution d'un probleme". In: *CR Acad. Sci. Paris* 105.1887 (1887), p. 369.

[53] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F Italiano. "Deterministic fully dynamic data structures for vertex cover and matching". In: *SIAM Journal on Computing* 47.3 (2018), pp. 859–887.

[54] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. "Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time". In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2017, pp. 470–489.

[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*. Ed. by Daniel Wichs and Yishay Mansour. ACM, 2016, pp. 398–411.

[56] Sayan Bhattacharya and Peter Kiss. "Deterministic rounding of dynamic fractional matchings". In: *arXiv preprint arXiv:2105.01615* (2021).

[57] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. "Dynamic $(1+\varepsilon)$-Approximate Matching Size in Truly Sublinear Update Time". In: *64th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2023, to appear*. 2023. arXiv: 2302.05030.

[58] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. "Dynamic $(1+\varepsilon)$-Approximate Matching Size in Truly Sublinear Update Time". In: *CoRR* abs/2302.05030 (2023). arXiv: 2302.05030.

[59] Sayan Bhattacharya, Peter Kiss, and Thatchaphol Saranurak. "Sublinear Algorithms for $(1.5+\varepsilon)$-Approximate Matching". In: *Proceedings of the 55th ACM Symposium on Theory of Computing, STOC 2023, Orlando, Florida, to appear*. 2023.

[60] Sayan Bhattacharya, Peter Kiss, Thatchaphol Saranurak, and David Wajc. "Dynamic Matching with Better-than-2 Approximation in Polylogarithmic Update Time". In: *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*. 2023, pp. 100–128.

[61] 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*. Ed. by Guy E. Blelloch and Maurice Herlihy. ACM, 2012, pp. 308–317.

[62] Sebastian Brandt, Manuela Fischer, and Jara Uitto. "Matching and MIS for Uniformly Sparse Graphs in the Low-Memory MPC Model". In: *CoRR* abs/1807.05374 (2018). arXiv: 1807.05374. URL: http://arxiv.org/abs/1807.05374.

[63] Jaroslaw Byrka, Fabrizio Grandoni, Thomas Rothvoß, and Laura Sanita. "An improved LP-based approximation for Steiner tree". In: *Proceedings of the forty-second ACM symposium on Theory of computing*. 2010, pp. 583–592.

[64] Moses Charikar and Shay Solomon. "Fully dynamic almost-maximal matching: Breaking the polynomial worst-case time barrier". In: *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2018.

[65] Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. "Approximating the Minimum Spanning Tree Weight in Sublinear Time". In: *SIAM J. Comput.* 34.6 (2005), pp. 1370–1379.

[66] Shiri Chechik and Tianyi Zhang. "Fully Dynamic Maximal Independent Set in Expected Poly-Log Update Time". In: *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*. Ed. by David Zuckerman. IEEE Computer Society, 2019, pp. 370–381. DOI: 10.1109/FOCS.2019.00031. URL: https://doi.org/10.1109/FOCS.2019.00031.

[67] Chandra Chekuri, Rhea Jain, Sepideh Mahabadi, and Ali Vakilian. *Streaming Algorithms for Network Design*. 2025. arXiv: 2503.00712 [cs.DS]. URL: https://arxiv.org/abs/2503.00712.

[68] 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)*. 2020, 30:1–30:19.

[69] Yu Chen, Sanjeev Khanna, and Zihan Tan. "Query Complexity of the Metric Steiner Tree Problem". In: *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2023, pp. 4893–4935.

[70] Yu Chen, Sanjeev Khanna, and Zihan Tan. "Sublinear Algorithms and Lower Bounds for Estimating MST and TSP Cost in General Metrics". In: *50th International Colloquium on Automata, Languages, and Programming, ICALP 2023, July 10-14, 2023, Paderborn, Germany*. LIPIcs 261 (2023). Ed. by Kousha Etessami, Uriel Feige, and Gabriele Puppis, 37:1–37:16. DOI: 10.4230/LIPICS.ICALP.2023.37. URL: https://doi.org/10.4230/LIPIcs.ICALP.2023.37.

[71] Avery Ching, Sergey Edunov, Milica Kabiljo, Dimitrios Logothetis, and S. Muthukrishnan. "One Trillion Edges: Graph Processing at Facebook-Scale". In: *Proceedings of the VLDB Endowment*. Vol. 8. 12. 2015, pp. 1804–1815. URL: https://www.vldb.org/pvldb/vol8/p1804-ching.pdf.

[72] Miroslav Chlebík and Janka Chlebíková. "The Steiner tree problem on graphs: Inapproximability results". In: *Theoretical Computer Science* 406.3 (2008), pp. 207–214.

[73] Nicos Christofides. *Worst-case analysis of a new heuristic for the travelling salesman problem*. Tech. rep. Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

[74] Artur Czumaj, Shaofeng H-C Jiang, Robert Krauthgamer, and Pavel Veselỳ. "Streaming algorithms for geometric steiner forest". In: *ACM Transactions on Algorithms* 20.4 (2024), pp. 1–38.

[75] Artur Czumaj and Christian Sohler. "Estimating the Weight of Metric Minimum Spanning Trees in Sublinear Time". In: *SIAM J. Comput.* 39.3 (2009), pp. 904–922.

[76] Artur Czumaj and Christian Sohler. "Estimating the weight of metric minimum spanning trees in sublinear-time". In: *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*. 2004, pp. 175–183.

[77] Søren Dahlgaard. "On the Hardness of Partially Dynamic Graph Problems and Connections to Diameter". In: *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*. Ed. by Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi. Vol. 55. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 48:1–48:14.

[78] Mina Dalirrooyfard, Konstantin Makarychev, and Slobodan Mitrović. "Pruned Pivot: correlation clustering algorithm for dynamic, parallel, and local computation models". In: *Proceedings of the 41st International Conference on Machine Learning*. ICML'24. Vienna, Austria: JMLR.org, 2024.

[79] Erik D Demaine, Piotr Indyk, Sepideh Mahabadi, and Ali Vakilian. "On streaming and communication complexity of the set cover problem". In: *Distributed Computing: 28th International Symposium, DISC 2014, Austin, TX, USA, October 12-15, 2014. Proceedings 28*. Springer. 2014, pp. 484–498.

[80] Mahsa Derakhshan, Mohammad Roghani, Mohammad Saneian, and Tao Yu. "A Simple Analysis of Ranking in General Graphs". In: *2026 Symposium on Simplicity in Algorithms (SOSA)*. to appear. SIAM, 2026.

[81] Mahsa Derakhshan, Mohammad Roghani, Mohammad Saneian, and Tao Yu. "Improved Approximation for Ranking on General Graphs". In: *Proceedings of the 2026 ACM-SIAM Symposium on Discrete Algorithms, SODA 2026*. to appear. SIAM, 2026.

[82] Jack Edmonds. "Paths, trees, and flowers". In: *Canadian Journal of mathematics* 17 (1965), pp. 449–467.

[83] Yuval Emek and Adi Rosén. "Semi-streaming set cover". In: *ACM Transactions on Algorithms (TALG)* 13.1 (2016), pp. 1–22.

[84] Pál Erdős and Alfréd Rényi. "On the existence of a factor of degree one of a connected random graph". In: *Acta Math. Acad. Sci. Hungar* 17.359-368 (1966), p. 192.

[85] Paul Erdős and Alfred Renyi. "On random matrices". In: *Magyar Tud. Akad. Mat. Kutató Int. Közl* 8 (1964), pp. 455–461.

[86] Hossein Esfandiari, MohammadTaghi Hajiaghayi, and Morteza Monemizadeh. "Finding Large Matchings in Semi-Streaming". In: *IEEE International Conference on Data Mining Workshops, ICDM Workshops 2016, December 12-15, 2016, Barcelona, Spain*. Ed. by Carlotta Domeniconi, Francesco Gullo, Francesco Bonchi, Josep Domingo-Ferrer, Ricardo Baeza-Yates, Zhi-Hua Zhou, and Xindong Wu. IEEE Computer Society, 2016, pp. 608–614.

[87] Matthew Fahrbach, Zhiyi Huang, Runzhou Tao, and Morteza Zadimoghaddam. "Edge-Weighted Online Bipartite Matching". In: *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*. Ed. by Sandy Irani. IEEE, 2020, pp. 412–423.

[88] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. "On graph problems in a semi-streaming model". In: *Theor. Comput. Sci.* 348.2-3 (2005), pp. 207–216.

[89] Moran Feldman and Ariel Szarf. "Maximum Matching Sans Maximal Matching: A New Approach for Finding Maximum Matchings in the Data Stream Model". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2022, September 19-21, 2022, University of Illinois, Urbana-Champaign, USA (Virtual Conference)*. Ed. by Amit Chakrabarti and Chaitanya Swamy. Vol. 245. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 33:1–33:24.

[90] Manuela Fischer and Andreas Noever. "Tight Analysis of Parallel Randomized Greedy MIS". In: *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*. 2018, pp. 2152–2160.

[91] Manuela Fischer and Andreas Noever. "Tight Analysis of Parallel Randomized Greedy MIS". In: *ACM Trans. Algorithms* 16.1 (2020), 6:1–6:13.

[92] Alan Frieze and Boris Pittel. "Perfect Matchings in Random Graphs with Prescribed Minimal Degree". In: *Mathematics and Computer Science III*. Ed. by Michael Drmota, Philippe Flajolet, Danièle Gardy, and Bernhard Gittenberger. Basel: Birkhäuser Basel, 2004, pp. 95–132. ISBN: 978-3-0348-7915-6.

[93] Buddhima Gamlath, Michael Kapralov, Andreas Maggiori, Ola Svensson, and David Wajc. "Online Matching with General Arrivals". In: *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*. Ed. by David Zuckerman. IEEE Computer Society, 2019, pp. 26–37.

[94] Ruiquan Gao, Mohammad Roghani, Aviad Rubinstein, and Amin Saberi. "Hardness of Approximate Sperner and Applications to Envy-Free Cake Cutting". In: *65th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2024, Chicago, IL, USA, October 27-30, 2024*. IEEE, 2024, pp. 1294–1331. DOI: 10.1109/FOCS61266.2024.00085. URL: https://doi.org/10.1109/FOCS61266.2024.00085.

[95] Naveen Garg, Anupam Gupta, Stefano Leonardi, and Piotr Sankowski. "Stochastic analyses for online combinatorial optimization problems". In: *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*. Ed. by Shang-Hua Teng. 2008, pp. 942–951.

[96] Mohsen Ghaffari. "An improved distributed algorithm for maximal independent set". In: *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*. SIAM. 2016, pp. 270–277.

[97] Mohsen Ghaffari. "Local Computation of Maximal Independent Set". In: *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*. 2022, pp. 438–449.

[98] Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrović, 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*. 2018, pp. 129–138.

[99] Mohsen Ghaffari, Christoph Grunau, and Ce Jin. "Improved MPC Algorithms for MIS, Matching, and Coloring on Trees and Beyond". In: *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*. Ed. by Hagit Attiya. Vol. 179. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 34:1–34:18. DOI: 10.4230/LIPICS.DISC.2020.34. URL: https://doi.org/10.4230/LIPIcs.DISC.2020.34.

[100] Mohsen Ghaffari and Bernhard Haeupler. "A Time-Optimal Randomized Parallel Algorithm for MIS". In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2021, pp. 2892–2903. DOI: 10.1137/1.9781611976465.172. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611976465.172.

[101] 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*. SIAM, 2019, pp. 1636–1653.

[102] Laleh Ghalami and Daniel Grosu. "A parallel approximation algorithm for the steiner forest problem". In: *2022 30th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. IEEE. 2022, pp. 47–54.

[103] Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. "A Randomized Rounding Approach to the Traveling Salesman Problem". In: *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*. IEEE Computer Society, 2011, pp. 550–559.

[104] Edgar N Gilbert and Henry O Pollak. "Steiner minimal trees". In: *SIAM Journal on Applied Mathematics* 16.1 (1968), pp. 1–29.

[105] Michel X Goemans and David P Williamson. "A general approximation technique for constrained forest problems". In: *SIAM Journal on Computing* 24.2 (1995), pp. 296–317.

[106] Oded Goldreich and Dana Ron. "A Sublinear Bipartiteness Tester for Bunded Degree Graphs". In: *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*. 1998, pp. 289–298.

[107] Oded Goldreich and Dana Ron. "Property Testing in Bounded Degree Graphs". In: *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*. 1997, pp. 406–415.

[108] Oded Goldreich and Luca Trevisan. "Three Theorems Regarding Testing Graph Properties". In: *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*. 2001, pp. 460–469.

[109] Fabrizio Grandoni, Chris Schwiegelshohn, Shay Solomon, and Amitai Uzrad. "Maintaining an edcs in general graphs: Simpler, density-sensitive and with worst-case time bounds". In: *Symposium on Simplicity in Algorithms (SOSA)*. SIAM. 2022, pp. 12–23.

[110] Martin Groß, Anupam Gupta, Amit Kumar, Jannik Matuschke, Daniel R Schmidt, Melanie Schmidt, and José Verschae. "A Local-Search Algorithm for Steiner Forest". In: *9th Innovations in Theoretical Computer Science Conference (ITCS 2018)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. 2018.

[111] Christoph Grunau, Slobodan Mitrovic, Ronitt Rubinfeld, and Ali Vakilian. "Improved Local Computation Algorithm for Set Cover via Sparsification". In: *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*. Ed. by Shuchi Chawla. SIAM, 2020, pp. 2993–3011. DOI: 10.1137/1.9781611975994.181. URL: https://doi.org/10.1137/1.9781611975994.181.

[112] Anupam Gupta, MohammadTaghi Hajiaghayi, and Amit Kumar. "Stochastic Steiner tree with non-uniform inflation". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer. 2007, pp. 134–148.

[113] Anupam Gupta and Amit Kumar. "Greedy algorithms for Steiner forest". In: *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*. 2015, pp. 871–878.

[114] Anupam Gupta and Amit Kumar. "Online steiner tree with deletions". In: *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SIAM. 2014, pp. 455–467.

[115] Anupam Gupta and Krzysztof Onak. "Sublinear.info Open Problem 71: Metric TSP Cost Approximation". In: Available at https://sublinear.info/index.php?title=Open_Problems:71. 2016.

[116] Anupam Gupta and Martin Pál. "Stochastic Steiner trees without a root". In: *Automata, Languages and Programming: 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005. Proceedings 32*. Springer. 2005, pp. 1051–1063.

[117] Manoj Gupta and Richard Peng. "Fully dynamic (1+ e)-approximate matchings". In: *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. IEEE. 2013, pp. 548–557.

[118] Guru Prashanth Guruganesh and Sahil Singla. "Online Matroid Intersection: Beating Half for Random Arrival". In: *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*. Ed. by Friedrich Eisenbrand and Jochen Könemann. Vol. 10328. Lecture Notes in Computer Science. Springer, 2017, pp. 241–253.

[119] Sariel Har-Peled, Piotr Indyk, Sepideh Mahabadi, and Ali Vakilian. "Towards Tight Bounds for the Streaming Set Cover Problem". In: *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*. Ed. by Tova Milo and Wang-Chiew Tan. ACM, 2016, pp. 371–383. DOI: 10.1145/2902251.2902287. URL: https://doi.org/10.1145/2902251.2902287.

[120] David G. Harris. "Distributed Local Approximation Algorithms for Maximum Matching in Graphs and Hypergraphs". In: *SIAM Journal on Computing* 49.4 (2020), pp. 711–746. DOI: 10.1137/19M1279241. eprint: https://doi.org/10.1137/19M1279241.

[121] Johan Hastad. "Clique is Hard to Approximate Within $n^{1-\varepsilon}$". In: *Proceedings of 37th Conference on Foundations of Computer Science*. IEEE. 1996, pp. 627–636.

[122] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. "Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture". In: *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*. 2015, pp. 21–30.

[123] John E. Hopcroft and Richard M. Karp. "An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs". In: *SIAM J. Comput.* 2.4 (1973), pp. 225–231.

[124] Zhiyi Huang, Binghui Peng, Zhihao Gavin Tang, Runzhou Tao, Xiaowei Wu, and Yuhao Zhang. "Tight Competitive Ratios of Classic Matching Algorithms in the Fully Online Model". In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*. Ed. by Timothy M. Chan. SIAM, 2019, pp. 2875–2886.

[125] Makoto Imase and Bernard M Waxman. "Dynamic Steiner tree problem". In: *SIAM Journal on Discrete Mathematics* 4.3 (1991), pp. 369–384.

[126] Piotr Indyk, Sepideh Mahabadi, Ronitt Rubinfeld, Jonathan Ullman, Ali Vakilian, and Anak Yodpinyanee. "Fractional set cover in the streaming model". In: *20th International Workshop on Approximation Algorithms for Combinatorial Optimization Problem (APPROX 2017)*. 2017.

[127] Piotr Indyk, Sepideh Mahabadi, Ronitt Rubinfeld, Ali Vakilian, and Anak Yodpinyanee. "Set Cover in Sub-linear Time". In: *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*. Ed. by Artur Czumaj. SIAM, 2018, pp. 2467–2486. DOI: 10.1137/1.9781611975031.158. URL: https://doi.org/10.1137/1.9781611975031.158.

[128] Ce Jin, Michael Kapralov, Sepideh Mahabadi, and Ali Vakilian. "Streaming Algorithms for Connectivity Augmentation". In: *51st International Colloquium on Automata, Languages, and Programming (ICALP 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. 2024, pp. 93–1.

[129] Kumar Joag-Dev and Frank Proschan. "Negative Association of Random Variables with Applications". In: *The Annals of Statistics* 11.1 (1983), pp. 286–295. ISSN: 00905364. (Visited on 11/12/2023).

[130] Sagar Kale and Sumedh Tirodkar. "Maximum Matching in Two, Three, and a Few More Passes Over Graph Streams". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*. Ed. by Klaus Jansen, José D. P. Rolim, David Williamson, and Santosh S. Vempala. Vol. 81. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 15:1–15:21.

[131] Michael Kapralov. "Space Lower Bounds for Approximating Maximum Matching in the Edge Arrival Model". In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*. Ed. by Dániel Marx. SIAM, 2021, pp. 1874–1893.

[132] 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*. 2020, pp. 1753–1772.

[133] Anna R. Karlin, Nathan Klein, and Shayan Oveis Gharan. "A (slightly) improved approximation algorithm for metric TSP". In: *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*. ACM, 2021, pp. 32–45.

[134] Richard M Karp. "Reducibility among combinatorial problems". In: *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[135] Marek Karpinski and Richard Schmied. "On Approximation Lower Bounds for TSP with Bounded Metrics". In: *Electron. Colloquium Comput. Complex.* (2012), p. 8.

[136] Alam Khursheed and K. M. Lai Saxena. "Positive dependence in multivariate distributions". In: *Communications in Statistics - Theory and Methods* 10.12 (1981), pp. 1183–1196.

[137] Peter Kiss. "Deterministic Dynamic Matching in Worst-Case Update Time". In: *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*. Ed. by Mark Braverman. Vol. 215. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 94:1–94:21. ISBN: 978-3-95977-217-4.

[138] D. König. "Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre". ger. In: *Mathematische Annalen* 77 (1916), pp. 453–465.

[139] 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*. Ed. by Igor Potapov, Paul G. Spirakis, and James Worrell. Vol. 117. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 74:1–74:16.

[140] 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*. Ed. by Anupam Gupta, Klaus Jansen, José D. P. Rolim, and Rocco A. Servedio. Vol. 7408. Lecture Notes in Computer Science. Springer, 2012, pp. 231–242.

[141] Christian Konrad and Kheeran K. Naidu. "On Two-Pass Streaming Algorithms for Maximum Bipartite Matching". In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2021, August 16-18, 2021, University of Washington, Seattle, Washington, USA (Virtual Conference)*. Ed. by Mary Wootters and Laura Sanita. Vol. 207. LIPIcs. 2021, 19:1–19:18. DOI: 10.4230/LIPICS.APPROX/RANDOM.2021.19. URL: https://doi.org/10.4230/LIPIcs.APPROX/RANDOM.2021.19.

[142] Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. "Filtering: a method for solving graph problems in MapReduce". In: *Proceedings of the Twenty-Third Annual ACM Symposium on Parallelism in Algorithms and Architectures*. SPAA '11. San Jose, California, USA: Association for Computing Machinery, 2011, pp. 85–94. ISBN: 9781450307437.

[143] Christoph Lenzen and Reut Levi. "A centralized local algorithm for the sparse spanning graph problem". In: *45th International Colloquium on Automata, Languages, and Programming*. Schloss Dagstuhl. 2018, pp. 1–47.

[144] Reut Levi, Ronitt Rubinfeld, and Anak Yodpinyanee. "Local Computation Algorithms for Graphs of Non-constant Degrees". In: *Algorithmica* 77.4 (2017), pp. 971–994.

[145] Michael Luby. "A simple parallel algorithm for the maximal independent set problem". In: *Proceedings of the seventeenth annual ACM symposium on Theory of computing*. 1985, pp. 1–10.

[146] Sepideh Mahabadi, Mohammad Roghani, and Jakub Tarnawski. "A 0.51-Approximation of Maximum Matching in Sublinear $n^{1.5}$ Time". In: *arXiv preprint arXiv:2506.01669* (2025).

[147] Sepideh Mahabadi, Mohammad Roghani, Jakub Tarnawski, and Ali Vakilian. *Sublinear Metric Steiner Forest via Maximal Independent Set*. 2025. arXiv: 2510.11627 [cs.DS]. URL: https://arxiv.org/abs/2510.11627.

[148] Sepideh Mahabadi, Mohammad Roghani, Jakub Tarnawski, and Ali Vakilian. "Sublinear Metric Steiner Tree via Improved Bounds for Set Cover". In: *16th Innovations in Theoretical Computer Science Conference (ITCS 2025)*. Ed. by Raghu Meka. Vol. 325. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025, 74:1–74:24. ISBN: 978-3-95977-361-4. DOI: 10.4230/LIPIcs.ITCS.2025.74. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITCS.2025.74.

[149] Nicole Megow, Martin Skutella, José Verschae, and Andreas Wiese. "The power of recourse for online MST and TSP". In: *SIAM Journal on Computing* 45.3 (2016), pp. 859–880.

[150] Matthias Mnich and Tobias Mömke. "Improved integrality gap upper bounds for traveling salesperson problems with distances one and two". In: *Eur. J. Oper. Res.* 266.2 (2018), pp. 436–457.

[151] Tobias Mömke and Ola Svensson. "Approximating Graphic TSP by Matchings". In: *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*. IEEE Computer Society, 2011, pp. 560–569.

[152] Marcin Mucha. "13/9-approximation for Graphic TSP". In: *29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, February 29th - March 3rd, 2012, Paris, France*. Vol. 14. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012, pp. 30–41.

[153] Ofer Neiman and Shay Solomon. "Simple deterministic algorithms for fully dynamic maximal matching". In: *ACM Transactions on Algorithms (TALG)* 12.1 (2015), pp. 1–15.

[154] 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*. 2008, pp. 327–336.

[155] Krzysztof Onak. "Round Compression for Parallel Graph Algorithms in Strongly Sublinear Space". In: *CoRR* abs/1807.08745 (2018). arXiv: 1807.08745.

[156] Krzysztof Onak, Dana Ron, Michal Rosen, and Ronitt Rubinfeld. "A Near-Optimal Sublinear-Time Algorithm for Approximating the Minimum Vertex Cover Size". In: *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*. 2012, pp. 1123–1131.

[157] 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*. Ed. by Leonard J. Schulman. ACM, 2010, pp. 457–464.

[158] Christos H. Papadimitriou and Mihalis Yannakakis. "The Traveling Salesman Problem with Distances One and Two". In: *Math. Oper. Res.* 18.1 (1993), pp. 1–11.

[159] Michal Parnas and Dana Ron. "Approximating the Minimum Vertex Cover in Sublinear Time and a Connection to Distributed Algorithms". In: *Theor. Comput. Sci.* 381.1-3 (2007), pp. 183–196.

[160] Merav Parter, Ronitt Rubinfeld, Ali Vakilian, and Anak Yodpinyanee. "Local computation algorithms for spanners". In: *arXiv preprint arXiv:1902.08266* (2019).

[161] Tristan Pollner, Mohammad Roghani, Amin Saberi, and David Wajc. "Improved Online Contention Resolution for Matchings and Applications to the Gig Economy". In: *EC '22: The 23rd ACM Conference on Economics and Computation, Boulder, CO, USA, July 11 - 15, 2022*. Ed. by David M. Pennock, Ilya Segal, and Sven Seuken. ACM, 2022, pp. 321–322. DOI: 10.1145/3490486.3538295. URL: https://doi.org/10.1145/3490486.3538295.

[162] Omer Reingold and Shai Vardi. "New techniques and tighter bounds for local computation algorithms". In: *J. Comput. Syst. Sci.* 82.7 (2016), pp. 1180–1200.

[163] Gabriel Robins and Alexander Zelikovsky. "Tighter bounds for graph Steiner tree approximation". In: *SIAM Journal on Discrete Mathematics* 19.1 (2005), pp. 122–134.

[164] Mohammad Roghani, Amin Saberi, and David Wajc. "Beating the Folklore Algorithm for Dynamic Matching". In: *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*. Ed. by Mark Braverman. Vol. 215. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 111:1–111:23.

[165] Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. "Fast Local Computation Algorithms". In: *Innovations in Computer Science - ICS 2011, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*. 2011, pp. 223–238.

[166] Barna Saha and Lise Getoor. "On maximum coverage in the streaming model & application to multi-topic blog-watch". In: *Proceedings of the 2009 siam international conference on data mining*. SIAM. 2009, pp. 697–708.

[167] Alexander Schrijver et al. *Combinatorial optimization: polyhedra and efficiency*. Vol. 24. 2. Springer, 2003.

[168] András Sebö and Jens Vygen. "Shorter tours by nicer ears: 7/5-Approximation for the graph-TSP, 3/2 for the path version, and 4/3 for two-edge-connected subgraphs". In: *Comb.* 34.5 (2014), pp. 597–629.

[169] Anatoliy I Serdyukov. "O nekotorykh ekstremal'nykh obkhodakh v grafakh". In: *Upravlyayemyye sistemy* 17 (1978), pp. 76–79.

[170] 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*. Ed. by Irit Dinur. IEEE Computer Society, 2016, pp. 325–334.

[171] David Wajc. "Negative association: definition, properties, and applications". In: *Manuscript, available from https://goo. gl/j2ekqM* (2017).

[172] David Wajc. "Rounding Dynamic Matchings against an Adaptive Adversary". In: *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2020. Chicago, IL, USA: Association for Computing Machinery, 2020, pp. 194–207. ISBN: 9781450369794.

[173] W. Allen Whitworth. "Arrangements of m things of one sort and n things of another sort under certain conditions of priority". In: *Messenger of Mathematics* 8 (1878), pp. 105–114.

[174] Wikipedia contributors. *Wikipedia: Size of Wikipedia*. `https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia`. Accessed: 2025-10-28. Oct. 2025.

[175] Andrew Chi-Chih Yao. "Probabilistic Computations: Toward a Unified Measure of Complexity (Extended Abstract)". In: *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*. IEEE Computer Society, 1977, pp. 222–227.

[176] 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*. Ed. by Michael Mitzenmacher. ACM, 2009, pp. 225–234.

[177] Mark Zuckerberg. *Meta CEO Zuckerberg says Instagram has grown to 3 billion monthly active users*. `https://www.reuters.com/business/meta-ceo-zuckerberg-says-instagram-has-grown-3-billion-monthly-active-users-2025-09-24/`. Accessed: 2025-10-28. Sept. 2025.

[178] David Zuckerman. "Linear degree extractors and the inapproximability of max clique and chromatic number". In: *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*. 2006, pp. 681–690.